

Artificial General Intelligence

**Second Conference
on Artificial General Intelligence**
AGI 2009
Arlington, Virginia, USA
March 6-9, 2009

Editors
Ben Goertzel, Pascal Hitzler,
Marcus Hutter

Ben Goertzel, Pascal Hitzler, Marcus Hutter (Editors)

Artificial General Intelligence

Proceedings of the Second Conference on Artificial General
Intelligence, AGI 2009, Arlington, Virginia, USA, March 6-9, 2009



Amsterdam-Paris

ISBN: 978-90-78677-24-6

Advances in Intelligent Systems Research

volume 8

Artificial General Intelligence

Volume Editors

Ben Goertzel

Novamente & Biomind LLC
1405 Bernerd Place
Rockville MD 20851, USA
Email: ben@goertzel.org
www: <http://www.goertzel.org>

Pascal Hitzler

AIFB
Universität Karlsruhe (TH)
Kollegiengebäude am Ehrenhof 11.40, Room 224
Englerstraße 11 76131 Karlsruhe, Germany
Email: pascal@pascal-hitzler.de
www: <http://www.pascal-hitzler.de>

Marcus Hutter

CSL@RSISE and SML@NICTA
Australian National University
Room B259, Building 115
Corner of North and Daley Road
Canberra ACT 0200, Australia
Email: marcus.hutter@gmx.net
www: <http://www.hutter1.net>

This book is part of the series *Advances in Intelligent Systems Research* (ISSN: 1951-6851) published by Atlantis Press.

Aims and scope of the series

During the past decade computer science research in understanding and reproducing human intelligence has expanded from the more traditional approaches like psychology, logics and artificial intelligence into multiple other areas, including neuroscience research. Moreover, new results in biology, chemistry, (surface) physics and gene technology, but also in network technology are greatly affecting current research in computer science, including the development of intelligent systems. At the same time, computer science's new results are increasingly being applied in these fields allowing for important cross-fertilisations. This series aims at publishing proceedings from all disciplines dealing with and affecting the issue of understanding and reproducing intelligence in artificial systems. Also, the series is open for publications concerning the application of intelligence in networked or any other environment and the extraction of meaningful data from large data sets.

Research fields covered by the series include: * Fuzzy sets * Machine learning * Autonomous agents * Evolutionary systems * Robotics and autonomous systems * Semantic web, incl. web services, ontologies and grid computing * Biological systems * Artificial Intelligence, incl. knowledge representation, logics * Neural networks * Constraint satisfaction * Computational biology * Information sciences * Computer vision, pattern recognition * Computational neuroscience * Datamining, knowledge discovery and modelling for e.g. life sciences.

© ATLANTIS PRESS, 2009
<http://www.atlantis-press.com>

ISBN: 978-90-78677-24-6

This book is published by Atlantis Press, scientific publishing, Paris, France.

All rights reserved for this book. No part of this book may be reproduced, translated, stored or transmitted in any form or by any means, including electronic, mechanical, photocopying, recording or otherwise, without prior permission from the publisher.

Atlantis Press adheres to the creative commons policy, which means that authors retain the copyright of their article.

Preface

Artificial General Intelligence (AGI) research focuses on the original and ultimate goal of AI – to create broad human-like and transhuman intelligence, by exploring all available paths, including theoretical and experimental computer science, cognitive science, neuroscience, and innovative interdisciplinary methodologies. Due to the difficulty of this task, for the last few decades the majority of AI researchers have focused on what has been called *narrow AI* – the production of AI systems displaying intelligence regarding specific, highly constrained tasks. In recent years, however, more and more researchers have recognized the necessity – and feasibility – of returning to the original goals of the field. Increasingly, there is a call for a transition back to confronting the more difficult issues of *human level intelligence* and more broadly *artificial general intelligence*.

The Conference on Artificial General Intelligence is the only major conference series devoted wholly and specifically to the creation of AI systems possessing general intelligence at the human level and ultimately beyond. Its second installation, AGI-09, in Arlington, Virginia, March 6-9, 2009, attracted 67 paper submissions, which is a substantial increase from the previous year. Of these submissions, 33 (i.e., 49%) were accepted as full papers for presentation at the conference. Additional 13 papers were included as position papers. The program also included a keynote address by Jürgen Schmidhuber on *The New AI*, a post-conference workshop on *The Future of AI*, and a number of pre-conference tutorials on various topics related to AGI.

Producing such a highly profiled program would not have been possible without the support of the community. We thank the organising committee members for their advise and their help in all matters of actually preparing and running the event. We thank the program committee members for a very smooth review process and for the high quality of the reviews – despite the fact that due to the very high number of submissions the review load per PC member was considerably higher than originally expected. And we thank all participants for submitting and presenting interesting and stimulating work, which is the key ingredient needed for a successful conference. We also gratefully acknowledge the support of a number of sponsors:

- Artificial General Intelligence Research Institute
- Association for the Advancement of Artificial Intelligence (AAAI)
- The University of Memphis
- Enhanced Education (Platinum Sponsor and Keynote Address)
- KurzweilAI.net (Gold Sponsor and Kurzweil Best AGI Paper 2009)
- Joi Labs (Silver Sponsor)
- Machine Intelligence Lab at the University of Tennessee (Silver Sponsor)
- Novamente LLC (Silver Sponsor)

March 2009

Ben Goertzel (Conference Chair)

Pascal Hitzler, Marcus Hutter (Program Committee Chairs)

Conference Organization

Chairs

Ben Goertzel (Conference Chair)	Novamente LLC, USA
Stan Franklin (Honorary Chair)	University of Memphis, USA
Pascal Hitzler (Programme Chair)	University of Karlsruhe, Germany
Marcus Hutter (Programme Chair)	Australian National University, Australia

Programme Committee

Itamar Arel	University of Tennessee, Knoxville, USA
Sebastian Bader	Rostock University, Germany
Eric Baum	Baum Research Enterprises, USA
Mark H. Bickhard	Lehigh University, USA
David Dowe	Monash University, Australia
Hugo de Garis	Xiamen University, China
Wlodek Duch	Nicolaus Copernicus University, Poland
Phil Goetz	J. Craig Venter Institute, USA
Artur Garcez	City University London, UK
Ben Goertzel	Novamente LLC, USA
Marco Gori	University of Siena, Italy
J. Storrs Hall	Institute for Molecular Manufacturing, USA
Hassan Mahmud	Australian National University, Australia
Cliff Joslyn	Pacific Northwest National Laboratory, USA
Kai-Uwe Kühnberger	University of Osnabrück, Germany
Christian Lebiere	Carnegie Mellon University, USA
Shane Legg	University of Lugano, Switzerland
Moshe Looks	Google Research, USA
Jose Hernandez Orallo	Politechnical University of Valencia, Spain
Jan Poland	ABB Research, Zurich, Switzerland
Sebastian Rudolph	University of Karlsruhe, Germany
Scott Sanner	NICTA, Australia
Ute Schmid	University of Bamberg, Germany
Jürgen Schmidhuber	IDSIA, Switzerland
Angela Schwing	University of Osnabrück, Germany
Ray Solomonoff	Oxbridge Research, Cambridge, USA
Frank van der Velde	Leiden University, The Netherlands
Karin Verspoor	University of Colorado, Denver, USA
Marco Wiering	University of Utrecht, The Netherlands
Mary-Anne Williams	University of Technology, Sydney, Australia

Organizing Committee

Tsvi Achler
Itamar Arel
Sam S. Adams
Wlodek Duch
Sandra S. Hall
Pascal Hitzler
Marcus Hutter
Bruce Klein
Stephen Reed

U. of Illinois Urbana-Champaign, USA
University of Tennessee Knoxville, USA
IBM Research, USA
Nicolaus Copernicus University, Poland
AT&T Research, USA
University of Karlsruhe, Germany
Australian National University, Australia
Novamente LLC, USA
Texai.org, USA

External Reviewers

Bobby Coop
Benjamin Johnston
Scott Livingston

Table of Contents

Full Articles.

Project to Build Programs that Understand	1
<i>Eric Baum</i>	
CHS-Soar: Introducing Constrained Heuristic Search to the Soar Cognitive Architecture	7
<i>Sean Bittle, Mark Fox</i>	
In Search of Computational Correlates of Artificial Qualia	13
<i>Antonio Chella, Salvatore Gaglio</i>	
Combining Analytical and Evolutionary Inductive Programming	19
<i>Neil Crossley, Emanuel Kitzelmann, Martin Hofmann, Ute Schmid</i>	
The China-Brain Project: Report on the First Six Months	25
<i>Hugo de Garis</i>	
AGI Preschool: A Framework for Evaluating Early-Stage Human-like AGIs	31
<i>Ben Goertzel, Stephan Bugaj</i>	
Pointer Semantics with Forward Propagation	37
<i>Sujata Ghosh, Benedikt Löwe, Sanchit Saraf</i>	
The Role of Logic in AGI Systems: Towards a Lingua Franca for General Intelligence	43
<i>Helmar Gust, Ulf Krumnack, Angela Schwering, Kai-Uwe Kuehnberger</i>	
The robotics path to AGI using Servo Stacks	49
<i>John Hall</i>	
A Unifying Framework for Analysis and Evaluation of Inductive Programming Systems	55
<i>Martin Hofmann, Emanuel Kitzelmann, Ute Schmid</i>	
Feature Markov Decision Processes	61
<i>Marcus Hutter</i>	
Feature Dynamic Bayesian Networks	67
<i>Marcus Hutter</i>	
Economic Attention Networks: Associative Memory and Resource Allocation for General Intelligence	73
<i>Matthew Ikle, Joel Pitt, Ben Goertzel, George Sellman</i>	

A formal framework for the symbol grounding problem	79
<i>Benjamin Johnston, Mary-Anne Williams</i>	
A Cognitive Map for an Artificial Agent	85
<i>Unmesh Kurup, B. Chandrasekaran</i>	
Claims and Challenges in Evaluating Human-Level Intelligent Systems . . .	91
<i>John Laird, Robert Wray, Robert Marinier, Pat Langley</i>	
Extending Cognitive Architectures with Mental Imagery	97
<i>Scott Lathrop, John Laird</i>	
A Comparative Approach to Understanding General Intelligence: Predicting Cognitive Performance in an Open-ended Dynamic Task	103
<i>Christian Lebiere, Cleotilde Gonzalez, Walter Warwick</i>	
Incorporating Planning and Reasoning into a Self-Motivated, Communicative Agent	108
<i>Daphne Liu, Lenhart Schubert</i>	
Program Representation for General Intelligence	114
<i>Moshe Looks, Ben Goertzel</i>	
Consciousness in Human and Machine: A Theory and Some Falsifiable Predictions	120
<i>Richard Loosemore</i>	
Hebbian Constraint on the Resolution of the Homunculus Fallacy Leads to a Network that Searches for Hidden Cause-Effect Relationships	126
<i>Andras Lorincz</i>	
Everyone's a Critic: Memory Models and Uses for an Artificial Turing Judge	132
<i>Joseph MacInnes, Blair Armstrong, Dwayne Pare, George Cree, Steve Joordens</i>	
Unsupervised Segmentation of Audio Speech Using the Voting Experts Algorithm	138
<i>Matthew Miller, Peter Wong, Alexander Stoytchev</i>	
Parsing PCFG within a General Probabilistic Inference Framework	144
<i>Arthi Murugesan, Nicholas Cassimatis</i>	
Self-Programming: Operationalizing Autonomy	150
<i>Eric Nivel, Kristinn R. Thorisson</i>	
Bootstrap Dialog: A Conversational English Text Parsing and Generation System	156
<i>Stephen Reed</i>	

Analytical Inductive Programming as a Cognitive Rule Acquisition Devise	162
<i>Ute Schmid, Martin Hofmann, Emanuel Kitzelmann</i>	
Human and Machine Understanding of Natural Language Character Strings	168
<i>Peter G. Tripodes</i>	
Embodiment: Does a laptop have a body?	174
<i>Pei Wang</i>	
Case-by-Case Problem Solving	180
<i>Pei Wang</i>	
What Is Artificial General Intelligence? Clarifying The Goal For Engineering And Evaluation	186
<i>Mark Waser</i>	
Integrating Action and Reasoning through Simulation	192
<i>Samuel Wintermute</i>	

Position Statements.

Neuroscience and AI Share the Same Elegant Mathematical Trap	198
<i>Tsvi Achler, Eyal Amir</i>	
Relevance Based Planning: Why its a Core Process for AGI	200
<i>Eric Baum</i>	
General Intelligence and Hypercomputation	202
<i>Selmer Bringsjord</i>	
Stimulus processing in autonomously active cognitive systems	204
<i>Claudius Gros</i>	
Distribution of Environments in Formal Measures of Intelligence	206
<i>Bill Hibbard</i>	
The Importance of Being Neural-Symbolic – A Wilde Position	208
<i>Pascal Hitzler, Kai-Uwe Kühnberger</i>	
Improving the Believability of Non-Player Characters in Simulations	210
<i>Jere Miles, Rahman Tashakkori</i>	
Understanding the Brain’s Emergent Properties	212
<i>Don Miner, Marc Pickett, Marie desJardins</i>	
Why BICA is Necessary for AGI	214
<i>Alexei Samsonovich</i>	

Importing Space-time Concepts Into AGI	216
<i>Eugene Surowitz</i>	
HELEN – Using Brain Regions and Mechanisms for Story Understanding And Modeling Language As Human Behavior	218
<i>Robert Swaine</i>	
Holistic Intelligence: Transversal Skills & Current Methodologies	220
<i>Kristinn R. Thorisson, Eric Nivel</i>	
Achieving Artificial General Intelligence Through Peewee Granularity ...	222
<i>Kristinn R. Thorisson, Eric Nivel</i>	
Author Index	224

Project to Build Programs that Understand

Eric B. Baum

Baum Research Enterprises
41 Allison Road
Princeton NJ 08540
ebaum@fastmail.fm

Abstract

This extended abstract outlines a project to build computer programs that understand. Understanding a domain is defined as the ability to rapidly produce computer programs to deal with new problems as they arise. This is achieved by building a CAD tool that collaborates with human designers who guide the system to construct code having certain properties. The code respects Occam's Razor, interacts with a domain simulation, and is informed by a number of mechanisms learned from introspection, the coding employed in biology, and analysis.

Introduction

This extended abstract outlines a project to build computer programs that understand.

Definition 1: Understanding a domain is defined as the ability to rapidly produce programs to deal with new problems as they arise in the domain.

This definition is proposed to model human understanding (which I hold to be of the domain of the natural world and extensions we have made into related domains such as mathematics), and also accurately describes biological evolution, which thus may be said to *understand* what it is doing.

A program that can rapidly produce programs for new problems is rather unlike the standard kinds of programs that we usually see. They deal with contingencies that had been previously planned for. We will describe both a new programming method, and new style of program, in order to accomplish our task.

Hypothesis 1: The property of understanding in thought and evolution arises through Occam's Razor. (Baum, 2004, 2007)

By finding a concise genome that solves a vast number problems, evolution built a program comprising a hierarchic collection of modules that generalize-- that know how to rapidly produce programs to solve new problems. That genetic program also encodes inductive

biases that construct a similar hierarchic collection of modules within your mind that rapidly produce programs to solve new problems.

As a result of conciseness and the structure of the programs it requires, random mutations in the genome lead to meaningful programs a sizable fraction of the time. For example, a point mutation of a fly's genome may create a fly with an extra pair of wings instead of halteres, or a fly with an extra pair of legs instead of antennae (Kirschner and Gerhart 2005). These are recognizably *meaningful* in that they are functional in an interesting way. Evolution thus rapidly searches over meaningful outcomes looking for one that solves new problems¹. That is, according to our definition, evolution understands.

Another example within evolution is the coding of limbs (Kirschner and Gerhart 2005). If asked to code up a limb, human software engineers might attempt to separately specify the structure of the bones, the muscles, the nerves, and the blood vessels. If done that way, the program would be huge, like standard engineering specs for machines such as a jet aircraft, and it couldn't adapt to new uses, and it couldn't evolve. A favorable mutation in the bone structure would have to be matched by one in each of the other systems to survive. Proponents of Intelligent Design would have a point.

Instead, biology is much more concisely coded. The bones grow out in a concisely and hierarchically specified way. There is no separate detailed specification of exactly how the muscles, nerves, or blood vessels grow. The muscle cells perform a search, and attach themselves to the bones, and grow so as to be functional in context. A muscle cell that is being useful expands (which is why rowers have big hearts). The nerves seek out muscles, and learn how to be functional. The vascular system grows out in a search toward cells that scream for oxygen. As a result of this extremely concise encoding, if you take up new exercises, your muscles adapt and grow in appropriate ways, and if a mutation changes the genetic coding of the

¹Language facilitates thinking in similar fashion. You can come up with incredibly concise formulations of big new ideas, just a sentence or several in natural language, that grow out into detailed executables much like concise specifications in the genome result in interacting searches that robustly construct a limb, or like RBP finds a high level plan and refines it through a series of searches. (Baum, 2008b)

bone structure, everything else adapts to make a functional system, so mutations can easily explore meaningful possibilities.

The same kind of conciseness is present in the genetic coding for your mind. Enough initial structure is provided, also called inductive bias, that a series of search programs can build a hierarchic series of modules to solve new problems.

Previous attempts to produce understanding programs have mostly followed one of two paths. One path has been purely automatic methods, such as direct attempts to simulate evolution. This is hopeless because, while we may already have or may soon have computational resources comparable to those of the brain, we will never be able to compete with evolution-- which ran through some 10^{44} creatures (Smith, 2006, footnote 95) (furthermore each creature individually interacted with the world and is thus expensive to simulate). To evaluate other automatic attempts, ask yourself what is enforcing Occam nature and keeping the solution constrained enough to generalize to new problems. An approach that can too easily add new knowledge without adequate constraint can (and will) simply memorize examples that it sees, rather than generalizing to new problems. In my view, this is a common flaw in AI/AGI approaches.

A second approach has been to craft such a program by hand. I believe this is also hopeless for a variety of reasons. First, a wealth of experience in computational learning theory indicates that finding Occam representations in any interesting hypothesis space is NP-hard. If finding Occam software is NP-hard, it is no more likely to be susceptible to hand solution than other huge NP-hard problems. Second, a wealth of experience with solving AGI by hand indicates it is hard. Winograd's SHRDLU, for example, seemed like a particularly well crafted project, yet a few years later he threw up his hands and wrote a book explaining why crafting an understanding program is impossible (Winograd and Flores 1987). Third, when its structure is examined (say by introspection and/or attempting to program it) the program of mind (and also the program of biological development) seems to contain a large number of ingeniously crafted modules. At the very least, figuring out each of these is a PhD dissertation level project, and some of them may be much harder. Evolution, which understands what it is doing, applied massive computational resources in designing them. And the Occam codings evolution found may inherently be hard to understand (Baum, 2004 chap 6). Finally, the problems are hard both at the inner level (figuring out ingenious, concise, fast modules to solve subproblems) and at outer levels of organizing the whole program. Humans are simply not competent to write computer programs to deal with hyper-complex domains in a robust way.

Hand crafting has at times sought to benefit from introspection. A problem with this is that people do not have introspective access to the internals of the meaningful modules, which are hidden from introspection by information hiding. Consider, for example, when you

invoke Microsoft Word. You know why you are invoking it, and what you expect it to do, but you do not have much idea of its internal code. The same is true of meaningful internal modules within your mental program. However, we appeal (based on observation) to:

Hypothesis 2: People do have introspective access to a meaning-level (Baum 2007, 2004 chap 14), at which they call meaningful modules by name or other pointer.

I use the term *concept* to denote a name (word) or other mental construction that you might think that has some meaning, i.e. that corresponds to a recognizable function in the world, and *module* to denote the computational procedure that would be summoned to compute a concept. Thus we may think of concepts as names that invoke modules.

You can state in words why you call a given module, and you can give examples of concepts (for example, inputs and outputs of the associated module). This is incredibly powerful and detailed information, that we will use to get a huge jump on evolution.

Note that the problem of finding an appropriate module is distinct from that of finding an appropriate concept, so this dichotomy factors the problem of producing code for new problems. For example, executing a certain module might create in a simulation model a corresponding goal condition (namely, realize a concept). *Agents* denote modules that recognize patterns and then may take other computational actions, as in, for example (Baum and Durdanovic, 2000, 2002). In the present work, patterns are typically recognized in a simulation domain or annotated simulation domain or model, that is seeing particular annotations may be critical to the recognition of the pattern. The recognition will frequently involve taking a series of simulated actions on the model and then checking for a pattern or condition (as in (Baum and Durdanovic, 2000)). Agents when activated will frequently post annotations on a model, which is how much of perception proceeds.

The solution we propose, by which to construct understanding programs, is thus based on the following objectives and principles.

(1) Humans can not write the detailed code. As much as possible it must be automated, but, the system must also support as much guidance as possible from humans.

(2) The goal is to produce a program that will rapidly assemble programs to solve new problems. To that end, we have to ask what kinds of modules can be provided that can be flexibly assembled. We also have to ask what kinds of modules can be provided for guiding the assembly process, for example by finding high level plans that are then refined. We propose mechanisms for these tasks. Assembling programs for new tasks from primitive level instructions without a detailed plan is prohibitively expensive. But if a program can be divided into half a dozen tasks, and a program for each them assembled by combining a handful of meaningful modules, the search becomes manageable. Thus we need building systems and

building blocks and means of adapting to new contexts.

(3) The reason understanding is possible is that the natural world (and relatedly, mathematics) have a very concise underlying structure that can be exploited to do useful computations. We embody this by providing domain simulations. The domain simulations typically live in 3 (or 2) Euclidean dimensions, plus a time dimension, and provide direct access to causal structure. All modules and agents interact with the domain simulation (in fact, with multiple copies, each agent may explore its own copy). Thus all thought may be model/image based in a way much more powerful than other systems of which we are aware. This grounds all our computations, that is to say, will allow us to choose modules that perform functions that are meaningful in the real world.

All agent or instruction calls are with respect to a particular domain simulation position. Everything is thus context dependent, and agents can communicate by taking actions on the internal model and by perceiving the model. A caching mechanism detects when a module recursively calls the same module in the identical position, and returns a default value at the inner call, and thus enables concise recursive coding without looping.

(4) Economics simulations as in Hayek (Baum and Durdanovic 2000, 2002; Baum 2004 chap 10) are used at multiple levels to implement the invisible hand and assign credit so that components of the program all have to contribute and survive in a competitive environment². This greatly contributes to conciseness. Such economies also have the property of efficiently caching the execution of modules at each level of abstraction, greatly speeding computation.

(5) Other encodings collectively called scaffolds are modeled on those discovered by evolution or perceived by introspection in ways to realize point (2) above. Scaffolds promote conciseness, and provide inductive bias for constructions. For example, scaffolds make great use of search programs modeled on those used in the development of limbs (or in computer chess (Baum 2007)).

We call the system Artificial Genie.

CAD Tool

The first step is to build a CAD tool with which humans collaborate in the construction of code. A number of module constructors are provided, which take as inputs such things as an instruction set out of which to build new programs, a fitness function and/or a definition of a state to

²The earliest use of which I'm aware of economics simulation for program evolution was (Holland 1985), who also invoked pattern perceiving agents. (Lenat 1982) also used related ideas. The Hayek model is cited as enforcing certain principles critical to proper functioning of economic evolutionary systems, and we also propose improved evolutionary economic systems with other features.

Copyright © 2008, The Second Conference on Artificial General Intelligence (agi-09.org). All rights reserved.

be achieved, and examples of a given concept, and return a module computing the concept. This is in principle straightforward to achieve by, for example, genetic programming³ (or, which we prefer, running a Hayek or other Economic Evolutionary System (EES)).

Once a module is constructed to compute a concept, the CAD tool also creates an instruction invoking the module, and makes that available for inclusion in instruction sets. Note that such instructions can be used in hand-written code, as well as fed to module constructors. So users are enabled to write code in terms of concepts, even though they may be unable to write the modules computing the concepts.

Note that, while above we said simulated evolution was too slow, what is generally done is to try to simulate the whole solution to a problem in one gulp. We are proposing instead to apply EES's or other module constructors to carefully chosen subproblems, and then build hierarchically upward. If the module construction fails, that is the module constructor is not able rapidly enough to construct a satisfactory module to compute the concept, the CAD tool also provides the opportunity to first construct other concepts which may be sub-concepts useful for computing the desired concept, the instructions for which can then be included in instruction sets.

The CAD tool will be used to improve itself until it can also learn from solved examples and natural language descriptions of the solution. (Baum, In prep).

Simulation Model

Our code interacts with a domain simulation that is analogous to mental imagery. This provides numerous critical functions. First, by encoding the underlying causal structure of the domain, the image allows modules to generalize, to encode meaningful function. For example, a simple physics simulation can describe what will happen in any kind of complex system or machine built of physical parts. Such are often easy to construct from ball and spring models (Johnston and Williams, 2008). All generalization ultimately derives from exploiting underlying structure, which is provided by the simulation.

Many AI programs are based on a set of logical axioms and logical reasoning, rather than incorporating a "physics" simulation. If you work that way, however, you can't readily generalize. Say I give you a new object, that you haven't seen before. Since you don't have axioms pertaining to that object, you have no understanding that lets you say anything about it. Logic just treats names (say of objects) as tokens, and if it doesn't recognize a particular token, it knows nothing about it. By contrast, if you have a physics simulation, you can work out what will happen when things are done to the object, when its rotated or dropped, just from its physical structure. This ability to

³A variety of module constructors may be provided suitable for certain tasks. For example, neural network-like module constructors may be useful for early vision.

generalize is essential to understanding.

Another problem with the logic approach is the frame problem. With everything in terms of axioms, you have a problem: if something is changed, what else is affected? Logically, there is no way to say, unless you have specific frame axioms, and then you have to do inference, which also may be computationally intractable. This classic problem paralyzed AI for many years. If you are working with a physical simulation, this problem is bypassed. You just propagate forward-- only things affected by causal chains change.

Interaction with a simulation allows the computational agents to be grounded, that is to say: to perform functions that are meaningful in the real world. Second, the domain simulation provides a medium by which different computational agents can communicate, since agents can both perceive (recognize patterns in) and take actions on the domain simulation. Third, agents can utilize the domain simulation to search to achieve goals. For example, they can achieve subgoals in the context of the current situation and the other agents. Robustness can be achieved by partitioning problems amongst interacting, concisely specified, goal-oriented agents. This is a key element of how biological development achieves robustness. (Development utilizes an analog medium rather than a domain simulation.) The domain simulation naturally biases in the spatial, temporal, causal structure that greatly facilitates dividing problems into parts that can be separately analyzed and then combined. Fourth, the domain simulation provides an intuitive and interactive mechanism for users (program designers) to input training examples. Training examples can be entered as configurations of the domain simulation (for example, small local regions representing some concept) or as worked examples (where, for example the program designer inputs the example as if playing a video game) (which may be accompanied by natural language description of what is being done).

Relevance Based Planning

The power of using domain simulations is demonstrated by our planning system. Our planner finds high level candidate solutions by examining the mental image, where each high level candidate solution is a path that would achieve a goal if a number of sub goals (counter-factuals) can be achieved. The planning system then orchestrates the collaboration of those subprograms that are relevant to achieve the sub goals. Subprograms or agents that are attempting to achieve sub goals perform look-ahead searches on the mental image, and these searches may produce patterns indicating other problems that were not previously anticipated, the perception of which invokes other agents. Because the whole process is grounded by interaction with the mental image, the system explores those subprogram calls that are relevant to achieving the goal. An example of the operation of the planning system

is discussed elsewhere in this volume (Baum, 2008a).

Like all the modules in Artificial Genie, the planner is supplied as an instruction within the CAD tool that can be incorporated into new modules or agents by automatic module constructors or by program designers. To our knowledge, this also is not a feature of any competing systems. This enables the construction of powerful agents that can invoke planning as part of their computation to achieve goals. Powerful programs can be constructed as compositions of agents by planning on the domain to break goals up into a series of sub goals, and then building agents to deal with the sub goals. An example of a Hayek constructing agents that invoke planning instructions to solve Sokoban problems was exhibited in (Schaul 2005).

Evolutionary Programming, Search, and Scaffolds

Consider a search to find a program to solve a problem. To do this by building the program from primitive instructions will often be prohibitively expensive. One needs appropriate macro instructions so that each discovery step is manageable size: no individual search is vast. Code discovery is only possible in bite sized portions. Scaffolds provide big chunks of the code, and break the full search down into a combination of manageable searches.

As discussed in section 1, this is largely how biological development is coded: as a series of tractable searches (each cell performs a search to produce its cyto-skeleton, mitosis invokes a search to build microtubules for organizing the chromosomes, the nerves perform a series of searches to enervate, the vascular system performs a separate search, and so on. (Kirschner and Gerhart 2005))

Coding in terms of constrained searches is an incredibly concise means of coding. As discussed in (Baum, 2004,2007) this is also the method used in computer chess. A handful of lines of code (encoding alpha-beta + evaluation function + quiescence) creates a search that accurately values a huge number of chess positions. Because the code for this is so concise, Occam's razor applies and it generalizes to almost all chess positions.

One simple form of scaffold supplies much of the search machinery, but requires that an evaluation function or goal condition for the search and/or a set of actions that the search will be over (which may generally be macro actions or agents) be supplied to use the scaffold for a particular problem. For example, the alpha-beta search from chess can be simply modified in this way into programs for other problems (like Othello) cf (Baum 2007). Another use of such search-scaffolds would be to compose them into larger programs in a way analogous to how development builds the body out of a collection of interacting searches. The use of search scaffolds should provide huge inductive bias, greatly speeding the production of programs for new problems. In other words, having such scaffolds will provide understanding of the domain.

Another form of scaffold is used in the evolutionary

construction of programs. The scaffold here is basically equivalent to a procedure, with slots (aka arguments) and possibly also annotations respective to some or all of the slots. One can build a program downward by starting with a scaffold, and then constructing subprograms to fit into each of the slots. The annotations may supply guidance as to independent examples to be provided for training the subprograms, or other guidance such as instruction sets to be provided to module constructors to build the subprograms out of.

The point here is, the naive way to be able to rapidly create a program for a new problem, is to have a set of macros from which the program for the new problem can be rapidly discovered because it is a 5 line program in terms of the macros. Another way, however, is to have a scaffold that builds all or part of the program by starting from the top and building down: discovering programs to fit into slots. The discovery of these subprograms may involve running an EES, or invoking another scaffold. And yet another way, is to have a scaffold like RBP that interacts with the mental image to build a high level plan and then to flesh it out into concrete code.

Evolutionary Economic Systems

The real economy is a powerful system in which billions of different economic actors, who don't even know each other and may not even share a common language, are organized to collaborate. A complex program is like an economy, with large numbers of parts that must work together. The usual process of constructing a complex program is like a command economy, with some human programmers sitting like the Politburo and attempting to fit everything together. As desired programs get more complex, it becomes harder and harder to integrate everything, and pieces are added that are counterproductive, and many opportunities for rationalization are missed.

Artificial Genie supplies an economic framework that motivates all the pieces by simple rules that implement the invisible hand, propagate price signals. Moreover, within the created environment of a simulated economy, major flaws that plague the real economy, such as the tragedy of the commons and theft and the dead-weight loss of taxation, are removed, leaving a perfect computational economy. Since it is made available as a module constructor, the economic framework can readily be applied at each level of the computational hierarchy, both as the simplest subprograms are built from instructions, and as the overall system is integrated, to ensure efficient and harmonious operation. So, a high level program will be composed of economically motivated agents, each of which itself may be a Hayek or other EES, composed of economically motivated agents, rather like the US economy is composed of corporations, each of which may have many subcontractors. This hierarchic construction is straightforward because each satisfactory module (often an

EES) that is found, is encapsulated by the CAD tool as an instruction, usable for constructing later agents and programs.

Many problems can only be solved after performing a search. For example, deciding on the best move in a chess position typically requires a search; likewise planning problems, optimization problems, even problems of perception such as recognizing some pattern in one's environment or sensory input, almost all require searches. These searches will usually be intractably large if not somehow focused. Extant automatic programming methods do not produce programs that search and hence are limited in their application. Artificial Genie incorporates an economic structure to automatically construct search-programs, programs that when presented with a task to solve, perform a search to find the solution.

Artificial Genie supports the building of perceptual systems consisting of a collection of agents that perceive patterns and post names of recognized concepts, that may enable other perception or action. Such patterns will typically be perceived in the simulation domain, and the perception may involve taking one or more simulated actions on the domain position followed by verifying a condition or checking a pattern. The syntactical analysis of (Hobbs 2004) is a model for such posting of concept labels that enable other conclusions, except that we have added economic structure and interaction with a simulation model. The whole program, from perception through decision, is economically evolved to be concise, efficient, and meaningful, so that such perceptual agents will only survive when they are later relied upon by agents earning reward from the world.

The economics naturally incorporates a smart caching mechanism, by which subprograms that initially require extensive computation, become fast and automatic in most circumstances. Concepts are frequently given a definition in terms of other modules, rather than an explicit algorithm to compute it. Frequently a concept may be defined in terms of a condition that is to hold or be achieved (which may in fact involve achieving multiple sub-conditions). A module constructor then must construct a program that achieves that condition (or recognizes when it holds). The module constructor will typically produce a search-EES. This search-EES caches into agents, methods that work rapidly in certain classes of positions. When the module is called, the search-EES cache will retrieve a solution rapidly, provided one is available. However, if the search-EES fails (ie no agent bids, or no solution is found) the system may return to the definition, and attempt to find a solution by a more extensive search (creating new agents) or other methods. If a solution is then found, the search-EES may be extended with more agents, caching the new solution. This is analogous to how you may have to resort to definitions and a slower, conscious thought process when confronted with a new situation, but as you learn about the new situation, you cache methods for dealing with it rapidly and unconsciously.

First Application

The above framework is to be built in the context of a particular domain. Sokoban has been experimented with to date, but a more practically interesting domain will likely be next. The tools, once built for a given domain, will be readily adapted to a next domain. For some applications (eg robotic applications), a perceptual front end mapping sensory input to internal simulation model will be necessary.

Sketch of Path to Cognition

The tools will then be used to build an inheritance structure corresponding to human cognition. Thought, language, and understanding are possible because we have a collection of meaningful modules, organized in a hierarchical fashion. One thing that is different about our approach from other hierarchic ontologies, however, is that in our view what is inherited are methods, procedures, that interact with a simulation domain, execute on a simulation domain, compose in a robust way using searches that determine meaningful combinations grounded with respect to the simulation domain.

References

Baum, Eric B. 2004. *What is Thought?* . Cambridge, MA: MIT Press.

Baum, Eric B. 2007. *A Working Hypotheses for Artificial General Intelligence*. In *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, eds. Goertzel, B., and Wang, P., IOS Press, pp 55-74.

Baum, Eric B. 2008a. *Relevance Based Planning, a Worked Example*. <http://www.whatisthought.com/planning.pdf> .

Baum, Eric B. 2008b. *A Model of Language as a Cognitive Tool*. In preparation.

Baum, E. B. and I. Durdanovic. 2000. *Evolution of Cooperative Problem-Solving in an Artificial Economy*. *Neural Computation* 12:2743-2775.

Baum, E. B. and I. Durdanovic. 2002. *An artificial economy of Post production systems*. In *Advances in Learning Classifier Systems*, P.L. Lanzi, W. Stoltzmann and S.M. Wilson (eds.). Berlin: Springer-Verlag, 3-21.

Johnson B. and M-A Williams. 2008. *Comirit: Commonsense Reasoning by Integrating Simulation and Logic*. In *Artificial General Intelligence 2008, Proceedings of the First AGI Conference*. Eds P. Wang, B. Goertzel and S. Franklin. Washington DC: IOS Press.

Hobbs J. R. 2004. *Discourse and Inference*. Preprint. <http://www.isi.edu/~hobbs/disinf-tc.html>

Holland, J. H. 1985. *Escaping brittleness: The possibilities of general purpose machine learning algorithms applied to parallel rule-based systems*. In *Machine Learning II*. Michalski, R.S., Carbonell, J. G., and Mitchell, T. M. (eds.). Morgan Kaufmann (Los Altos). pp593-623.

Kirschner M. and J. Gerhart. 2005. *The Plausibility of Life*. New Haven: Yale University Press.

Lenat, D. B. 1982. *EURISKO: A Program That Learns New Heuristics and Domain Concepts*. (2) *Artificial Intelligence* 19(2) : 189-249.

Schaul, T. 2005. *Evolution of a compact Sokoban solver*. Master Thesis, École Polytechnique Fédérale de Lausanne. posted on <http://whatisthought.com/eric.html>.

Smith, W. D. (2006) *Mathematical Definition of 'Intelligence' (and Consequences)*, preprint, reference 93 on <http://math.temple.edu/~wds/homepage/works.html>

Winograd, T. and F. Flores. 1987. *Understanding Computers and Cognition: A New Foundation for Design*. Boston: Addison-Wesley Professional

CHS-Soar: Introducing Constrained Heuristic Search to the Soar Cognitive Architecture

Sean A. Bittle¹, Mark S. Fox²

Department of Mechanical and Industrial Engineering, University of Toronto,
King's College Road, Toronto, Ontario, Canada, M5S 3G9
sean.bittle@utoronto.ca¹, msf@mie.utoronto.ca²

Abstract

Cognitive architectures aspire for generality both in terms of problem solving and learning across a range of problems, yet to date few examples of domain independent learning has been demonstrated. In contrast, constraint programming often utilizes the same domain independent heuristics to find efficient solutions across a broad range of problems types. This paper provides a progress report on how a specific form of constraint-based reasoning, namely Constrained Heuristic Search (CHS) can be effectively introduced into an integrated symbolic cognitive architecture (Soar) to achieve domain independent learning. The integration of CHS into Soar retains the underlying problem-solving generality of Soar, yet exploits the generalized problem representation and solving techniques associated with constraint programming. Preliminary experiments are conducted on two problems types: Map Colouring and Job Shop Scheduling, both of which are used to demonstrate a domain independent learning using texture based measures.

Introduction

Cognitive architectures specify the underlying infrastructure of tightly coupled mechanisms that support the acquisition and use of knowledge. They aspire to demonstrate problem-solving capabilities ranging from solving highly routine to more difficult problems using large bodies of diverse knowledge [Laird, 2008]. Research on cognitive architectures is important because it supports a central goal of artificial intelligence - namely the creation and understanding of artificial agents that demonstrate similar problem-solving capabilities to humans [Langley et. al., 2006]. While AI research over the past two decades has successfully pursued specialized algorithms for specific problems, cognitive architectures aim for breadth of coverage across a diverse set of tasks and domains [Laird, 2008].

However, to date there appears to be few examples of effective problem-solving or domain independent learning on realistic problems by symbolic cognitive architectures

[Diaper, et. al, 2007]. In contrast, Constraint Programming (CP) has for years established its ability to find efficient solutions to a broad domain of specific real-world problems using domain independent heuristics [Wallace, 1996]. CP, however, has not generally promoted itself as the central problem-solving approach or learning framework for cognitive architectures.

Symbolic cognitive architectures often employ rule-based deductive reasoning to encode knowledge and guide problem solving and learning. Often the division between problem representation and problem solving is often mingled resulting in agents that can only solve a specific problem type and often preclude any form of domain independent learning. In contrast, constraint programming employs a different, yet complementary form of deductive reasoning based on a generalized problem representation which allows it to utilize generic problem solving techniques such as constraint propagation.

Our work seeks to integrate these two different problem-solving paradigms¹ (constraint and rule-based reasoning) into a unified, declarative architecture; namely to introduce Constrained Heuristic Search (CHS) [Fox et. al, 1989] to the Soar cognitive architecture [Laird et. al, 1987]. This paper reports progress to date demonstrating a specific form of domain independent learning. Our broader research effort posits that CHS can provide the requisite framework for unifying the features of constraint programming and symbolic cognitive architectures in order to demonstrate practical problem solving performance and domain independent learning with cognitive architectures.

Soar Cognitive Architecture

One of the earliest symbolic cognitive architectures developed was Soar, created by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University [Laird et. al. 1987]. Soar is a symbolic architecture for general intelligence that integrates basic mechanisms for

¹ In unpublished work by Shivers, the idea of introducing a form of constraint propagation in Soar was suggested [Shivers, 1986].

problem solving, use of knowledge, learning, and perceptual-motor behaviour [Laird et al., 1987].

Soar consists of a single long-term memory encoded as production rules and a single transient declarative working memory. All long-term knowledge in Soar is represented as if-then rules, called productions or production rules. Soar's symbolic working memory holds the agent's assessment of the current situation derived from perception and retrieval of knowledge from its long-term memory [Laird, et. al. 1987]. Working and long-term memories are linked through a decision cycle which selects operators to change states and detects impasses. Problem solving is driven by the act of selecting problem spaces, states, and operators in which each selection is accomplished through a three-phase decision cycle. The phases are repeated until the goal of the current task has been achieved. Chunking is Soar's learning mechanism that converts the results of problem solving in subgoals into rules.

Constraint Programming

Independently, yet in close chronological parallel to the problem-solving work of Newell and Simon, an alternative problem-solving approach evolved called Constraint Programming (CP). Constraint satisfaction is a sub-domain of constraint programming dealing with problems defined over a finite domain and involve no objective function. More formally, a Constraint Satisfaction Problem (CSP) is defined as a set of variables; for each variable, a finite set of possible values (its domain), and a set of constraints restricting the values that the variables can simultaneously assume [Kumar, 1992]. Variables are linked by constraints that can be either logical (i.e. $X \leq Y$) or symbolic (i.e. $X \text{ Loves } Y$).

The principal techniques utilized to solve a CSP include a combination of constraint propagation, variable and value ordering and search. Constraint propagation is the term for propagating the implications of a constraint on one variable onto other variables [Kumar, 1992]. The objective of propagation is to reduce the size of the variable domains based on the constraints imposed on them. Variable and value ordering heuristics can be utilized to dramatically improve the efficiency of search and suggest which variable should be assigned next and in what order the values should be tried [Kumar, 1992]. The role of constraints in problem solving has been reviewed by Fox, where he notes constraints can provide structure to search thereby reducing the size of the search space [Fox, 1986].

Constrained Heuristic Search The idea of augmenting the definition of the problem space through the introduction of a constraint graph representation is encompassed in the Constrained Heuristic Search (CHS) problem-solving model developed by Fox [Fox et. al., 1989]. CHS is a combination of constraint satisfaction and heuristic search. In this model, search is performed in the problem space where each state is defined by a problem

topology and is represented as a constraint graph. CHS augments the definition of a problem space by refining a state to include problem topology (structure), textures (measures of structure) and objective (rating alternative solutions) [Fox, 1989].

Related Work

In a prototype system called CRIPS, Liu demonstrated how constraint and rule-based reasoning can be integrated into a hybrid problem-solving paradigm to deal with the problem of representing disjunctions in rule-based systems [Liu, 1994]. Specifically, the disjunctions are represented as constraint variables and their domains and the relations among disjunctions are represented as constraints. Our work extends this effort by integrating a more general form of constraint based reasoning into an integrated rule-based cognitive architecture.

A hybrid learning system was developed for automating the acquisition of problem-solving knowledge using CSP approaches [Subramanian, Freuder, 1990]. The technique proposed compiles production rules from the observation of constraint-based problem solving behaviour. The compiled rules corresponded to the values deleted by the constraint-based problem solving. In contrast to the work of Subramanian and Freuder, our research proposes to utilize Soar's learning ability to encode texture-based problem independent heuristics as learned rules.

The Adaptive Constraint Engine (ACE) is described as "cognitively oriented" architecture that can learn from experience solving problems in the CSP domain [Epstein, et. al., 2002]. ACE is built upon an underlying general problem solving and learning architecture called FORR (FOr the Right Reasons) which represents knowledge as a three-tiered hierarchy of procedures or "advisors."

Varieties of learning within Soar have been extensively investigated, including across different tasks, and the types it has not yet exhibited include: vary large problems; complex analogies; similarity-based generalization; and, representational shifts [Steier, et. al, 1987]. Our work seeks to reason over a much more abstracted problem space thereby producing more generalized chunks applicable to a much broader range of problem domains.

Design of CHS-Soar

There are two central, yet inter-related design goals of CHS-Soar. First is the use of domain independent problem solving techniques and the second is domain independent learning. Specifically, CHS introduces a standard CSP problem representation which encompasses the use of variables, which have a domain of discrete values, and constraints between variables. CHS also allows us to introduce general purpose problem solving techniques such

as propagation and the use of domain independent problem structure textures to guide variable and value selection. The second design goal is to focus Soar’s internal reasoning on the selection of textures measures to facilitate the learning of domain independent problem solving rules ideally useful across different problem types. Architecturally (Figure 1) CHS-Soar is composed of three parts: [1] the Soar kernel; [2] Soar agent (production rules); and, [3] the external agent. A more detailed overview of the design of CHS-Soar is provided by Bittle and Fox [Bittle, Fox, 2008].

Soar Kernel

The Soar kernel is invariant and encompasses a fixed set of computational mechanisms including: working memory (state representation); long term memory (repository for production rules); decision cycle (used to link working and long term memory); and, learning system.

Soar Agent (Production Rules)

Production rules provide the “programming language” for developers to create different Soar programs, called agents, and form the content of Soar’s Long Term Memory. Soar’s general problem-solving ability results from the fact that the same architecture (the Soar kernel) can be used to solve different problems as defined by different sets of production rules. Rules are used to; propose, select and apply operators.

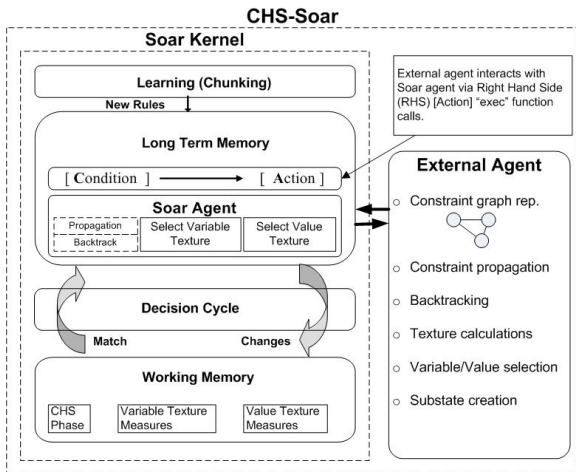


Figure 1: CHS-Soar Architecture

Operators perform actions and are consequently the locus of decision making [Laird et. al, 1987]. CHS-Soar operators are focused on variable and value texture measure selection. CHS-Soar operators provide the framework to introduce the CHS problem solving cycle.

CHS-Soar Problem Solving Cycle Soar’s three-phase decision cycle (proposal, decision, and apply) provides a suitable framework to introduce the CHS problem solving cycle summarized in Table 1.

Table 1: CHS Problem Solving Cycle

Repeat	
1	Conduct propagation (or backtrack) within state <ul style="list-style-type: none"> • calculate variable texture measures
2	Select variable (based on heuristics as suggested by textures measures) <ul style="list-style-type: none"> • calculate value texture measures
3	Select value (based on heuristics as suggested by textures measures)
Until (all variables values instantiated & all constraints satisfied)	

As depicted in Figure 2, three Soar decision cycles are performed during one CHS cycle. Instrumental to the learning of domain independent problem-solving rules is the requirement to have the Soar agent component of the CHS cycle only reason about the selection of normalized variable and value textures measures — not actual problem variable and values. Consequently the Soar agent is decoupled from the actual problem constraint graph. The Soar agent interacts with the external agent via production action-side function calls.

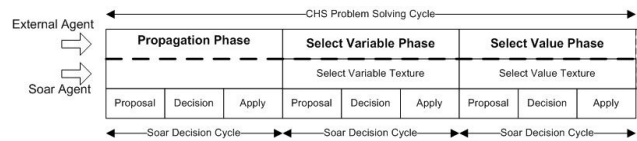


Figure 2: CHS-Soar Problem Solving Cycle

External Agent

The external agent is an optional user-defined program where Soar agent production rule functions reside. From the external agent we can register new functions which can extend the functionality of Soar agent productions. The CHS external agent provides five functions including: CSP problem representation (binary constraint graph); constraint propagation using the AC-3 algorithm [Mackworth, 1977]; chronological backtracking; and, variable and value texture calculations.

Texture Measures Textures are structural measures of the constraint graph used to guide the order of variable and value selection [Fox et. al., 1989]. In order to demonstrate the future introduction of more insightful texture measures [Fox, 1990], three frequently cited [Kumar, 1992] variable and value ordering measures (and their associated heuristics) are utilized as outlined in Table 2.

Table 2: Texture Measures and Associated Heuristics

Name	Texture Measures	Heuristics
Minimum Remaining Value (MRV)	D_i , Number of remaining values in domain.	Select the variable with the smallest D_i value
Degree (DEG)	C_i , number of constraints linked to variable.	Select the variable with the largest C_i value
Least Constraining Value (LCV)	F_i , number of available values in domain of linked variables not instantiated.	Select the value with the largest F_i value

Of particular note is the separation of the texture measure from its associated unary heuristic. In order to facilitate learning across different problem sizes and more importantly different problem domains, texture measures are normalized between 0 (minimum value) and 1 (maximum value). A “pruned” (duplicate values removed) list of normalized texture measures are returned to the Soar agent. Pruning has the effect of further decoupling the texture measures from any specific problem domain.

Soar and External Agent State Representation

The Soar agent initially establishes a binary constraint graph in Soar’s working memory. Once problem solving begins, the Soar agent state representation is defined by the CHS phase (i.e. select variable), state status, desired state, and the collection of normalized texture measures. In contrast, the external agent maintains a complete binary constraint graph problem representation.

Subgoal and Learning

Planning in Soar arises out of its impasse detection and substate creation mechanism. Soar automatically creates a subgoal whenever the preferences are insufficient for the decision procedure to select an operator [Laird, et. al, 1987]. Soar includes a set of default production rules that allow subgoaling to be performed using a simple look-ahead search which CHS-Soar utilizes to evaluate variable and value texture measure performance. After propagation, the Soar agent has no prior knowledge of which variable and/or value texture measure to select (assuming no previous learning). Soar will detect an impasse and automatically subgoal to evaluate each texture measure returned and now cast as operators.

Texture Measure Evaluation CHS-Soar uses three separate, yet related texture measure evaluations. The first is a simple numerical evaluation which gives a higher preference to the texture measure that requires fewer propagation cycles to achieve a candidate solution. Second, if a texture measure leads to a substate domain blow-out it is symbolically evaluated as a failure and rejected. Third, if we cannot match a texture measure in the substate (i.e. an updated constraint graph is generated that does not include the texture measure under evaluation) the texture measure is symbolically evaluated as a “partial failure.”

Since CHS requires both variable and value selections to advance a solution, in order to facilitate the evaluation of any substate variable texture measure we are required to make some type of value texture measure commitment leading to a variable – value texture measure pairing. The approach utilized is to allow Soar to further subgoal (i.e. a sub-subgoal) about value texture selection based on the value texture measures generated for each variable texture measure under consideration. Within each sub-substate,

the variable — value pairing is held constant and a “candidate” solution is attempted.

The second design goal of CHS-Soar is the learning of domain independent problem solving rules — specifically rules that can be transferred and effectively used on different problem types. Chunks are created in the CHS-Soar agent as we resolve impasses. Resulting chunks have as their conditions either unary or binary conditions composed of texture measures (cast as operators).

Implementation of CHS-Soar

The CHS-Soar agent was developed using Soar version 8.6.3 (Windows versions). The external agent was developed in C++ using Windows Visual Studio 2005.

Experiments

The paper reports on two selected computational experiments performed to-date. The objective of experiment 1 is to establish the subgoaling, learning and transfer of learning ability of CHS-Soar. Experiment 2 considered the impact of problem size/complexity on the performance of externally learned chunks. Problem instances were run 10 times and averaged. Experiments were conducted for two sample problems: [1] map colouring (MC); [2] job-shop scheduling (JSS), both well known combinatorial problems. The relationship of the map colouring problem and the more formal four-colour problem to general intelligence is highlighted by Swart [Swart, 1980] who notes one of the main attractions of the problem lies in the fact that it is “so simple to state that a child can understand it.” The JSS problem instance is taken from [Pang, et. al, 2000]. Results are presented in terms of “Decisions (cycles)” — the basic unit of reasoning effort in Soar (see Figure 2).

Experiment 1: Learning and Transfer of Learning

The first experiment investigated the impact of subgoaling, internal learning and the transfer of externally learned rules between problem types. In order to assess the performance of an external set of learned chunks on the map colouring problem, the chunks acquired through internal learning from the JSS problem were introduced for test case 4 (see Table 3) and vice versa for the JSS.

Table 3: Learning and Transfer of Learning Test Cases

Ref	Test Case	Description
1	Random	Random selection of variable and value texture measures (indifferent).
2	Hard-Coded	Explicitly coded MRV, DEG variable selection heuristics and LCV value selection heuristic.
3	Internal	Problem is run using internally acquired chunks.
4	External	Problem is run using externally acquired chunks from different problem domain.

Results and Discussion

Figure 3 presents the number of decisions required to solve the CHS-Soar model of a map colouring problem (11 variables and 23 constraints) for the test cases outlined in Table 3. As highlighted the hard-coded variable and value texture selection heuristics (case 2) result in a lower number of decisions (41% reduction) then when using a

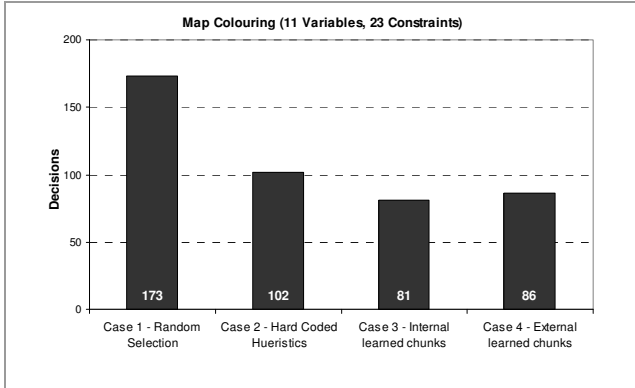


Figure 3: Map Colouring-Decisions as a Function of Subgoaling/Learning Test Cases.

random selection of texture measures (case 1). When subgoaling and learning were enabled (case not shown), CHS-Soar generated on average 60 chunks. Case 3 demonstrates improved problem solving using the 60 internally acquired learned rules as compared to the explicit use of hard-coded heuristics (case 2) resulting in a 21% reduction in decision cycles. For case 3 it was observed that CHS-Soar used both a combination and range of variable and value texture measure types and values to secure a solution.

Case 4 demonstrates the domain independent problem solving performance of CHS-Soar chunks using 177 chunks externally learned from the JJS problem. Case 4 resulted in a 16% reduction in decision cycles over case 2 (hard-coded heuristics). Yet for case 4 we observe a 6% increase in decision cycles as compared case 3 (internal learned chunks).

Figure 4 presents the number of decision cycles required to solve the CHS-Soar model of a 3x5 Job-Shop Schedule (JSS) problem (15 variables and 35 constraints) for the test cases outlined in Table 3. Similar to the map colouring problem instance, we can observe a reduction in decision cycles (29% lower) using hard coded heuristics (case 2) over the random selection (case 1). Subgoaling and learning (one training run) resulted in 177 internally learned chunks. We note a 23% reduction in decision cycles when the problem is solved using internally learned rules (case 3) as compared with the explicit use of hard coded heuristics (case 2). Case 4 illustrates the impact of using 60 externally learned rules acquired from the map colouring problem. Case 4 shows a 10% reduction in the number of decision cycles as compared to case 2 (hard coded).

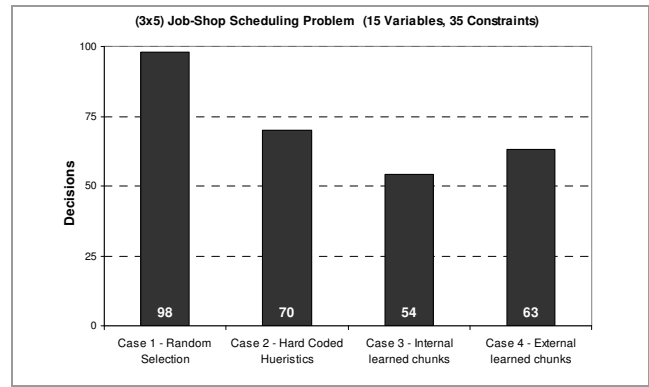


Figure 4: JSS-Decisions as a Function of Subgoaling/Learning Test Cases.

Experiment 2: Scalability

The second experiment explored the performance of externally learned chunks as a function of problem complexity for the map colouring problem. Specifically, the 177 learned chunks from the job-shop scheduling problem (3x5) were introduced into progressively larger map colouring problems ranging in size from 7 variables (9 constraints) to 44 variables (103 constraints).

Results and Discussion

Figure 5 presents a comparison of decisions cycles for 3 selected test cases (see Table 3 as a function of problem complexity) for the map colouring problem. For case 5, the 177 externally learned chunks from the 3x5 JJS problem were introduced to solve each MC problem instance.

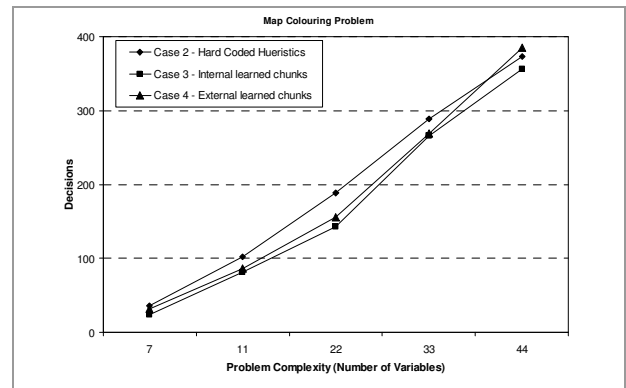


Figure 5: Map Colouring-Comparison of Decisions versus Problem Complexity for Selected Test Cases.

As illustrated for this specific problem series, the externally acquired learned rules demonstrate similar problem solving performance, to both the hard-coded heuristics and internally generated chunks as problem complexity increases.

Conclusions and Future Work

While preliminary, work to date has demonstrated how a specific form of constraint-based reasoning (CHS) can be effectively introduced into a symbolic cognitive architecture (Soar) using a generalized set of 34 productions. This results in an integration of two important types of reasoning techniques, namely constraint propagation and rule chaining.

We have demonstrated the ability of CHS-Soar to learn rules while solving one problem type (i.e., graph colouring) that can be successfully applied in solving another problem type (i.e., Job Shop Scheduling). CHS and specifically the use of texture measures allow us to transform a problem specific search space (based on variables and values) into a more generalized one based on abstracted texture measures which provides the environment to achieve domain independent learning.

Future work will include an expanded portfolio of test case problems to further validate and better understand CHS-Soar ability to learn and use domain independent texture based rules for more practical problems. More insightful texture measures, support augmentation (i.e. frequency) and improved texture evaluation functions are also being investigated as well as a texture based “discovery system.”

References

- Bittle, S.A., Fox, M.S., (2008), "Introducing Constrained Heuristic Search to the Soar Cognitive Architecture", Technical Report, Enterprise Integration Laboratory <http://www.eil.utoronto.ca/other/papers/index.html>.
- Diaper, D., Huyck, C., Amavasai, B., Cheng, X. and Oussalah, M. 2007. Intelligently Engineering Artificial Intelligence Engineering: The Cognitive Architectures Competition. Proceedings of the workshop W3 on Evaluating Architectures for Intelligence, at the Association for the Advancement of Artificial Intelligence annual conference (Vancouver).
- Epstein, S. L., Freuder, E.C., Wallace, R.J, Morozov, A., Samuels, B. 2002. The Adaptive Constraint Engine. In P. Van Hentenryck, editor, Principles and Practice of Constraint Programming -- CP 2002: 8th International Conference, Proceedings, volume LNCS 2470 of Lecture Notes in Computer Science, pages 525--540. SpringerVerlag, 2002.
- Fox, M.S., Sadeh, N., Bayken, C., 1989. Constrained Heuristic Search. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pg:309– 315.
- Fox, M.S., 1986. Observations on the Role of Constraints in Problem Solving, Proceedings Sixth Canadian Conference on Artificial Intelligence, Montreal, Quebec.
- Kumar, V., 1992. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32-44, 1992
- Laird, J.E., Newell, A., Rosenbloom, P. 1987. Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 33: 1-64.
- Laird, J. E. 2008. Extending the Soar Cognitive Architecture. *Artificial General Intelligence Conference*, Memphis, TN.
- Langley, P., Laird, J., Rogers, S., 2006. Cognitive Architectures: Research Issues and Challenges, Technical Report, Computational Learning Laboratory, CSLI, Stanford University, CA..
- Liu, B., 1994. Integrating Rules and Constraints Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence (TAI-94), November 6-9, 1994, New Orleans, United States, 1994.
- Mackworth, A.K., 1977. Consistency in Networks of Relations, *J. Artificial Intelligence*, vol. 8, no. 1, pp. 99-118, 1977.
- Pang, W., Goodwin, S.D., 2004. Application of CSP Algorithms to Job Shop Scheduling Problems, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.0.9563>.
- Shivers, O., 1986. Constraint propagation and macro-compilation in Soar. In Proceedings of the Soar Fall '86 Workshop, November 22, 1986.
- Steier, D.M., Newell, A., Flynn, R., Polk, T.A., Unruh, A., 1987. "Varieties of Learning in Soar." *The Soar Papers: Research on Integrated Intelligence*, Volume 1, Chapter 18, Pages 536-548, MIT Press, Cambridge, Massachusetts.
- Subramanian, S.; Freuder, E.C. 1990. Rule compilation from constraint-based problem solving. *Tools for Artificial Intelligence*, 1990, Proceedings of the 2nd International IEEE Conference on, Vol., Iss, 6-9 Nov 1990, pp: 38-47.
- Swart, E.R., 1980. "The philosophical implications of the four-color problem". *American Mathematical Monthly* (JSTOR) 87 (9): 697--702. http://www.joma.org/images/upload_library/22/Ford/Swart697-707.pdf.
- Wallace, M., 1996. Practical applications of constraint programming, *Constraints*, An International Journal, 1, 139-168.

In Search of Computational Correlates of Artificial Qualia

Antonio Chella, Salvatore Gaglio

Dipartimento di Ingegneria Informatica Università di Palermo

Viale delle Scienze, building 6, 90128 Palermo, Italy

{chella, gaglio}@unipa.it

Abstract

In previous papers we presented a robot cognitive architecture organized in three computational *areas*. The *subconceptual* area is concerned with the processing of data coming from the sensors. In the *linguistic* area representation and processing are based on a logic-oriented formalism. The *conceptual* area is intermediate between the subconceptual and the linguistic areas and it is based on the notion of *conceptual spaces*. The robot, starting from the 3D information stored in the conceptual area and from the data coming from sensors and processed by the subconceptual area, is able to build a 2D viewer dependent reconstruction of the scene it is perceiving. This 2D model corresponds to what the robot is seeing at any given time. We suggest that the conceptual and the linguistic areas are at the basis of the robot artificial qualia.

Introduction

It has been questioned if robots may have qualia, i.e., qualitative, phenomenal experiences in the sense discussed, among others, by Chalmers (1996).

We are not interested in the problem of establishing whether robots can have real phenomenal experiences or not. For our present concerns, speak of robot's "artificial qualia" in a sense similar to Aleksander (1996). We use this expression in a somewhat metaphorical sense: we call "artificial quale" a state that in some sense corresponds to the "phenomenal" experience of the robot, without making any hypothesis concerning the fact that the robot truly experiences it.

In previous papers (Chella et al. 1997, 2000) we presented a robot cognitive architecture organized in three computational *areas* - a term which is reminiscent of the cortical areas in the brain.

The *subconceptual* area is concerned with the processing of data coming from the sensors. Here information is not yet organized in terms of conceptual structures and categories. From the point of view of the artificial vision, this area includes all the processes that extract the 3D model of the perceived scene. In the *linguistic* area representation and processing are based on a logic-oriented formalism. We adopt the term "linguistic" instead of the overloaded term "symbolic", because we want to stress the

reference to formal languages in the knowledge representation tradition.

The *conceptual* area is intermediate between the subconceptual and the linguistic areas. Here, data is organized in conceptual "gestaltic" structures, that are still independent of any linguistic description. The symbolic formalism of the linguistic area is interpreted on aggregation of these structures.

We suggest that the conceptual and the linguistic areas are at the basis of the robot artificial qualia. In our model, the robot, starting from the 3D information stored in the conceptual area and from the data coming from sensors and processed by the subconceptual area, is able to build a 2D, viewer dependent reconstruction of the scene it is perceiving. This 2D model corresponds to what the robot is seeing at any given time. Its construction is an active process, driven by both the external flow of information and the inner model of the world.

The Cognitive Architecture

The proposed architecture (Fig. 1) is organized in computational "areas". In our model, the areas are concurrent computational components working together on different commitments. There is no privileged direction in the flow of information among them: some computations are strictly bottom-up, with data flowing from the subconceptual up to the linguistic through the conceptual area; other computations combine top-down with bottom-up processing.

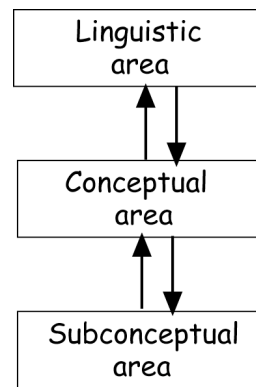


Figure 1: The cognitive architecture

Conceptual Spaces

The conceptual area, as previously stated, is the area between the subconceptual and the linguistic area. This area is based on the theory of conceptual spaces (Gärdenfors 2000).

Conceptual spaces provide a principled way for relating high level, linguistic formalisms with low level, unstructured representation of data. A conceptual space CS is a metric space whose dimensions are the quantities generated as outputs of computational processes occurring in the subconceptual area, e.g., the outputs of the neural networks in the subconceptual area. Different cognitive tasks can presuppose different conceptual spaces, and different conceptual spaces can be characterized by different dimensions. Examples of possible dimensions, with reference to object perception tasks, are: color components, shape parameters, spatial coordinates, motion parameters, and so on. In general, dimensions are strictly related to the results of measurements obtained by sensors. In any case, dimensions do not depend on any specific linguistic description. In this sense, conceptual spaces come before any symbolic or propositional characterization of cognitive phenomena.

We use the term *knoxel* to denote a point in a conceptual space. The term *knoxel* (in analogy with the term *pixel*) stresses the fact that a point in CS is the knowledge primitive element at the considered level of analysis.

The conceptual space CS acts as a workspace in which low-level and high-level processes access and exchange information respectively from bottom to top and from top to bottom. However, the conceptual space is a workspace with a precise geometric structure of metric space and also the operations in CS are geometrics: this structure allow us to describe the functionalities of the robot awareness in terms of the language of geometry.

It has been debated if visual perception is based on a 3D representation, as presupposed by Marr (Marr 1982). In the present architecture, we maintain the Marrian approach, according to which our *knoxel* corresponds to a moving 3D shape.

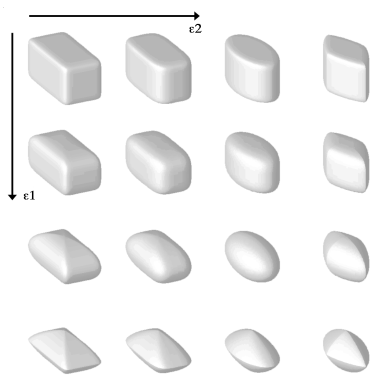


Figure 2: Superquadric shapes obtained by changing the form factors.

Object and Scene Representation

In (Chella et al. 1997) we assumed that, in the case of static scenes, a *knoxel* k coincides with a 3D primitive shape, characterized according to Constructive Solid Geometry (CSG) schema. In particular, we adopted *superquadrics* (Jaklič et al. 2000) as the primitive of CSG. Superquadrics allow us to deal with a compact description of the objects in the perceived scene. This approach is an acceptable compromise between the compression of information in the scene and the necessary computational costs. Moreover, superquadrics provide good expressive power and representational adequacy.

Superquadrics are geometric shapes derived from the quadric parametric equation with the trigonometric functions raised to two real exponents. Fig. 2 shows the shape of a superquadric obtained by changing its form factors.

In order to represent composite objects that cannot be reduced to single *knoxels*, we assume that they correspond to groups of *knoxels* in CS. For example, a chair can be naturally described as the set of its constituents, i.e., its legs, its seat and so on.

Fig. 3 (left) shows a hammer composed by two superquadrics, corresponding to its handle and to its head. Fig. 3 (right) shows a picture of how hammers are represented in CS. The concept hammer consists of a set of pairs, each of them is made up of the two components of a specific hammer, i.e., its handle and its head.

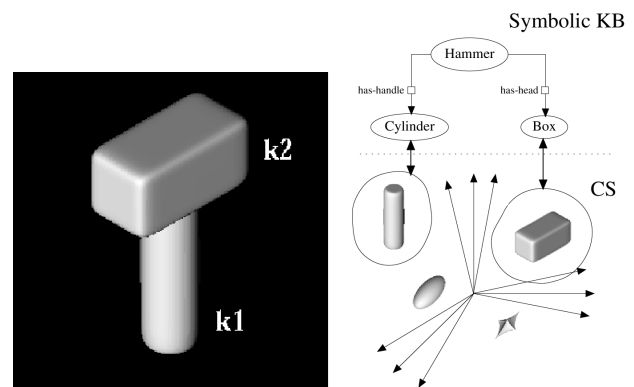


Figure 3: A hammer made up by two superquadrics and its representation in the conceptual space.

Dynamic scenes

In order to account for the perception of dynamic scenes, we choose to adopt an intrinsically dynamic conceptual space. It has been hypothesized that simple motions are categorized in their wholeness, and not as sequences of static frames. In other words, we assume that simple motions of geometrically primitive shapes are our perceptual primitives for motion perception.

In our dynamic conceptual space, a knoxel now corresponds to a “generalized” simple motion of a superquadric. By “generalized” we mean that the motion can be decomposed in a set of components each of them associated with a degree of freedom of the moving superquadric.

A way of doing this, is suggested by the well known Discrete Fourier Transform (DFT). Given a parameter of the superquadric, e.g., \mathbf{a}_x , consider the function of time $\mathbf{a}_x(t)$; this function can be seen as the superimposition of a discrete number of trigonometric functions. This allows the representation of $\mathbf{a}_x(t)$ in a discrete functional space, whose basis functions are trigonometric functions.

By a suitable composition of the time functions of all superquadric parameters, the overall function of time describing superquadrics parameters may be represented in its turn in a discrete functional space. We adopt the resulting functional space as our dynamic conceptual space. This new CS can be taught as an “explosion” of the space in which each main axis is split in a number of new axes, each one corresponding to a harmonic component. In this way, a point \mathbf{k} in the CS now represents a superquadric along with its own simple motion. This new CS is also consistent with the static space: a quiet superquadric will have its harmonic components equal to zero.

In Fig. 4 (left) a static CS is schematically depicted; Fig. 4 (right) shows the dynamic CS obtained from it. In the CS on the left, axes represent superquadric parameters; in the rightmost figure each of them is split in the group of axes, that represent the harmonics of the corresponding superquadric parameter.

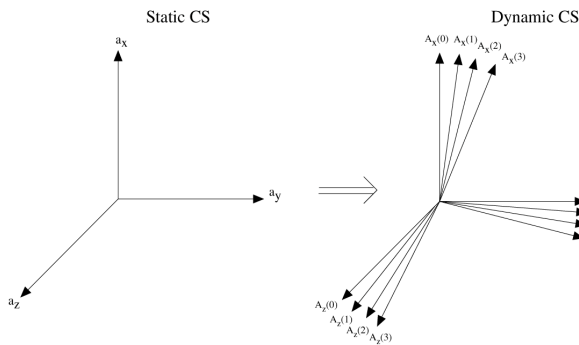


Figure 4: An evocative, pictorial representation of the static and dynamic conceptual spaces.

Situations and Actions

Let us consider a scene made up by the robot itself along with other entities, like objects and persons. Entities may be approximated by one or more superquadrics. Consider the robot moving near an object. We call *situation* this kind of scene. It may be represented in CS by the set of the

knoxels corresponding to the simple motions of its components, as in Fig. 5 (left) where \mathbf{k}_a corresponds to an obstacle object, and \mathbf{k}_b corresponds to the moving robot.

A situation is therefore a configuration of knoxels that describe a state of affairs perceived by the robot. We can also generalize this concept, by considering that a configuration in CS may also correspond to a scene imagined or remembered by the robot.

For example, a suitable imagined situation may correspond to a goal, or to some dangerous state of affairs, that the robot must figure out in order to avoid it. We added a binary valuation that distinguish if the knoxel is effectively perceived, or it is imagined by the robot. In this way, the robot represents both its perceptions and its imaginations in conceptual space.

In a perceived or imagined situation, the motions in the scene occur simultaneously, i.e., they correspond to a single configuration of knoxels in the conceptual space.

To consider a composition of several motions arranged according to a temporal sequence, we introduce the notion of *action*: an action corresponds to a “scattering” from one situation to another situation in the conceptual space, as in Fig. 5 (right).

We assume that the situations within an action are separated by instantaneous events. In the transition between two subsequent configurations, a “scattering” of some knoxels occur. This corresponds to a discontinuity in time that is associated to an instantaneous event.

The robot may perceive an action passively when it sees some changes in the scene, e.g., a person in the robot environment changing her position. More important, the robot may be the actor of the action itself, when it moves or when it interacts with the environment, e.g., when it pushes an object. In both cases, an action corresponds to a transition from a situation to another.

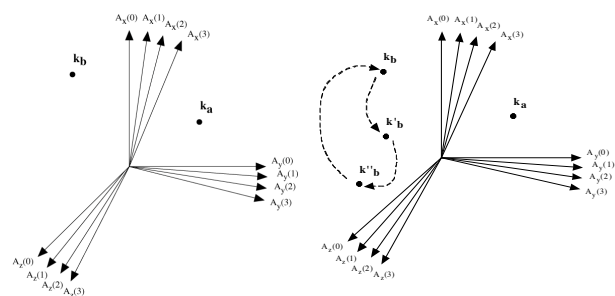


Figure 5: An example of situation and action in CS.

Linguistic Area

The representation of situations and actions in the linguistic area is based on a high level, logic oriented formalism. The linguistic area acts as a sort of “long term memory”, in the sense that it is a semantic network of symbols and their relationships related with the robot

perceptions and actions. The linguistic area also performs inferences of symbolic nature.

In the current implementation, the linguistic area is based on a hybrid KB in the KL-ONE tradition (Brachman and Schmoltze 1985). A hybrid formalism in this sense is constituted by two different components: a terminological component for the description of concepts, and an assertional component, that stores information concerning a specific context.

In the domain of robot actions, the terminological component contains the description of relevant concepts such as *Situation*, *Action*, *Time_instant*, and so on.

In general, we assume that the description of the concepts in the symbolic KB is not completely exhaustive. We symbolically represent only that information that is necessary for inferences.

The assertional component contains facts expressed as assertions in a predicative language, in which the concepts of the terminological components correspond to one argument predicates, and the roles (e.g. *precond*, *part_of*) correspond to two argument relations.

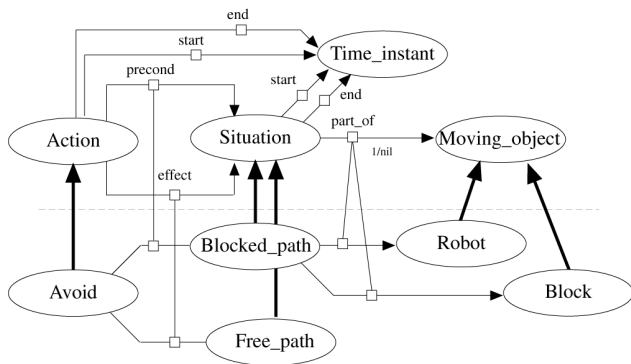


Figure 6. A fragment of the adopted KB.

Fig. 6 shows a fragment of the terminological knowledge base; in order to increase the readability, we adopted a graphical notation of the kind used by (Brachman and Schmoltze 1985). In the upper part of the figure some highly general concept is represented. In the lower part, the *Avoid* concept is shown, as an example of the description of an action in the terminological KB.

Every *Situation* has a starting and an ending instant. So, the concept *Situation* is related to *Time_instant* by the roles *start* and *end*. A *Robot* is an example of a moving object. Also actions have a start instant and an end instant. An *Action* involves a temporal evolution (a scattering in CS). *Actions* have at least two parts, that are *Situations* not occurring simultaneously: the *precond* (i.e., the precondition) and the *effect* of the action itself.

An example of *Action* is *Avoid*. According to the KB reported in the figure, the precondition of *Avoid* is a *Blocked_path* situation, to which participate the robot and

a blocking object. The effect of the *Avoid* action is a *Free_path* situation.

In general, we assume that the description of the concepts in the symbolic KB (e.g. *Blocked_path*) is not completely exhaustive. We symbolically represent only that information that is necessary for inferences.

The assertional component contains facts expressed as assertions in a predicative language, in which the concepts of the terminological components correspond to one argument predicates, and the roles (e.g. *precond*, *part_of*) correspond to two argument relations. For example, the following predicates describe that the instance of the action *Avoid* has as a precondition the instance of the situation *Blocked_path* and it has as an effect the situation *Free_path*:

```
Avoid(av1)
precond(av1, bl1)
effect(av1, fr1)
Blocked_path(bl1)
Free_path(fr1)
```

The linguistic area is the area where the robot interacts with the user: the user may performs queries by using the symbolic language in order to orient the actions, for example, the user may ask the robot to search for an object. The user may performs queries by using the symbolic language in order to orient the actions, for example, the user may ask the robot to search for an object.

Moreover, the system may generate assertions describing the robot current state, its perceptions, its planned actions, and so on. However, the terms in our linguistic area are strictly “anchored” to knoxels in the conceptual area, in the sense that the meaning of the terms in the linguistic area is represented by means of the corresponding knoxels in the conceptual area. Therefore, in our architecture, symbolic terms are strictly related with the robot visual perceptions. The role of language is to “summarize” the dynamics of the knoxels at the conceptual area.

Artificial Qualia

It has been questioned if robot may have “qualia”, i.e., qualitative, phenomenal experience. In our opinion it should be more correct to talk about the robot “artificial qualia” in the sense of (Aleksander 1996), so the problem is: during its mission tasks, has the robot some phenomenal experience?

In the proposed architecture, we have shown that the basis of the robot perception is the conceptual area, where the perceived scene is represented in terms of knoxels that describe the shape and the motion of the perceived entities, and the linguistic area where the scene is represented in terms of linguistic entities that summarize the dynamics of the knoxels in the conceptual space.

Now, we introduce an *iconic* area where a 2D reconstruction of the scene is built as a geometric projection of the knoxels in its conceptual space (where the

information about the scene is maintained in 3D) and from the data coming from sensors and processed by the subconceptual area.

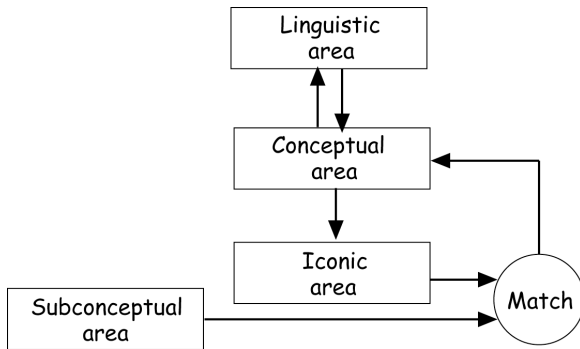


Figure 7. The revised architecture with the 2D iconic area and the perception loop.

Fig. 7 shows the robot architecture revisited to take into account the iconic area: in the revised architecture there is a perception loop between the conceptual area where the knoxels are represented, the iconic area and the subconceptual area. This perception loop has the role to adjust the match between the 2D iconic representation of the scene obtained from the knoxels in the conceptual area, and the external flow of perception data coming out from the subconceptual area.

We present the operation of the revised architecture with reference to the *CiceRobot* robotic project, an operating autonomous robot performing guided tours at the Archaeological Museum of Agrigento (Chella & Macaluso 2008).

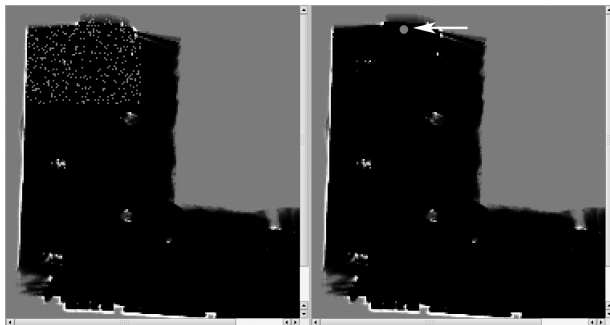


Figure 8. The initial distribution of expected robot positions (left), and the cluster of winning expected positions, highlighted by the arrow.

In order to compare the CS content with the external environment by the iconic area, the robot is equipped with a stochastic match algorithm based on particle filter (see, e.g., Thrun et al. 2005; details are reported in Chella &

Macaluso 2008). In brief, the algorithm generates a cloud of hypothesized possible positions of the robot (Fig. 8). For each position, the corresponding expected image scene is generated in the iconic area by means of geometric projection operations of the corresponding knoxels in CS. The generated image is then compared with the acquired image (Fig. 9).



Figure 9. The 2D image from the robot video camera (left) and the corresponding 2D image generated in the iconic area by the knoxels of CS (right).

An error measure ϵ of the match is computed between the expected and the effective image scene. The error ϵ weights the expected position under consideration by considering the distribution of the vertical edges in the generated and the acquired images (mathematical details in Chella & Macaluso 2008). In subsequent steps, only the winning expected positions that received the higher weight are taken, while the other ones are dropped.

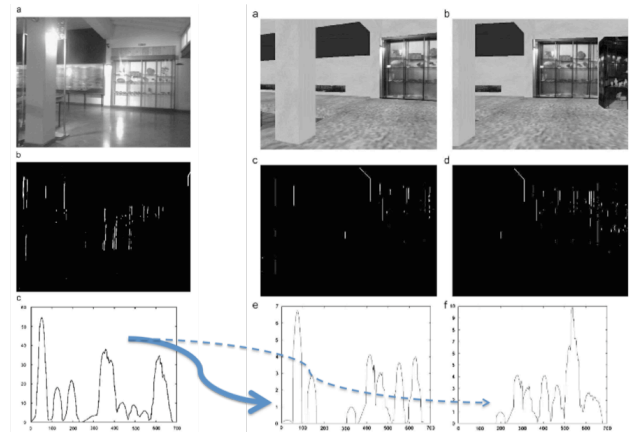


Figure 10. The image acquired by the robot camera along with the vertical edges and their distribution (left). The simulated image from CS corresponding to the hypothesis with the highest weight. (center) A simulated image corresponding to an hypothesis with a lesser weight (right).

When the image generated in the iconic area matches with the image acquired from the robot camera, the knoxels in CS corresponding to the winning image are highlighted: they give rise to the description of the perceived scene by

means of the knowledge stored in the CS and the linguistic area, as described in previous Sects. As an example, in the situation described in Fig. 9, the assertional component of the KB generates the predicates stating that the robot is in a free path, and there is an armchair in front and on the right, and there is a column on the left:

```
Free_path(p1)
Armchair(a1)
Armchair(a2)
Column(c1)
Right_of(robot, a2)
Front_of(robot, a1)
Left_of(robot, c1)
```

We propose that the described reconstruction and match process constitutes the phenomenal experience of the robot, i.e., what the robot sees at a given instant. This kind of seeing is an active process, since it is a reconstruction of the inner percept in ego coordinates, but it is also driven by the external flow of information. It is the place in which a global consistency is checked between the internal model and the visual data coming from the sensors (Gaglio et al. 1984).

The synthesized pictures of the world so generated projects back in the external space the geometrical information contained in the knoxels in the conceptual space and, matched to incoming sensor data, it accounts for the understanding of the perceptive conscious experience. There is no need for a homunculus that observes it, since it is the ending result of an active reconstruction process, which is altogether conscious to the robot, which sees according to its own geometric (not yet linguistic) interpretation.

The phenomenal experience is therefore the stage in which the two flows of information, the internal and the external, compete for a consistent match by the particle filter algorithm. There a strong analogy with the phenomenology in human perception: when one perceives the objects of a scene he actually experiences only the surfaces that are in front of him, but at the same time he builds a geometric interpretation of the objects in their whole shape. In “gestaltian” terms, the robot in the described example perceives the whole armchairs and columns and not their visible sides only.

Conclusions

According to the “quantum reality hypothesis” proposed by (Goertzel 2006), the described conceptual space has some similarities with the Goertzel *internal virtual multiverse*, in the sense that the CS is able to generate possible worlds and possible sequences and branches of events. Also the described match operation between the image acquired by the camera and the 2D reconstructed image from the iconic area may be seen as a sort of “collapse” of the several possible situations and actions in CS to a single perceived situation. Related ideas have been

proposed by Edelman (1989), Humphrey (1992) and Grush (2004).

The described model of robot perceptual phenomenology highlights open problems from the point of view of the computational requirements. The described architecture requires that the 3D reconstruction of the dynamic scenes and the match with the scene perceived by the robot during its tasks should be computed in real time. At the current state of the art in computer vision and computer graphics literature, this requirement may be satisfied only in case of simplified scenes with a few objects where all the motions are slow.

However, we maintain that our proposed architecture is a good starting point to investigate robot phenomenology. As described in the paper it should be remarked that a robot equipped with artificial phenomenology performs complex tasks better and more precisely than an “unconscious” reactive robot.

References

- Aleksander, I. 1996. *Impossible Minds: My Neurons, My Consciousness*. London: Imperial College Press.
- Brachman, R.J. and Schmoltze, J.C. 1985. An overview of the KL-ONE knowledge representation system, *Cognitive Science*, 9, pp. 171-216.
- Chalmers, D. J. 1996. *The Conscious Mind: In Search of a Fundamental Theory*. Oxford: Oxford University Press.
- Chella, A. and Macaluso, I. 2008. The Perception Loop in CiceRobot, a Museum Guide Robot, *Neurocomputing*, doi:10.1016/j.neucom.2008.07.011.
- Chella, A., Frixione, M. and Gaglio, S. 1997. A cognitive architecture for artificial vision, *Artificial Intelligence*, 89, pp. 73-111.
- Chella, A., Frixione, M. and Gaglio, S. 2000. Understanding dynamic scenes, *Artificial Intelligence*, 123, pp. 89-132.
- Edelman, G. 1989. *The Remembered Present: A Biological Theory of Consciousness*. New York: Basic Books.
- Gaglio, S., Spinelli, G. and Tagliascio, V. 1984. Visual Perception: an Outline of a Generative Theory of Information Flow Organization, *Theoretical Linguistics*, 11, Vol. 1-2, pp. 21-43.
- Goertzel, B. 2006. *The Hidden Pattern – A Patternist Philosophy of Mind*. Boca Raton: BrownWalker Press.
- Grush, R. 2004. The Emulator Theory of Representation: Motor Control, Imagery and Perception, *Behavioral Brain Sciences*, 27, pp. 377-442.
- Gärdenfors, P. 2000. *Conceptual Spaces*. Cambridge, MA: MIT Press.
- Humphrey, N. 1992. *A History of Mind*. New York: Springer-Verlag.
- Jaklič, A., Leonardis, A., Solina, F. 2000. *Segmentation and Recovery of Superquadrics*. Boston, MA: Kluwer Academic Publishers.
- Marr, D. 1982. *Vision*. New York: W.H. Freeman.
- Thrun, S., Burgard, W., Fox, D. *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

Combining Analytical and Evolutionary Inductive Programming*

Neil Crossley and Emanuel Kitzelmann and Martin Hofmann and Ute Schmid

Faculty Information Systems and Applied Computer Science, University of Bamberg, Germany
neil.crossley@stud.uni-bamberg.de, {emanuel.kitzelmann, martin.hofmann, ute.schmid}@uni-bamberg.de

Abstract

Analytical inductive programming and evolutionary inductive programming are two opposing strategies for learning recursive programs from incomplete specifications such as input/output examples. Analytical inductive programming is data-driven, namely, the minimal recursive generalization over the positive input/output examples is generated by recurrence detection. Evolutionary inductive programming, on the other hand, is based on searching through hypothesis space for a (recursive) program which performs sufficiently well on the given input/output examples with respect to some measure of fitness. While analytical approaches are fast and guarantee some characteristics of the induced program by construction (such as minimality and termination) the class of inducible programs is restricted to problems which can be specified by few positive examples. The scope of programs which can be generated by evolutionary approaches is, in principle, unrestricted, but generation times are typically high and there is no guarantee that such a program is found for which the fitness is optimal. We present a first study exploring possible benefits from combining analytical and evolutionary inductive programming. We use the analytical system IGOR2 to generate skeleton programs which are used as initial hypotheses for the evolutionary system ADATE. We can show that providing such constraints can reduce the induction time of ADATE.

Introduction

Automated programming research addresses the old dream of AI having computer systems which can automatically generate computer programs (Green *et al.* 1983; Biermann, Guiho, & Kodratoff 1984). Such systems would mimic the cognitive ability and expertise of human programmers. Deductive approaches to automated programming might reflect the use of general and specific knowledge about a programming language and the domain of the given problem which is available to experienced programmers. But neither proof-based approaches (Manna & Waldinger 1975; 1992) nor transformational approaches (Burstall & Darlington 1977) seem to be plausible cognitive strategies.

*Research was supported by the German Research Community (DFG), grant SCHM 1239/6-1.

Furthermore, as programming assistants such systems can only be used by highly trained experts, since programs must be completely and correctly specified in some formal language. Inductive approaches, on the other hand, might reflect strategies used by human programmers with limited experience in recursive programming. Common to all approaches to inductive programming is that recursive programs are constructed from *incomplete* specifications, typically samples of the desired input/output behavior and possibly additional constraints such as length or time efficiency. Such kind of information can be much more easily provided by programmers without special training and therefore, inductive programming approaches are good candidates for the development of programming assistants. There are two distinct approaches to inductive programming: analytical and evolutionary inductive programming.

Analytical inductive programming is data-driven and often relies on specifications which consist only of a small set of positive input/output examples. A recursive program is learned by detecting recurrence relations in the input/output examples and generalization over these regularities (Summers 1977; Kitzelmann & Schmid 2006). Typically, analytical approaches are fast and they can guarantee certain characteristics for the constructed program such as minimality of the generalization with respect to the given examples and termination. However, the class of learnable programs is necessarily restricted to such problems which can be specified by small sets of input/output examples. The scope of learnable programs can be somewhat widened by allowing the use of background knowledge (Kitzelmann 2008). Analytical inductive programming mimics a strategy often used by human programmers with limited experience in coding recursive programs: Explicitly write down the behavior of the desired program for the first possible inputs, observe the regularities between succeeding examples which reflect how the problem of size n can be solved using the solution of the problem with size $n-1$ and use this information to construct the recursive solution (Kahney 1989; Kruse 1982; Pirolli & Anderson 1985).

Evolutionary inductive programming is based on search through the hypothesis space of possible pro-

grams given some (syntactically restricted) programming language. A hypothesis is returned as a solution if it performs sufficiently well on the input/output examples with respect to some measure of fitness, typically involving code length and time efficiency. The scope of programs learnable with an evolutionary approach is, in principle, unrestricted. But, generation times are typically high and there is no guarantee that the returned program is the optimal solution with respect to the fitness function. Evolutionary inductive programming follows a generate-and-test strategy which – to some extent – might be used by inexperienced programmers when they do have a clear idea about the desired program behavior but no clear idea about the algorithm. A cognitively more plausible search strategy is hill climbing, that is, searching for a solution by stepwise transforming the current solution such that it becomes more similar to the desired goal by covering more of the positive input/output examples and having a more desirable fitness. This idea is also incorporated in the means-end strategy (Newell & Simon 1961) and was shown as a strategy often exhibited in human problem solving (Greeno 1974). That is, to make evolutionary programming a more plausible strategy and at the same time to make it more efficient, it would be helpful to provide a program skeleton as initial seed which is afterwards stepwise refined with respect to coverage of examples and fitness.

Therefore, we propose to use analytical inductive programming to generate initial seeds for evolutionary programming. The combination of both approaches should be such that if a solution can be generated by analytical means alone, this fast and reliable approach should be used exclusively. If the problem is out of scope for analytical programming, at least a partial solution could be provided which then can be used as input for program evolution. In the following, we first describe the evolutionary programming system ADATE and the analytical programming system IGOR2. Afterwards we will introduce different strategies for the analytical generation of program seeds with IGOR2 and their incorporation in ADATE. We will report results of our first experiments and give a short conclusion.

Evolutionary Programming with ADATE

ADATE (Olsson 1995; Vattekar 2006) was initially proposed in the nineties and has been continually extended ever since. To our knowledge, it is the most powerful approach to inductive programming which is currently available. ADATE constructs programs in a subset of the functional language ML, called ADATE-ML. The problem specification presented to ADATE consists of: a set of data types and a set of primitive functions; a set of sample inputs; an evaluation function; an initial declaration of the goal function f . Sample inputs typically are input/output pairs. It is enough to give only positive examples, but it is additionally possible to provide negative examples. There are a number of predefined evaluation functions, each using different mea-

asures for syntactic complexity and time efficiency of the goal program. These are completed by a callback evaluation function given in the problem specification which evaluates the return value of a inferred function for a given input example. In general, the search heuristic is to prefer smaller and faster functions. As typical for evolutionary approaches, there are sets of individuals which are developed over generations such that fitter individuals have more chances to reproduce. If no additional knowledge is provided, in contrast to usual approaches, ADATE starts with a single individual – the empty function f .

The function declarations of all constructed program candidates use the declaration of f , differing only in the program body. To construct program bodies, only the programming constructs available in ADATE-ML can be used together with additionally data types and primitive functions provided in the problem specification.

The search operators are transformations used in reproduction to generate new individuals. These transformations include: replacements of expressions in the program body, abstraction of expressions by introducing a call to a newly introduced function, distributing a function call currently outside a case expression over all cases, and altering the number and type of function arguments by various embedding techniques. From these ADATE constructs compound transformations, consisting of multiple atomic transformations, depending of the current stage of the search. Through management of an upper bound for the number of compound transformations used to generate individuals, ADATE can employ iterative deepening in its exploration of the problem space. In the current version of ADATE crossover is realized by applying a compound transformation from one individual to another (Vattekar 2006).

In our experiments we used ADATE with the same set of only positive input/output examples which can be presented to the analytical system IGOR2. No data types and function primitives beyond ADATE-ML were used. Performance was evaluated with the predefined measures. Correctness was evaluated by a user-defined all-or-nothing function. The to be learned program f was either presented with an empty body or with one of four variants of initial seeds constructed by analytical induction. A simple example is given in figure 1.

With only the minimum necessary background knowledge, such as necessary data types and atomic functions, ADATE is able to find reasonable solutions given enough time. Additional background knowledge can reduce the required transformations to infer correct solutions, which can also reduce search time. However, additional background knowledge exacts deeper insights into the problem on behalf of the problem specifier. From a programming assistance system perspective, it would be not convenient to expect that a human programmer could provide exactly such knowledge which would increase the speed of generating a solution. This would presuppose (a) that the user understands the in-

Examples (represented in MAUDE notation):

```

sorts elem list.
ops a b c d: -> elem list [ctor].
op nil: -> list [ctor].
op _,_ : elem list -> list [ctor].

eq swap(nil) = nil.
eq swap((a,nil)) = (a,nil).
eq swap((a,b,nil)) = (b,a,nil).
swap((a,b,c,nil)) = (c,b,a,nil).
swap((a,b,c,d,nil)) = (d,b,c,a,nil).

```

Program induced by IGOR2:

```

1. swap( nil ) = nil.
2. swap( (X, XS) ) = (last((X,XS)), sub((X,XS))).
3. last( (X, nil) ) = X.
4. last( (X, XS) ) = last(XS).
5. sub ( (X, nil) ) = nil.
6. sub ( (X,Y,XS) ) = swap((X , sub((Y,XS)))).

```

Best program induced by ADATE with empty seed:

```

fun f Xs =
  case Xs of
    nil => Xs
  | cons( V144C, V144D ) =>
    case V144D of
      nil => Xs
    | cons( V63EC5, V63EC6 ) =>
      case f( V63EC6 ) of
        nil => cons( V63EC5, cons( V144C, V63EC6 ) )
      | cons( V66B8B, V66B8C ) =>
        cons( V66B8B, cons( V63EC5, f( cons( V144C, V66B8C ) ) ) )

```

Figure 1: Swap specified for ADATE and IGOR2 and resulting programs

ner workings of ADATE and (b) has a deep insight in the programming problem at hand. From a cognitive perspective, such additional knowledge to guide ADATE's search might be gained by a closer inspection of the structure of the input/output examples, thereby providing ADATE with a helpful initial hypothesis.

Analytical Inductive Programming with IGOR2

IGOR2 (Kitzelmann 2008) – to our knowledge – is currently the most powerful system for analytical inductive programming. Its scope of inducible programs and the time efficiency of the induction algorithm compares very well with classical approaches to inductive logic programming and other approaches to inductive programming (Hofmann, Kitzelmann, & Schmid 2008). IGOR2 continues the tradition of previous work in learning LISP functions from examples (Summers 1977) as the successor to IGOR1 (Kitzelmann & Schmid 2006).

The system is realized in the constructor term rewriting system MAUDE. Therefore, all constructors specified for the data types used in the given examples are available for program construction. IGOR2 specifications consist of: a small set of positive input/output

examples, presented as equations, which have to be the first examples with respect to the underlying data type and a specification of the input data type. Furthermore, background knowledge for additional functions can (but must not) be provided.

IGOR2 can induce several dependent target functions (i.e., mutual recursion) in one run. Auxiliary functions are invented if needed. In general, a set of rules is constructed by generalization of the input data by introducing patterns and predicates to partition the given examples and synthesis of expressions computing the specified outputs. Partitioning and searching for expressions is done systematically and completely which is tractable even for relatively complex examples because construction of hypotheses is data-driven. An example of a problem specification and a solution produced by IGOR2 is given in figure 1.

Considering hypotheses as equations and applying equational logic, the analytical method assures that only hypotheses entailing the provided example equations are generated. However, the intermediate hypotheses may be unfinished in that the rules contain unbound variables in the rhs, i.e., do not represent functions. The search stops, if one of the currently best hypotheses is finished, i.e., all variables in the rhss are bound.

IGOR2's built-in inductive bias is to prefer fewer case distinctions, most specific patterns and fewer recursive calls. Thus, the initial hypothesis is a single rule per target function which is the least general generalization of the example equations. If a rule contains unbound variables, successor hypotheses are computed using the following operations: (i) Partitioning of the inputs by replacing one pattern by a set of disjoint more specific patterns or by introducing a predicate to the righthand side of the rule; (ii) replacing the righthand side of a rule by a (recursive) call to a defined function (including the target function) where finding the argument of the function call is treated as a new induction problem, that is, an auxiliary function is invented; (iii) replacing subterms in the righthand side of a rule which contain unbound variables by a call to new subprograms.

Refining a Pattern. Computing a set of more specific patterns, case (i), in order to introduce a case distinction, is done as follows: A position in the pattern p with a variable resulting from generalising the corresponding subterms in the subsumed example inputs is identified. This implies that at least two of the subsumed inputs have different constructor symbols at this position. Now all subsumed inputs are partitioned such that all of them with the same constructor at this position belong to the same subset. Together with the corresponding example outputs this yields a partition of the example equations whose inputs are subsumed by p . Now for each subset a new initial hypothesis is computed, leading to one set of successor rules. Since more than one position may be selected, different partitions may be induced, leading to a set of successor rule-sets.

For example, let

$$\begin{aligned} reverse(\[]) &= \[] \\ reverse([X]) &= [X] \\ reverse([X, Y]) &= [Y, X] \end{aligned}$$

be some examples for the *reverse*-function. The pattern of the initial rule is simply a variable Q , since the example input terms have no common root symbol. Hence, the unique position at which the pattern contains a variable and the example inputs different constructors is the root position. The first example input consists of only the constant $\[]$ at the root position. All remaining example inputs have the list constructor *cons* as root. Put differently, two subsets are induced by the root position, one containing the first example, the other containing the two remaining examples. The least general generalizations of the example inputs of these two subsets are $\[]$ and $[Q|Qs]$ resp. which are the (more specific) patterns of the two successor rules.

Introducing (Recursive) Function Calls and Auxiliary Functions. In cases (ii) and (iii) help functions are invented. This includes the generation of I/O-examples from which they are induced. For case (ii) this is done as follows: Function calls are introduced by matching the currently considered outputs, i.e., those outputs whose inputs match the pattern of the currently considered rule, with the outputs of any defined function. If all current outputs match, then the rhs of the current unfinished rule can be set to a call of the matched defined function. The argument of the call must map the currently considered inputs to the inputs of the matched defined function. For case (iii), the example inputs of the new defined function also equal the currently considered inputs. The outputs are the corresponding subterms of the currently considered outputs.

For an example of case (iii) consider the last two *reverse* examples as they have been put into one subset in the previous section. The initial rule for these two examples is:

$$reverse([Q|Qs]) = [Q2|Qs2] \quad (1)$$

This rule is unfinished due to the two unbound variables in the rhs. Now the two unfinished subterms (consisting of exactly the two variables) are taken as new subproblems. This leads to two new example sets for two new help functions *sub1* and *sub2*:

$$\begin{aligned} sub1([X]) &= X & sub2([X]) &= \[] \\ sub1([X, Y]) &= Y & sub2([X, Y]) &= [X] \end{aligned}$$

The successor rule-set for the unfinished rule contains three rules determined as follows: The original unfinished rule (1) is replaced by the finished rule:

$$reverse([Q|Qs]) = [sub1([Q|Qs] | sub2[Q|Qs])]$$

And from both new example sets an initial rule is derived.

Finally, as an example for case (ii), consider the example equations for the help function *sub2* and the generated unfinished initial rule:

$$sub2([Q|Qs]) = Qs2 \quad (2)$$

The example outputs, $\[], [X]$ of *sub2* match the first two example outputs of the *reverse*-function. That is, the unfinished rhs $Qs2$ can be replaced by a (recursive) call to the *reverse*-function. The argument of the call must map the inputs $[X], [X, Y]$ of *sub2* to the corresponding inputs $\[], [X]$ of *reverse*, i.e., a new help function, *sub3* is needed. This leads to the new example set:

$$\begin{aligned} sub3([X]) &= \[] \\ sub3([X, Y]) &= [X] \end{aligned}$$

The successor rule-set for the unfinished rule contains two rules determined as follows: The original unfinished rule (2) is replaced by the finished rule:

$$sub2([Q|Qs]) = reverse(sub3([Q|Qs]))$$

Additionally it contains the initial rule for *sub3*.

Analytically Generated Seeds for Program Evolution

As proposed above, we want to investigate whether using IGOR2 as a preprocessor for ADATE can speed-up ADATE's search for a useful program. Furthermore, it should be the case that the induced program should be as least as efficient as a solution found unassisted by ADATE with respect to ADATE's evaluation function. Obviously, coupling of IGOR2 with ADATE becomes only necessary in such cases where IGOR2 fails to generate a completed program. This occurs if IGOR2 was presented with a too small set of examples or if analytically processing the given set of examples is not feasible within the given resources of memory and time. In these cases IGOR2 terminates with an incomplete program which still contains unbound variables in the body of rules, namely, with missing recursive calls or auxiliary functions.

To have full control over our initial experiments, we only considered problems which IGOR2 can solve fully automatically. We artificially created partial solutions by replacing function calls by unbound variables. We investigated the following strategies for providing ADATE with an initial seed:

For a given ADATE-ML program of the form

```
fun f ( ... ) : myType = raise D1
fun main ( ... ) : myType = f ( ... )
```

- the function f is **redefined** using the partial solution of IGOR2,
- or the problem space becomes **restricted** from the top-level by introducing the partial solution in the function *main*.
- Any IGOR2 induced auxiliary **functions** can also be included: as an atomic, predefined function to be called by f or as an inner function of f also subject to transformations.

Experiments

We presented examples of the following problems to IGOR2:

switch(X) = Y iff the list Y can be obtained from the list X by swapping every element on an odd index in X with the element with the next incremental even index.

sort(X) = Y iff the list Y is a permutation of X with all elements sorted in increasing order.

swap(X) = Y iff the list Y is identical to the list X, except that the first and last element are swapped in around in Y.

lasts(X) = Y iff X is a list of lists and Y is a list containing the last element of each list in X in the order those lists appear in X.

shiftR(X) = Y iff the list Y is identical to the list X, except that the last element in X is on the first position in Y and all other elements are shifted one position to the right.

shiftL(X) = Y iff the list Y is identical to the list X, except that the first element in X is on the last position in Y and all other elements are shifted one position to the left.

insert(X, Y) = Z iff X is a list of elements sorted in an ascending order and Z is a list of elements X + Y sorted in an ascending order.

To generate an initial seed for ADATE, typically the righthand side of a recursive rule was replaced by an unbound variable. For example, the solution for *switch* provided by IGOR2 was

```
switch ( [] ) = []
switch ( [X] ) = [X]
switch ( [X,Y|XS] ) = [Y, X, switch(XS)]
```

and the third rule was replaced by

```
switch ( [X,Y|XS] ) = Z.
```

If IGOR2 induced solutions with auxiliary functions, either the function calls on the righthand side of the rules were made known to ADATE (see section *Analytically Generated Seeds for Program Evolution*) or this information was obscured by again replacing the complete righthand side by a variable.

For example, for *swap*, IGOR2 inferred one atomic function *last* and inferred that the solution consists of two functions that recursively call each other as shown in figure 1. ADATE was presented with the rule 1, 2, 5 and 6 from figure 1 where the righthand side of rule 6 was replaced with an unbound variable.

The results were ascertained by analysing the log files produced to document an ADATE run. To effectively compare the specifications we evaluated each according to the time taken to generate the most correct functions. Because ADATE infers many incorrect programs in the search process, we restricted our focus to those programs that:

- were tested by ADATE against the complete set of given training examples,
- terminated for each training example, and

Table 1: Results for the best strategy (Time in seconds, Execution see text)

Problem	Type	Time	Execution
switch	Unassisted	4.34	302
	Restricted	0.47	344
	Redefined	3.96	302
sort	Unassisted	457.99	2487
	Restricted	225.13	2849
swap	Unassisted	292.05	1076
	Restricted + functions	41.43	685
lasts	Unassisted	260.34	987
	Restricted	6.25	1116
shiftR	Unassisted	8.85	239
	Redefined + functions	1.79	239
shiftL	Unassisted	4.17	221
	Restricted	0.61	281
insert	Unassisted	7.81	176
	Restricted	18.37	240

- generated a correct output for each training example.

This allowed us to achieve a meaningful overview of the performance of the specifications. While an analysis of the inferred programs with poorer performance provides insights into the learning process, it is outside of our scope. Nonetheless, ADATE generates a very complete overview of inferred programs in the log files. For the analysis of the ADATE runs we needed only the following information:

- the elapsed **time** since the start of the search until the creation of the program,
- the breakdown of the results the function produced for the examples, which in our case is the number of results evaluated as correct, incorrect or timed-out. Due to our accepted evaluation restrictions, we filtered out all inferred functions which did not attain 100% correct results with the test examples.
- an ADATE time evaluation of the inferred function. This is the total **execution** time taken by the function for all the test examples as defined by ADATE's built in time complexity measure. This is comparable to a count of all execution operations in runtime, including method and constructor calls and returns.

Because all the specifications designed to solve the same problem included exactly the same examples, we could now compare the respective runs with each other. Table 1 is a summary comparing the most efficient solutions of the unassisted specifications with those of the best specification for the same problem. Included is the problem name, the specification type (either unassisted or the type of assistance), the creation time of the solution, the execution time necessary for the same set of examples.¹

¹To run ADATE only the 1995 ML compiler can be used. The technical details are given in a report

IGOR2 produced solutions with auxiliary functions for the problems *sort*, *swap* and *shiftR*. In the case of *sort* the best result for ADATE was gained by giving no information about the auxiliary functions.

Attention should be drawn to the uniformly quicker inference times achieved by the assisted ADATE specifications with the notable exception of *insert*. Two assisted specifications – *swap* and *shiftR* resulted in better results that were also inferred sooner, whereas the remaining assisted specifications produced results between 14% and 27% less efficient than their unassisted counterparts. All in all, one could summarise, that this relatively small comparative inefficiency is more than compensated by the drastically reduced search time, just over 41 times quicker in the case of *lasts*. That is, our initial experiments are promising and support the idea that search in a generate-and-test approach can be guided by additional knowledge which can be analytically obtained from the examples.

Conclusions

In inductive programming, generate-and-test approaches and analytical, data-driven methods are diametrically opposed. The first class of approaches can in principal generate each possible program given enough time. The second class of approaches has a limited scope of inducible programs but achieves fast induction times and guarantees certain characteristics of the induced programs such as minimality and termination. We proposed to marry these two strategies hoping to combine their respective strengths and get rid of their specific weaknesses. First results are promising since we could show that providing analytically generated program skeletons mostly guides search in such a way that performance times significantly improve.

Since different strategies showed to be most promising for different problems and since for one problem (*insert*) providing an initial solution did result in longer search time, in a next step we hope to identify problem characteristics which allow to determine which strategy of knowledge incorporation into ADATE will be the most successful. Furthermore, we hope either to find a further strategy of knowledge incorporation which will result in speed-up for *insert*, or – in case of failure – come up with additional criteria to determine when to refrain from providing constraining knowledge to ADATE. Our research will hopefully result in a programming assistant which, given a set of examples, can determine whether to use IGOR2 or ADATE stand alone or in combination.

References

- Biermann, A. W.; Guiho, G.; and Kodratoff, Y., eds. 1984. *Automatic Program Construction Techniques*. New York: Macmillan.
- Burstall, R., and Darlington, J. 1977. A transformation system for developing recursive programs. *JACM* 24(1):44–67.
- Green, C.; Luckham, D.; Balzer, R.; Cheatham, T.; and Rich, C. 1983. Report on a knowledge-based software assistant. Technical Report KES.U.83.2, Kestrel Institute, Palo Alto, CA.
- Greeno, J. 1974. Hobbits and orcs: Acquisition of a sequential concept. *Cognitive Psychology* 6:270–292.
- Hofmann, M.; Kitzelmann, E.; and Schmid, U. 2008. Analysis and evaluation of inductive programming systems in a higher-order framework. In Dengel, A.; Berns, K.; Breuel, T. M.; Bomarius, F.; and Roth-Berghofer, T. R., eds., *KI 2008: Advances in Artificial Intelligence (31th Annual German Conference on AI (KI 2008) Kaiserslauten September 2008)*, number 5243 in LNAI, 78–86. Berlin: Springer.
- Kahney, H. 1989. What do novice programmers know about recursion? In Soloway, E., and Spohrer, J. C., eds., *Studying the Novice Programmer*. Lawrence Erlbaum. 209–228.
- Kitzelmann, E., and Schmid, U. 2006. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research* 7(Feb):429–454.
- Kitzelmann, E. 2008. Analytical inductive functional programming. In Hanus, M., ed., *Pre-Proceedings of the 18th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2008, Valencia, Spain)*, 166–180.
- Kruse, R. 1982. On teaching recursion. *ACM SIGCCE-Bulletin* 14:92–96.
- Manna, Z., and Waldinger, R. 1975. Knowledge and reasoning in program synthesis. *Artificial Intelligence* 6:175–208.
- Manna, Z., and Waldinger, R. 1992. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering* 18(8):674–704.
- Newell, A., and Simon, H. 1961. GPS, A program that simulates human thought. In Billing, H., ed., *Lernende Automaten*. München: Oldenbourg. 109–124.
- Olsson, R. 1995. Inductive functional programming using incremental program transformation. *Artificial Intelligence* 74(1):55–83.
- Pirolli, P., and Anderson, J. 1985. The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology* 39:240–272.
- Summers, P. D. 1977. A methodology for LISP program construction from examples. *Journal ACM* 24(1):162–175.
- Vattekarak, G. 2006. Adate User Manual. Technical report, Ostfold University College.

The China-Brain Project

Report on the First Six Months

**Prof. Dr. Hugo de GARIS, Dean Prof. Dr. ZHOU Changle, Prof. Dr. SHI Xiaodong,
Dr. Ben GOERTZEL, Prof. PAN Wei, Prof. MIAO Kehua,
Prof. Dr. ZHOU Jianyang, Dr. JIANG Min, Prof. ZHEN Lingxiang,
Prof. Dr. WU Qinfang, Prof. Dr. SHI Minghui, LIAN Ruiting, CHEN Ying**

Artificial Brain Lab
Cognitive Science Department
School of Information Science and Technology
Xiamen University, Xiamen, China

profhugodegaris@yahoo.com

Abstract

The “China Brain Project” is a 4 year (2008-2011), 10.5 million RMB research project to build China’s first artificial brain, which will consist of 10,000-50,000 neural net modules which are evolved rapidly in special FPGA hardware, downloaded one by one into a PC or supercomputer, and then connected according to the designs of human “BAs” (brain architects) to build an artificial brain with thousands of pattern detectors to control the hundreds of behaviors of a two legged robot.

1. Introduction

The “China Brain Project”, based at Xiamen University, is a 4 year (2008-2011), 10.5 million RMB, 20 person, research project to design and build China’s first artificial brain (AB). An artificial brain is defined here to be a “network of (evolved neural) networks”, where each neural net(work) module performs some simple task (e.g. recognizes someone’s face, lifts an arm of a robot, etc), somewhat similar to Minsky’s idea of a “society of mind” [1], i.e. where large numbers of unintelligent “agents” link up to create an intelligent “society of agents”. 10,000s of these neural net modules are evolved rapidly, one at a time, in special (FPGA based) hardware and then downloaded into a PC (or more probably, a supercomputer PC cluster). Human “BAs” (brain architects) then connect these evolved modules according to their human designs to architect artificial brains. Special

software, called IMSI (see section 5) is used to specify these connections, and to perform the neural signaling of the whole brain (in real time). The main aim of this research project is to show that using this (evolutionary engineering) approach to brain building is realistic, by simply building one and show that it can have thousands of pattern recognizers, and hundreds of motions that are switched between, depending on external and internal stimuli. This project already has (Fujian) province “key lab” financial support. It is hoped, in three years, that it will win “key state (i.e. federal) lab” status. In 5-10 years, it is hoped that China will establish a “CABA” (Chinese Artificial Brain Administration), consisting of thousands of scientists and engineers, to build national brains to fuel the home robot industry (which may become the worlds largest) (See section 11.)

There are about 20 people (8 professors) involved in this project, divided into specialist teams, i.e.

- a) “Vision team” (who evolve pattern recognition modules and create vision system architectures).
- b) “Robot team” (who program the NAO robot [2] to perform the many (hundreds of) behaviors that the robot is to perform).
- c) “Hardware Acceleration team” (who program the FPGA electronic boards we use to

evolve neural net modules as quickly as possible).

d) “Supercomputer team” (who port the Parcone and IMSI code to a supercomputer, to accelerate the neural evolution and signaling of the artificial brain).

e) “Language team” (who give the robot language capabilities, i.e. speech, listening, understanding)

f) “Consciousness team” (who aim to give the NAO robot some degree of self awareness).

By the end of this 4 year project, we hope to be able to show the world an artificial brain, consisting of 10,000s of evolved neural net modules that control the 100s of behaviors of a NAO robot (and probably a more sophisticated robot) that makes a casual observer feel that the robot “has a brain behind it”. It also hoped that this artificial brain project will encourage other research teams to work in this new area, as well as help establish an artificial brain industry that will stimulate the growth of the home robot industry, probably the world’s largest by 2030.

2. The “Parcone” (Partially Connected Neural Evolutionary) Neural Net Model

If one chooses to build an artificial brain based on the “evolutionary engineering” of neural net modules, then the choice of the neural net model that one uses to evolve the modules is critical, since everything else follows from it. Hence quite some thought went into its choice. We eventually decided upon a partially connected model (that we called the “Parcone” (i.e. partially connected neural evolutionary) model, since we wanted to be able to input images from digital cameras that started off as mega-pixel images, which were then compressed to 1000s to 10,000s of pixels. In earlier work, the first author [3], had always used fully connected neural networks for his neural net evolution work, but with 10,000s of pixel inputs (with one pixel per input neuron) a fully connected neuron would have an unreasonably large number of connections (i.e. hundreds of millions).

The moment one chooses a partially connected neural net model, one must then keep a list of all

the neurons that each neuron connects to. This we did in the form of a hash table. See the data structures of Fig. 2. Each neuron that is connected to from a given neuron has a unique integer ID that is hashed to find its index in the hash table of the given neuron. This hash table slot contains a pointer to a struct that contains the integer ID of the neuron connected to, the weight bits of the connection, and the decimal weight value.

These connections and weight values are used to calculate the neural output signal of the given neuron. The weight bits are mutated during the evolution of the neural net, as well as the connections, by adding and cutting them randomly. The model contained many parameters that are chosen by the user, e.g. the number of input, middle, and output neurons, the number of weight-sign bits, the number of hash table slots, the population size, mutation rates, etc. These parameters were “tuned” empirically for maximum evolution speed.

3. Pattern Detection Results

Once the Parcone code was written and debugged, the first pattern recognition task we undertook was to see how well it could recognize faces. Fig. 3 shows an example of the face inputs we used. We took photos of 6 people, with 5 images of each person at different angles, as Fig. 3 shows. 3 of these images of a given person were used as the positive cases in the training set, plus 3 each of two other people, as the negative cases in the training set. The Parcone neural net was evolved to output a strong positive neural signal for the positive cases, and a strong negative signal for the negative cases. When the other positive images not seen by the evolved module were input, the outputs were strong positive, so the module generalized well.

We then automated the pattern detection so that P positive images could be used in the evolution, and N negative images. Users could then select from a large menu of images the P positive and N negative images they wanted in the evolution. We evolved shoe detectors, mobile phone detectors etc. in a similar way. When a shoe detector was presented with a face, it rejected it (i.e. it output a negative neural signal) in about 95% of cases, and vice versa. Thus we found that when a detector for object “X” was evolved, it rejected objects of type “not X”. This ability of

the Parcone model will hopefully prove to be very useful for constructing the 1000s of pattern detectors for the artificial brain.

At the time of writing (Oct 2008) tests are currently underway by the vision group to evolve *motion* detectors. A moving image is input as a set of “movie frames”. Once we have the Parcone model implemented in FPGA based electronics (see section 6) we hope to be able to evolve pattern detectors (whether stationary or moving) in real time (i.e. in about 1 second).

4. The NAO (Robocup Robot Standard) Robot

Fig. 1 shows the NAO robot, a product of the French company Aldebaran [2], in Paris. It is of waist height, costs about \$20,000, can walk on its two legs, talk, listen, grip with a thumb and two fingers, and has one eye. It is now the Robocup robot standard, after Sony stopped supporting their Aibo robo-pet dog, which was the previous Robocup robot standard. Interestingly for our project, the NAO robot (which means “brain” in Chinese by the way (coincidence? marketing?)) comes with accompanying motion control software, called “Choregraphe” which we have chosen to use, rather than try to evolve motion control for all the 25 motors that the NAO robot possesses. We expect to have hundreds of motions for the NAO robot so that it can accomplish many tasks that the artificial brain initiates.

We are acutely conscious that no one will actually “see” the artificial brain, since it will consist of 10,000s of neural net modules hidden inside the PC or supercomputer that performs the neural signaling of the artificial brain. All that human observers will see will be the robot, that the artificial brain controls, so we are hoping that when observers watch the hundreds of behaviors of the robot, with its 1000s of pattern detectors, they will have the impression that the NAO robot “has a brain behind it” and be suitably impressed (at least enough for the funding of the research project to be continued). Later in the project, we intend building a more sophisticated robot with two eyes, and hands with better fingers, capable of real grasping, with touch sensors, etc. The project has a dedicated “robot group” who work on generating its motions, and control.

5. IMSI (Inter Module Signaling Interface)

IMSI stands for “inter module signaling interface”, i.e. the software “operating system” that is used for several purposes, namely :-

a) It allows the “BAs” (brain architects, i.e. the people who decide which neural net modules to evolve (i.e. their fitness definitions, etc) and the architectures of artificial brains) to specify the connections between the modules (e.g. the output of module M2849 (which performs task “X”) connects to the 2nd input of module M9361 (which performs task “Y”). Such information is stored in special look up tables (LUTs).

b) These LUTs are then used to allow the IMSI to perform the neural signaling of the whole artificial brain. When the output signal is being calculated for a particular module, it needs to know the values of the neural signals it is getting from other modules, and to which modules to send its output signal.

The IMSI calculates the output neural signal values of each module sequentially, for all modules in the artificial brain. Placing dummy weight values for about 1000 connections per module, allowed us to use a PC to determine how many such “dummy” modules could have their neural output signals calculated sequentially in “real time” (defined to be 25 output signals for every *neuron* in the artificial brain). The answer was 10,000s depending on the speed of the PC. Since we have a 10-20 PC node supercomputer cluster at our disposal, we can realistically envision building an artificial brain with several 10,000s of neural net modules.

At first, we envisioned that the artificial brain would consist solely of evolved neural net modules, interconnected appropriately to generate the functionality we desired. However the decision mentioned in section 4 (on the NAO robot) that we would use Aldebaran’s “Choregraphe” software to control the motions of the robot’s many behaviors, implies that the IMSI will be a hybrid of neural net modules and motion control routines written with Choregraphe.

The IMSI software will be designed and coded in November of 2008, allowing the first “micro-

brain”, consisting of some dozen or so modules, to be designed and tested.

6. FPGA Based Parcone Module Evolution

The accelerator group is using 3 FPGA electronic boards to evolve neural net modules (based on the Parcone model) as quickly as possible, hopefully in real time (i.e. in about a second). Real time evolution will allow continuous learning of the artificial brain, so evolution speed has always been a dominant factor in our research. If these FPGA boards prove to be too slow, we may try a hybrid analog-digital approach, where the neural signaling is done using analog neurons based on Prof. Chua’s “CNN” (cellular neural networks) [4], and the evolution is controlled digitally. This latter approach will demand a much higher learning curve, so will not be undertaken if the FPGA board approach proves to be sufficient.

7. The Language Component

The NAO robot is to be given language capabilities. Prof. SHI Xiaodong and Dr. Ben GOERTZEL are in charge of the “Language team”. The NAO robot (actually, the artificial brain) is to be made capable of speaking, listening, and understanding spoken commands and answering spoken questions. This will involve speech to text and text to speech conversion, which will probably use standard “off the shelf” products. The research effort will be based more on language understanding, parsing, etc. The aim is to be able to give the robot spoken commands, e.g. “Go to the door”. “Pick up the pencil on the table”, etc. The robot should also be capable of giving verbal replies to simple questions, e.g. the question “Where is Professor X” might get an answer “Near the window”.

8. The Consciousness (Self Awareness) Component

Dean Zhou Changle, (dean of the School of Information Science and Technology) at Xiamen University, is responsible for the consciousness (self awareness) component of the project. At the time of writing (Oct 2008), this component is

still under consideration. The dean is keen on this component, even referring to this project as the “Conscious Robot Project”.

9. Near Future Work

At the time of writing (Oct 2008), the China Brain Project is only about 6 months old, so there is not a lot to report on in terms of completed work. Now that the evolvable neural net model (Parcone) is complete and tested, the most pressing task is to put a (compressed) version of it into the FPGA boards and (hopefully) speed up the evolution of a neural net (Parcone) module so that it takes less than a second. This “real time” evolution then opens up an exciting prospect. It would allow “real time” continuous learning. For example – imagine the robot sees an object it has never seen before. All the pattern recognition circuits it has already stored in its artificial brain give weak, i.e. negative output signals. Hence the robot brain can detect that the object is not recognized, hence a new pattern detector circuit can then be learned in real time and stored.

The robot group will use the Choregraphe software to generate hundreds of different behaviors of the NAO robot.

The vision group will continue testing the Parcone model for evolving pattern detectors, e.g. detecting motion (e.g. distinguishing objects moving left from those moving right, between those moving towards the eye of the robot quickly, from those that are moving slowly, etc). There may be thousands of pattern detectors in the artificial brain by the time the project contract finishes at the end of 2011.

We hope to have integrated a language component by March of 2009 (before the AGI-09 conference), so that we can have the robot obey elementary spoken commands, e.g. “move to the door”, “point to the window”, “what is my name?” etc.

Also by then, the aims of the “consciousness (self awareness) component” should be better clarified and whose implementation should have begun.

10. Goals for the Next Few Years

The first year is devoted largely to *tool building* (e.g. choosing and coding the evolvable neural net model, testing its evolvability as a pattern detector, implementing the Parcone model in the FPGA boards, programming the motions of the NAO robot with the Choregraphe software, writing the IMSI code, etc). In the second year, the first artificial brain *architectures* will be created and implemented, with 10, 20, 50, 100, etc modules. The language component will be added to the brain. At regular intervals, demos will be built, to show off progress. We expect that each scaling up of the size of the artificial brain (i.e. each substantial increase in the number of modules in the brain) will raise new challenges that will have to be overcome by the creativity of the team's BAs (brain architects). At the end of 4 years, it is hoped that the artificial brain will have 10,000s of modules. Observing a robot controlled by such an artificial brain should make a casual observer feel that the robot "has a brain behind it".

11. Artificial Brain Research Policy in China

The first author thinks that the artificial brain industry will be the world's biggest by about 2030, because artificial brains will be needed to control the home robots that everyone will be prepared to spend big money on, if they become genuinely intelligent and hence useful (e.g. baby sitting the kids, taking the dog for a walk, cleaning the house, washing the dishes, reading stories, educating its owners etc). China has been catching up fast with the western countries for decades. The first author thinks that China should now aim to start leading the world (given its huge population, and its 3 times greater average economic growth rate compared to the US) by aiming to dominate the artificial brain industry. At the time of writing (Oct 2008), plans are afoot (with the support of the most powerful people in China in artificial intelligence) to attempt to persuade the Chinese Ministers of Education and of Science and Technology to undertake a long term strategy (over a 20 year time frame) to dominate the global artificial brain industry, by initially stimulating the establishment of artificial brain (and intelligence science) labs in universities across China (somewhat similar to our Artificial Brain Lab at

Xiamen), then awarding "key lab" status at both province and national levels to some of these labs, the creation of a Chinese National Artificial Brain Association, and especially, within a 5 to 10 year time frame, the establishment of a "CABA" (Chinese Artificial Brain Administration), which would be a government administration (similar to America's NASA) that would employ thousands of engineers and scientists to build artificial brains for the Chinese artificial brain industry. Copying the human brain with its 100 billion neurons and quadrillion synapses will be an immense task requiring large numbers of Chinese "brain workers". It is expected that other countries will quickly copy China's lead, so that one will soon see national brain building projects in most of the leading high tech nations.

References

[1] Marvin MINSKY, "Society of Mind", Simon & Schuster, 1988.

[2] <http://www.aldebaran-robotics.com/eng/index.php>

[3] Hugo de Garis, in book, Artificial General Intelligence 2008 : Proceedings of the First AGI Conference (Frontiers in Artificial Intelligence and Applications) (Frontiers in Artificial Intelligence and Applications)

[4] Leon CHUA and Tamas ROSKA, "Cellular Neural Networks and Visual Computing, Foundations and Applications", Cambridge University Press, 2002.



Fig. 1 The NAO Robot Controlled by Our Artificial Brain

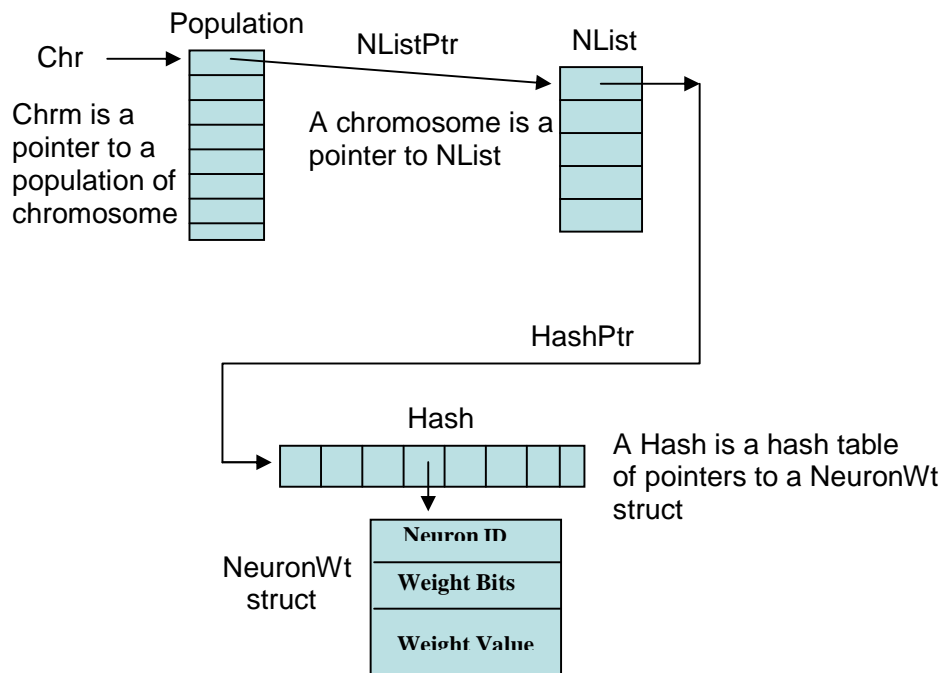
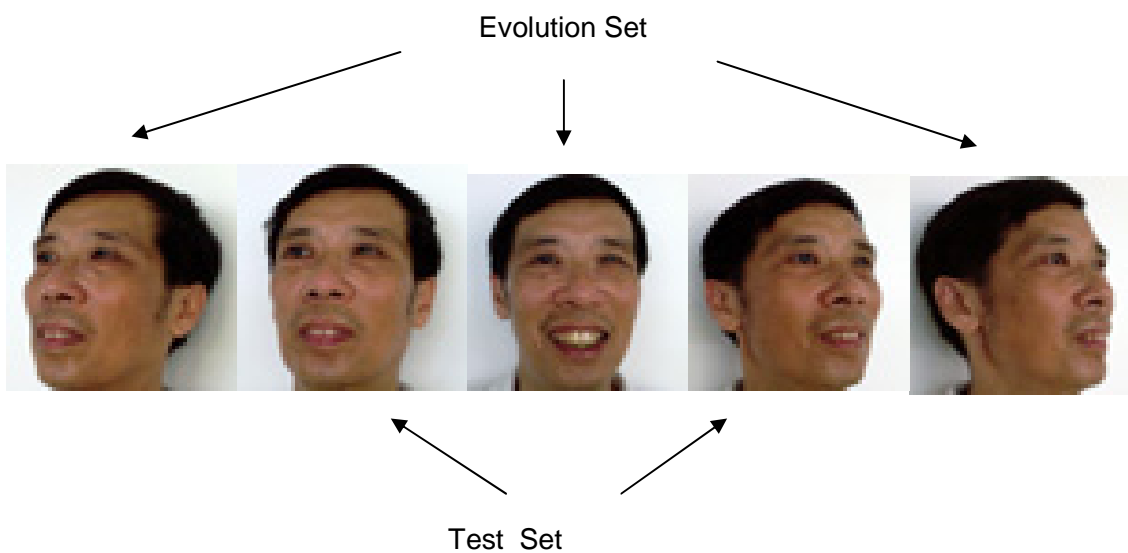


Fig. 2 Data Structures of the “Parcone” Neural Net Model



60*60 (pixel)*3 (RGB) = 10800 pixel values = 10800 input neurons to Parcone

Fig. 3 Face Images for Face Detection Evolution of Parcone

AGI Preschool: A Framework for Evaluating Early-Stage Human-like AGIs

Ben Goertzel, Stephan Vladimir Bugaj

Novamente LLC & Singularity Institute for AI, AGI Research Institute
1405 Bernerd Place, Rockville MD 20851, USA
ben@goertzel.org, stephan@bugaj.com

Abstract

A class of environments for teaching and evaluating AGI systems is described, modeled loosely on preschools used for teaching human children and intended specifically for early-stage systems aimed at approaching human-level, human-inspired AGI through a process resembling human developmental psychology. Some particulars regarding implementation of environments in this class in online virtual worlds are presented, along with details on associated evaluation tasks, and discussion of environments potentially appropriate for “AGI preschool graduates,” based on more advanced human schooling.

Introduction

One of the many difficult issues arising in the course of research on human-level AGI is that of “evaluation and metrics” – i.e., AGI intelligence testing. The Turing test (Turing, 1950) approaches the question of how to tell if an AGI has achieved human-level intelligence, and while there are debates and subtleties even with this well-known test (Moor, 2001; French, 1990; Hayes and Ford, 1995), the question we’ll address here is a significantly trickier one: assessing the quality of *incremental* progress toward human-level AI.

(Laird, et al, 2009) discusses some of the general difficulties involved in this type of assessment, and some requirements that any viable approach must fulfill. Here, rather than surveying the spectrum of possibilities, we will focus on describing in detail one promising approach: emulation, in a multiuser online virtual world, of an environment similar to preschools used in early human childhood education. Complete specification of an “AGI Preschool” would require much more than a brief conference paper; our goal here is to sketch the idea in broad outline, and give a few examples of the types of opportunities such an environment would afford for instruction, spontaneous learning and formal and informal evaluation of certain sorts of early-stage AGI systems.

The Need for “Realistic” AGI Testing

One might question the need or importance of a new, overall framework for AGI intelligence testing, such as the

AGI Preschool approach described here. After all, there has already been a lot of work on evaluating the capability of various AI systems and techniques. However, we believe that the approaches typically taken have significant shortcomings from an AGI perspective.

Certainly, the AI field has inspired many competitions, each of which tests some particular type or aspect of intelligent behavior. Examples include robot competitions, tournaments of computer chess, poker, backgammon and so forth at computer olympiads, trading-agent competition, language and reasoning competitions like the Pascal Textual Entailment Challenge, and so on. In addition to these, there are many standard domains and problems used in the AI literature that are meant to capture the essential difficulties in a certain class of learning problems: standard datasets for face recognition, text parsing, supervised classification, theorem-proving, question-answering and so forth.

However, the value of these sorts of tests for AGI is predicated on the hypothesis that the degree of success of an AI program at carrying out some domain-specific task, is correlated the the potential of that program for being developed into a robust AGI program with broad intelligence. If AGI and problem-area-specific “narrow AI” are in fact very different sorts of pursuits requiring very different principles, as we suspect (Goertzel and Pennachin, 2006), then these tests are not strongly relevant to the AGI problem.

There are also some standard evaluation paradigms aimed at AI going beyond specific tasks. For instance, there is a literature on “multitask learning,” where the goal for an AI is to learn one task quicker given another task solved previously (Caruna, 1997; Thrun and Mitchell, 1995; Ben-David and Schuller, 2003; Taylor and Stone, 2007). This is one of the capabilities an AI agent will need to simultaneously learn different types of tasks as proposed in the Preschool scenario given here. And there is a literature on “shaping,” where the idea is to build up the capability of an AI by training it on progressively more difficult versions of the same tasks (Laud and Dejong, 2003; Walsh and Littman, 2006). Again, this is one sort of capability an AI will need to possess if it is to move up some type of curriculum, such as a school curriculum.

While we applaud the work done on multitask learning

and shaping, we feel that exploring these processes using mathematical abstractions, or in the domain of various machine-learning or robotics test problems, may not adequately address the problem of AGI. The problem is that generalization among tasks, or from simpler to more difficult versions of the same task, is a process whose nature may depend strongly on the overall nature of the set of tasks and task-versions involved. Real-world tasks have a subtlety of interconnectedness and developmental course that is not captured in current mathematical learning frameworks nor standard AI test problems.

To put it mathematically, we suggest that the universe of real-world human tasks has a host of “special statistical properties” that have implications regarding what sorts of AI programs will be most suitable; and that, while exploring and formalizing the nature of these statistical properties is important, an easier and more reliable approach to AGI testing is to create a testing environment that embodies these properties implicitly, via its being an emulation of the cognitively meaningful aspects of the real-world human learning environment.

One way to see this point vividly is to contrast the current proposal with the “General Game Player” AI competition, in which AIs seek to learn to play games based on formal descriptions of the rules.¹ Clearly doing GGP well requires powerful AGI; and doing GGP even mediocly probably requires robust multitask learning and shaping. But we suspect GGP is far inferior to AGI Preschool as an approach to testing early-stage AI programs aimed at roughly humanlike intelligence. This is because, unlike the tasks involved in AI Preschool, the tasks involved in doing simple instances of GGP seem to have little relationship to humanlike intelligence or real-world human tasks.

Multiple Intelligences

Intelligence testing is, we suggest, best discussed and pursued in the context of a theoretical interpretation of “intelligence.” As there is yet no consensus on the best such interpretation (Legg and Hutter (2006) present a summary of over 70 definitions of intelligence presented in the research literature), this is a somewhat subtle point.

In our own prior work we have articulated a theory based on the notion of intelligence as “the ability to achieve complex goals in complex environments,” a definition that relates closely to Legg and Hutter’s (2007) more rigorous definition in terms of statistical decision theory. However, applying this sort of theory to practical intelligence testing seems very difficult, in that it requires an assessment of the comparative complexity of various real-world tasks and environments. As real-world tasks and environments are rarely well-formalized, one’s only pragmatic option is to assess complexity in terms of human common sense or natural language, which is an approach fraught with “hidden rocks,” though it might prove fruitful

if pursued with sufficient care and effort.

Here we have chosen an alternate, complementary approach, choosing as our inspiration Gardner’s (1983) multiple intelligences (MI) framework -- a psychological approach to intelligence assessment based on the idea that different people have mental strengths in different high-level domains, so that intelligence tests should contain aspects that focus on each of these domains separately. MI does not contradict the “complex goals in complex environments” view of intelligence, but rather may be interpreted as making specific commitments regarding which complex tasks and which complex environments are most important for roughly human-like intelligence.

MI does not seek an extreme generality, in the sense that it explicitly focuses on domains in which humans have strong innate capability as well as general-intelligence capability; there could easily be non-human intelligences that would exceed humans according to both the commonsense human notion of “general intelligence” and the generic “complex goals in complex environments” or Hutter/Legg-style definitions, yet would not equal humans on the MI criteria. This strong anthropocentrism of MI is not a problem from an AGI perspective so long as one uses MI in an appropriate way, i.e. only for assessing the extent to which an AGI system displays specifically *human-like* general intelligence. This restrictiveness is the price one pays for having an easily articulable and relatively easily implementable evaluation framework.

Table 1 summarizes the types of intelligence included in Gardner’s MI theory. Later on, we will suggest that one way to assess an AGI Preschool implementation is to ask how well it covers all the bases outlined in MI theory.

Intelligence Type	Aspects
Linguistic	words and language, written and spoken; retention, interpretation and explanation of ideas and information via language, understands relationship between communication and meaning
Logical-Mathematical	logical thinking, detecting patterns, scientific reasoning and deduction; analyse problems, perform mathematical calculations, understands relationship between cause and effect towards a tangible outcome
Musical	musical ability, awareness, appreciation and use of sound; recognition of tonal and rhythmic patterns, understands relationship between sound and feeling
Bodily-Kinesthetic	body movement control, manual dexterity, physical agility and balance; eye and body coordination
Spatial-Visual	visual and spatial perception; interpretation and creation of images; pictorial imagination and expression; understands relationship between images and meanings, and between space and effect
Interpersonal	perception of other people’s feelings; relates to others; interpretation of behaviour and communications; understands relationships between people and their situations

Table 1. Types of Intelligence, Aspects and Testing

¹ <http://games.stanford.edu/>

Elements of Preschool Design

What we mean by an “AGI Preschool” is simply a porting to the AGI domain of the essential aspects of human preschools. While there is significant variance among preschools there are also strong commonalities, grounded in educational theory and experience. We will briefly discuss both the physical design and educational curriculum of the typical human preschool, and which aspects transfer effectively to the AGI context.

On the physical side, the key notion in modern preschool design is the “learning center,” an area designed and outfitted with appropriate materials for teaching a specific skill. Learning centers are designed to encourage learning by doing, which greatly facilitates learning processes based on reinforcement, imitation and correction (see Goertzel et al (2008) for a discussion of the importance of this combination in an AGI context); and also to provide multiple techniques for teaching the same skills, to accommodate different learning styles and prevent overfitting and overspecialization in the learning of new skills.

Centers are also designed to cross-develop related skills. A “manipulatives center,” for example, provides physical objects such as drawing implements, toys and puzzles, to facilitate development of motor manipulation, visual discrimination, and (through sequencing and classification games) basic logical reasoning. A “dramatics center,” on the other hand, cross-trains interpersonal and empathetic skills along with bodily-kinesthetic, linguistic, and musical skills. Other centers, such as art, reading, writing, science and math centers are also designed to train not just one area, but to center around a primary intelligence type while also cross-developing related areas. For specific examples of the learning centers associated with particular contemporary preschools, see (Neilsen, 2006).

In many progressive, student-centered preschools, students are left largely to their own devices to move from one center to another throughout the preschool room. Generally, each center will be staffed by an instructor at some points in the day but not others, providing a variety of learning experiences. At some preschools students will be strongly encouraged to distribute their time relatively evenly among the different learning centers, or to focus on those learning centers corresponding to their particular strengths and/or weaknesses.

Elements of Preschool Curriculum

While preschool curricula vary considerably based on educational philosophy and regional and cultural factors, there is a great deal of common, shared wisdom regarding the most useful topics and methods for preschool teaching. Guided experiential learning in diverse environments and using varied materials is generally agreed upon as being an optimal methodology to reach a wide variety of learning types and capabilities. Hands-on learning provides grounding in specifics, where as a diversity of approaches allows for generalization.

Core knowledge domains are also relatively consistent, even across various philosophies and regions. Language, movement and coordination, autonomous judgment, social skills, work habits, temporal orientation, spatial orientation, mathematics, science, music, visual arts, and dramatics are universal areas of learning which all early childhood learning touches upon. The particulars of these skills may vary, but all human children are taught to function in these domains. The level of competency developed may vary, but general domain knowledge is provided. For example, most kids won’t be the next Maria Callas, Ravi Shankar or Gene Ween, but nearly all learn to hear, understand and appreciate music.

<i>Type of Capability</i>	<i>Specific Skills to be Evaluated</i>
Story Understanding	<ul style="list-style-type: none"> • Understanding narrative sequence • Understanding character development • Dramatize a story • Predict what comes next in a story
Linguistic	<ul style="list-style-type: none"> • Give simple descriptions of events • Describe similarities and differences • Describe objects and their functions
Linguistic / Spatial-Visual	<ul style="list-style-type: none"> • Interpreting pictures
Linguistic / Social	<ul style="list-style-type: none"> • Asking questions appropriately • Answering questions appropriately • Talk about own discoveries • Initiate conversations • Settle disagreements • Verbally express empathy • Ask for help • Follow directions
Linguistic / Scientific	<ul style="list-style-type: none"> • Provide possible explanations for events or phenomena • Carefully describe observations • Draw conclusions from observations
Logical-Mathematical	<ul style="list-style-type: none"> • Categorizing • Sorting • Arithmetic • Performing simple “proto-scientific experiments”
Nonverbal Communication	<ul style="list-style-type: none"> • Communicating via gesture • Dramatizing situations • Dramatizing needs, wants • Express empathy
Spatial-Visual	<ul style="list-style-type: none"> • Visual patterning • Self-expression through drawing • Navigate
Objective	<ul style="list-style-type: none"> • Assembling objects • Disassembling objects • Measurement • Symmetry • Similarity between structures (e.g. block structures and real ones)
Interpersonal	<ul style="list-style-type: none"> • Cooperation • Display appropriate behavior in various settings • Clean up belongings • Share supplies
Emotional	<ul style="list-style-type: none"> • Delay gratification • Control emotional reactions • Complete projects

Table 2. Preschool cognitive tests

Table 2 reviews the key capabilities taught in preschools, and identifies the most important specific skills that need to be evaluated in the context of each capability. This table was assembled via surveying the curricula from a number of currently existing preschools employing different methodologies both based on formal academic cognitive theories (Schunk 2007) and more pragmatic approaches, such as: Montessori (Montessori, 1912), Waldorf (Steiner 2003), Brain Gym (www.braingym.org) and Core Knowledge (www.coreknowledge.org).

Preschool in the Light of Intelligence Theory

Comparing Table 2 to the Multiple Intelligences framework, the high degree of harmony is obvious, and is borne out by more detailed analysis which is omitted here for space reasons. Preschool curriculum as standardly practiced is very well attuned to MI, and naturally covers all the bases that Gardner identifies as important. And this is not at all surprising since one of Gardner’s key motivations in articulating MI theory was the pragmatics of educating humans with diverse strengths and weaknesses.

Regarding intelligence as “the ability to achieve complex goals in complex environments,” it is apparent that preschools are specifically designed to pack a large variety of different micro-environments (the learning centers) into a single room, and to present a variety of different tasks in each environment. The environments constituted by preschool learning centers are designed as microcosms of the most important aspects of the environments faced by humans in their everyday lives.

Task-Based Assessment in AGI Preschool

Professional pedagogues such as (Chen & McNamee, 2007) discuss evaluation of early childhood learning as intended to assess both specific curriculum content knowledge as well as the child's learning process. It should be as unobtrusive as possible, so that it just seems like another engaging activity, and the results used to tailor the teaching regimen to use different techniques to address weaknesses and reinforce strengths.

For example, with group building of a model car, students are tested on a variety of skills: procedural understanding, visual acuity, motor acuity, creative problem solving, interpersonal communications, empathy, patience, manners, and so on. With this kind of complex, yet engaging, activity as a metric the teacher can see how each student approaches the process of understanding each subtask, and subsequently guide each student's focus differently depending on strengths and weaknesses.

Next we describe some particular tasks that AGIs may be meaningfully assigned in the context of a general AGI Preschool design and curriculum as described above. Due to length limitations this is a very partial list, and is intended as evocative rather than comprehensive.

Any one of these tasks can be turned into a rigorous

quantitative test, thus allowing the precise comparison of different AGI systems’ capabilities; but we have chosen not to emphasize this point here, partly for space reasons and partly for philosophical ones. In some contexts the quantitative comparison of different systems may be the right thing to do, but as discussed in (Laird et al, 2008) there are also risks associated with this approach, including the emergence of an overly metrics-focused “bakeoff mentality” among system developers, and overfitting of AI abilities to test taking. What is most important is the isolation of specific tasks on which different systems may be experientially trained and then qualitatively assessed and compared, rather than the evaluation of quantitative metrics.

Table 3 lists a sampling of different tests for each intelligence type to be assessed.

Intelligence Type	Test
Linguistic	write a set of instructions; speak on a subject; edit a written piece or work; write a speech; commentate on an event; apply positive or negative 'spin' to astory
Logical-Mathematical	perform arithmetic calculations; create a process to measure something; analyse how a machine works; create a process; devise a strategy to achieve an aim; assess the value of a proposition
Musical	perform a musical piece; sing a song; review a musical work; coach someone to play a musical instrument
Bodily-Kinesthetic	juggle; demonstrate a sports technique; flip a beer-mat; create a mime to explain something; toss a pancake; fly a kite
Spatial-Visual	design a costume; interpret a painting; create a room layout; create a corporate logo; design a building; pack a suitcase or the boot of a car
Interpersonal	interpret moods from facial expressions; demonstrate feelings through body language; affect the feelings of others in a planned way; coach or counsel another

Table 3. Prototypical intelligence assessment tasks.

Task-oriented testing allows for feedback on applications of general pedagogical principles to real-world, embodied activities. This allows for iterative refinement based learning (shaping), and cross development of knowledge acquisition and application (multitask learning). It also helps militate against both cheating, and over-fitting, as teachers can make ad-hoc modifications to the tests to determine if this is happening and correct for it if necessary.

E.g., consider a linguistic task in which the AGI is required to formulate a set of instructions encapsulating a given behavior (which may include components that are physical, social, linguistic, etc.). Note that although this is presented as centrally a linguistic task, it actually involves a diverse set of competencies since the behavior to be described may encompass multiple real-world aspects.

To turn this task into a more thorough test one might involve a number of human teachers and a number of human students. Before the test, an ensemble of copies of the AGI would be created, with identical knowledge state. Each copy would interact with a different human teacher, who would demonstrate to it a certain behavior. After testing the AGI on its own knowledge of the material, the teacher would then inform the AGI that it will then be tested on its ability to verbally describe this behavior to another. Then, the teacher goes away and the copy interacts with a series of students, attempting to convey to the students the instructions given by the teacher.

The teacher can thereby assess both the AGI's understanding of the material, and the ability to explain it to the other students. This separates out assessment of understanding from assessment of ability to communicate understanding, attempting to avoid conflation of one with the other. The design of the training and testing needs to account for potential

This testing protocol abstracts away from the particularities of any one teacher or student, and focuses on effectiveness of communication in a human context rather than according to formalized criteria. This is very much in the spirit of how assessment takes place in human preschools (with the exception of the copying aspect): formal exams are rarely given in preschool, but pragmatic, socially-embedded assessments are regularly made.

By including the copying aspect, more rigorous statistical assessments can be made regarding efficacy of different approaches for a given AGI design, independent of past teaching experiences. The multiple copies may, depending on the AGI system design, then be able to be reintegrated, and further "learning" be done by higher-order cognitive systems in the AGI that integrate the disparate experiences of the multiple copies.

This kind of parallel learning is different from both sequential learning that humans do, and parallel presences of a single copy of an AGI (such as in multiple chat rooms type experiments). All three approaches are worthy of study, to determine under what circumstances, and with which AGI designs, one is more successful than another.

It is also worth observing how this test could be tweaked to yield a test of generalization ability. After passing the above, the AGI could then be given a description of a new task (acquisition), and asked to explain the new one (variation). And, part of the training behavior might be carried out unobserved by the AGI, thus requiring the AGI to infer the omitted parts of the task it needs to describe.

Another popular form of early childhood testing is puzzle block games. These kinds of games can be used to assess a variety of important cognitive skills, and to do so in a fun way that not only examines but also encourages creativity and flexible thinking. Types of games include pattern matching games in which students replicate patterns described visually or verbally, pattern creation games in which students create new patterns guided by visually or verbally described principles, creative interpretation of patterns in which students find meaning in

the forms, and free-form creation. Such games may be individual or cooperative.

Cross training and assessment of a variety of skills occurs with pattern block games: for example, interpretation of visual or linguistic instructions, logical procedure and pattern following, categorizing, sorting, general problem solving, creative interpretation, experimentation, and kinematic acuity. By making the games cooperative, various interpersonal skills involving communication and cooperation are also added to the mix.

The puzzle block context bring up some general observations about the role of kinematic and visuospatial intelligence in the AGI Preschool. Outside of robotics and computer vision, AI research has often downplayed these sorts of intelligence (though, admittedly, this is changing in recent years, e.g. with increasing research focus on diagrammatic reasoning). But these abilities are not only necessary to navigate real (or virtual) spatial environments. They are also important components of a coherent, conceptually well-formed understanding of the world in which the student is embodied. Integrative training and assessment of both rigorous cognitive abilities generally most associated with both AI and "proper schooling" (such as linguistic and logical skills) along with kinematic and aesthetic/sensory abilities is essential to the development of an intelligence that can successfully both operate in and sensibly communicate about the real world in a roughly humanlike manner. Whether or not an AGI is targeted to interpret physical-world spatial data and perform tasks via robotics, in order to communicate ideas about a vast array of topics of interest to any intelligence in this world, an AGI must develop aspects of intelligence other than logical and linguistic cognition.

Issues with Virtual Preschool Engineering

One technical point that has come to our attention in exploring the AGI Preschool concept pertains to the available infrastructure for creating such a framework. Current standard methods of controlling avatar behaviors and avatar-object interactions in existing virtual environment engines are not sufficiently flexible to fulfill the requirements of an AGI Preschool. Such flexibility is feasible using current technology, but has not been implemented due to lack of motivation.

The most glaring deficit in current platforms is the lack of flexibility in terms of tool use. In most of these systems today, an avatar can pick up or utilize an object, or two objects can interact, only in specific, pre-programmed ways. For instance, an avatar might be able to pick up a virtual screwdriver only by the handle, rather than by pinching the blade between its fingers. This places severe limits on creative use of tools, which is absolutely critical in a preschool context. The solution to this problem is clear: adapt existing generalized physics engines to mediate avatar-object and object-object interactions. This would require more computation than current approaches, but not more than is feasible in a research context.

One way to achieve this goal would be to integrate a robot simulator with a virtual world or game engine, for instance to modify the OpenSim (opensimulator.org) virtual world to use the Gazebo (playerstage.sourceforge.net) robot simulator in place of its current physics engine. While tractable, such a project would require considerable software engineering effort.

Beyond Preschool

Once an AGI passes preschool, what are the next steps? There is still a long way to go, from preschool to an AGI system that is capable of, say, passing the Turing Test or serving as an effective artificial scientist.

Our suggestion is to extend the school metaphor further, and make use of existing curricula for higher levels of virtual education: grade school, secondary school, and all levels of post-secondary education. If an AGI can pass online primary and secondary schools such as e-tutor.com, and go on to earn an online degree from an accredited university, then clearly said AGI has successfully achieved “human level, roughly humanlike AGI.” This sort of testing is interesting not only because it allows assessment of stages intermediate between preschool and adult, but also because it tests humanlike intelligence without requiring precise imitation of human behavior.

If an AI can get a BA degree at an accredited university, via online coursework (assuming for simplicity courses where no voice interaction is needed), then we should consider that AI to have human-level intelligence. University coursework spans multiple disciplines, and the details of the homework assignments and exams are not known in advance, so like a human student the AGI team can’t cheat.

In addition to the core coursework, a schooling approach also tests basic social interaction and natural language communication, ability to do online research, and general problem solving ability. However, there is no rigid requirement to be strictly humanlike in order to pass university classes.

References

Ben-David, S., Schuller, R.: Exploiting task relatedness for learning multiple tasks. In: Proceedings of the 16th Annual Conference on Learning Theory. (2003)

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28, 41–75.

Chen, Jie-Qi and Gillian McNamee (2007). *Bridging: Assessment for Teaching and Learning in Early Childhood Classrooms.*, Corwin Press.

French, R. (1990) “Subcognition and the Limits of the Turing Test” *Mind* 99, 53-65.

Gardner, Howard. (1983) "Frames of Mind: The Theory of Multiple Intelligences." New York: Basic Books.

Goertzel, Ben, Cassio Pennachin, Nil Geissweiller, Moshe Looks, Andre Senna, Welter Silva, Ari Heljakka, Carlos Lopes (2008). *An Integrative Methodology for Teaching Embodied*

Non-Linguistic Agents, Applied to Virtual Animals in Second Life. Proceedings of the First Conference on Artificial General Intelligence, IOS Press.

Goertzel, Ben and Stephan Vladimir Bugaj (2006). *Stages of Cognitive Development in Uncertain-Logic-Based AI Systems*, in Proceedings of First Conference on Artificial General Intelligence, IOS Press

Goertzel, Ben, Ari Heljakka, Stephan Vladimir Bugaj, Cassio Pennachin, and Moshe Looks (2006). *Exploring Android Developmental Psychology in a Simulation World.* Presented at ICCS/CogSci 2006 Android Science Workshop.

Goertzel, Ben, Ken Silverman, Cate Harteley, Stephan Vladimir Bugaj and Mike Ross. *The Baby Webmind Project.* Presented at AISB 2000.

Hayes, P., and Ford, K. (1995)“Turing Test Considered Harmful” Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence Montreal, 972-7.

Laird, John, Robert Wray, Robert Marinier and Pat Langley (2009). *Claims and Challenges in Evaluating Human-Level Intelligent Systems.* Proceedings of AGI-09

Laud, A., & Dejong, G. (2003). The influence of reward on the speed of reinforcement learning: An analysis of shaping. Proceedings of the 20th international conference on machine learning.

Legg, Shane and Marcus Hutter (2007). *Universal Intelligence: A Definition of Machine Intelligence.* In *Minds and Machines*, pages 391-444, volume 17, number 4, November 2007.

Legg, Shane and Marcus Hutter (2006). *A Collection of Definitions of Intelligence* In B. Goertzel, editor, *Advances in Artificial General Intelligence*, IOS Press, 2006.

Li, L., Walsh, T., & Littman, M. (2006). Towards a unified theory of state abstraction for MDPs. Proceedings of the ninth international symposium on AI and mathematics.

Moor, J. (2001) “The Status and Future of the Turing Test” *Minds and Machines* 11, 77-93.

Montessori, Maria (1912). *The Montessori Method*, Frederick A. Stokes Company,

Neilsen, Dianne Miller (2006). *Teaching Young Children, Preschool-K: A Guide to Planning Your Curriculum, Teaching Through Learning Centers, and Just About Everything Else*, Corwin Press.

Reece, Charlotte Strange, Kamii, Constance (2001). *The measurement of volume: Why do young children measure inaccurately?*, *School Science and Mathematics*, Nov 2001,

Schunk, Dale (2007). *Theories of Learning: An Educational Perspective*, Prentice Hall.

Steiner, Rudolf and S.K. Sagarin. (2003) *What is Waldorf Education? Three Lectures*, Steiner Books.

Taylor, M., & Stone, P. (2007). Cross-domain transfer for reinforcement learning. Proceedings of the 24th International Conference on Machine Learning.

Thrun, S., & Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15, 25–46.

Turing, Alan (October 1950), "Computing Machinery and Intelligence", *Mind* LIX(236): 433–460,

Pointer Semantics with Forward Propagation

Sujata Ghosh*

Center for Soft Computing Research
Indian Statistical Institute
Kolkata, West Bengal, India

Benedikt Löwe†

Institute for Logic,
Language and Computation
Universiteit van Amsterdam
1018 TV Amsterdam, The Netherlands

Sanchit Saraf‡

Department of Mathematics and Statistics
Indian Institute of Technology
Kanpur 208016, India

Abstract

In this paper, we will discuss a new approach to formally modelling belief change in systems of sentences with inter-dependency. Our approach is based on the paradigm called *pointer semantics* or *revision theory* which forms a fundamental way of successfully understanding the semantics of logic programming, but has also been used extensively in philosophical logic and other applications of logic. With a purely unidirectional (backward) flow of change, pointer semantics are not fit to deal with belief change. We propose an extension that allows flow of change in both directions in order to be applied for belief change.

Introduction

Pointer semantics

Pointer semantics are a formal propositional language for finitely many propositions $\{p_0, \dots, p_n\}$ defined in terms of each other, and are a well-established tool in logic. The underlying idea of pointer semantics is to

“iterate away uncertainties and keep the values that stabilize”: (#)

Starting with an initial hypothesis for the truth values that may be in conflict with each other, you apply the Tarskian definition of truth repeatedly and consider the iteration sequence. Those values that stabilize will be kept, the others discarded. The semantics based on (#) have been rediscovered several times independently and have found applications in various areas of foundational study and philosophy.¹ The language of pointer semantics is closely related to logic programming and is the logical reflection of the “Revision

Theory of Truth” (GB93). The version of pointer semantics that we shall be using in this paper is essentially that of Haim Gaifman (Gai88; Gai92). The revision rules of pointer semantics let the truth value of a proposition depend on the values of those propositions it points to in the dependency graph (cf. Definition 1), so truth values “flow backward in the dependency graph”.

Pointer semantics for belief systems

It has been suggested several times (e.g., cf. (Löw06)) that the underlying principle (#) could be used for other processes involving revision, e.g., the process of revising and updating our beliefs in the face of learning new and unexpected facts. But the mentioned “backward flow” of traditional pointer semantics makes sure that truth values of terminal nodes in the dependency graph (which would typically correspond to those newly learned atomic statements) will never change in the revision process. In the context of belief, we would need *forward propagation* of truth values along the dependency graph. This idea has been implemented in a formal system in (GLS07), but the system proposed by the authors was not easy to handle.

Aims, motivation and related work

Our aim is to provide a new framework for a semantics of belief change based on the general principle (#) using the standard definition of revision semantics (Definition 2). In this paper, we cannot give a motivation of the general framework of revision theory and refer the reader to the extensive philosophical discussion in (GB93) or the survey (Löw06). Our aim is to stay as close as possible to the spirit of that framework.

There are many competing formal frameworks that deal with the question “How can we model rational belief change?”, most of which far more developed than what can be outlined in this short note. Transition systems and action description languages have been used in (HD05; HD07); there is a rich literature on using probabilistic logics for modelling belief change; in machine learning, dependency networks, Bayesian networks and Markov logic networks have been used (KDR07; HCH03; KD05); an argumentation theoretic approach can be found in (CnS07). This short list only scratches the surface of the vast literature of very good, intuitive and successful answers to our general

*Additional affiliation: Department of Mathematics, Visva-Bharati, Santiniketan, India.

†Additional affiliations: Department Mathematik, Universität Hamburg, Hamburg, Germany; Mathematisches Institut, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany.

‡The third author would like to thank the Institute for Logic, Language and Computation (ILLC) of the *Universiteit van Amsterdam* and the *Department Mathematik* of the *Universität Hamburg* for their hospitality during his visit from May to July 2008. All three authors would like to thank Bjarni Hilmarsson (Amsterdam) for programming support.

¹For more details, cf. the section ‘The ubiquity of revision’ in (Löw06, § 6).

question. Our approach of staying as close as possible to the pointer semantics paradigm of (#) cannot yet compete at the same level of depth and maturity at the moment. So, we need to explain why one should be interested in a new approach to belief change based on pointer semantics:

Traditional logic approaches to belief update are at the level of axiomatics of what we require of a belief change function, not at the more detailed level of how to actually make the decisions about the adequate belief change. For instance, if an agent believes $\{p \rightarrow q, p\}$ and learns $\{\neg q\}$, then the axiomatic prediction would be that either $p \rightarrow q$ or p has to be dropped, but without further details, it is difficult to predict which one.

As discussed in (L w06), pointer semantics carries a lot of information transcending the pure definition of the semantics (Definition 2): you can look at how fast stable values stabilize, at the various oscillation patterns of oscillating hypotheses, etc. This information can be used for definitions of *levels* or *types* of stability in order to help with prioritizing, promising to provide new insight in possible belief revision operators. These are tools that might be applicable directly in modelling belief change, or could serve as a subsidiary tool to support other (established) systems of formal belief change models for applications in artificial intelligence.

Overview of this paper

In our section “Definitions”, we give the standard Definition 2 from pointer semantics (following (L w06)) and define an algebra of pointer systems. The latter definition is new to this paper, allowing to state and prove Propositions 3 and 4 in the case of operators restricted to backward propagation. The central new definition of this paper is in the section “Belief Semantics with Forward Propagation”. In the section “Properties of our Belief Semantics” we test our system in an example originally used in (GLS07) and finally see that our system is ostensibly non-logical as expected for a system is intended to model systems of belief.² We close with a discussion of future work in our final section.

Definitions

Abstract Pointer Semantics

Fix a finite set of propositional variables $\{p_0, \dots, p_n\}$. An **expression** is just a propositional formula using \wedge , \vee , and \neg and some of the propositional variables or the empty sequence, denoted by $_$.

We fix a finite algebra of truth values \mathbb{T} with operations \wedge , \vee and \neg corresponding to the syntactic symbols. We assume a notion of order corresponding to information content that gives rise to a notion of **infimum** in the algebra of truth values, allowing to form $\inf(X)$ for some subset of $X \subseteq \mathbb{T}$. A truth value will represent the lowest information content (i.e., a least element in the given order); this truth value will be denoted by $\frac{1}{2}$. We allow \inf to be applied to the empty set and let $\inf \emptyset := \frac{1}{2}$.

²“The fact that the logic of belief, even rational belief, does not meet principles of truth-functional deductive logic, should no longer surprise us (Gol75, p. 6).”

Our salient example is the algebra $\mathbb{T} := \{0, \frac{1}{2}, 1\}$ with the following operations (“strong Kleene”):

\wedge	0	$\frac{1}{2}$	1	\vee	0	$\frac{1}{2}$	1	\neg	0	1
0	0	0	0	0	0	$\frac{1}{2}$	1	0	1	
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{1}{2}$	1	1	1	1	1	1	0	0

The value $\frac{1}{2}$ stands for ignorance, and thus the infimum is defined as $\inf(\{t\}) := t$, $\inf(\{\frac{1}{2}\} \cup X) := \frac{1}{2}$, $\inf(\{0, 1\}) := \frac{1}{2}$. This algebra of truth values will be used in this paper, even though the set-up in this section is more general.

If E is an expression and p_i is one of the propositional variables, then $p_i \leftarrow E$ is a **clause**. We intuitively interpret $p_i \leftarrow E$ as “ p_i states E ”. If E_0, \dots, E_n are expressions, a set of clauses $\Sigma := \{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$ is called a **pointer system**. An **interpretation** is a function $I: \{p_0, \dots, p_n\} \rightarrow \mathbb{T}$ assigning truth values to propositional letters. Note that if \mathbb{T} is finite, the set of interpretations is a finite set (we shall use this later). A given interpretation I can be naturally extended to a function assigning truth values to all expressions (using the operations \wedge , \vee and \neg on \mathbb{T}). We denote this extended function with the same symbol I .

A clause can be transformed into an equation in \mathbb{T} : if $p_i \leftarrow E$ is a clause, we can read it as an equation $p_i = E$ in \mathbb{T} . If Q is such an equation, we say that an interpretation I is a **solution** of Q if plugging the values $\{I(p_0), \dots, I(p_n)\}$ into the corresponding variables of the equation results in the same value left and right of the equals sign. An interpretation is a solution of a set of equations if it is a solution of each equation in the set.

A function mapping interpretations to interpretations is called a **revision function**; a family of these functions indexed by pointer systems is called a **revision operator**. If δ is a revision operator, we write δ_Σ for the revision function assigned to the pointer system Σ (and sometimes just write δ if Σ is clear from the context). We use the usual notation for iteration of revision functions, i.e., $\delta^0(I) := I$, $\delta^{n+1}(I) := \delta(\delta^n(I))$.

Definition 1 Given a pointer system $\{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$, we define its **dependency graph** by letting $\{0, \dots, n\}$ be the vertices and allowing an edge from i to j if p_j occurs in E_i .³

Given a proposition p_i , arrows point to i from the propositions occurring in E_i , and thus we call a revision operator δ an **B-operator** (for “backward”) if the value of $\delta(I)(p_i)$ only depends on the values of $I(p_j)$ for p_j occurring in E_i .

Fix Σ and δ . We call an interpretation $I(\Sigma, \delta)$ -**stable** if there is some k such that for all $n \geq k$, $\delta^n(I) = I$. We call $I(\Sigma, \delta)$ -**recurring** if for every k there is a $n \geq k$ such that $\delta^n(I) = I$. If Σ is fixed by the context, we drop it from the notation and call interpretations δ -**stable** and δ -**recurring**. If H is an interpretation, we consider the sequence $H^\infty := \{\delta^i(H); i \in \mathbb{N}\}$ of interpretations occurring in the infinite iteration of δ on H . Clearly, if there is a stable interpretation in H^∞ , then this is the only recurring interpretation in H^∞ . We write $\text{Rec}_{\Sigma, \delta}(H)$ for the set of recurring interpretations

³The relationship between pointer semantics and the dependency graph has been investigated in (Bol03).

in H^∞ . Note that since the set of interpretations is finite, this set must be non-empty. If $I \in \text{Rec}_{\Sigma, \delta}(H)$, then there are $i, j > 0$ such that $I = \delta^i(H) = \delta^{i+j}(H)$. Then for every $k < j$ and every n , we have $\delta^{i+k} = \delta^{i+n \cdot j+k}(H)$, so the sequence H^∞ exhibits a periodicity of length j (or a divisor of j). After the first occurrence of an $I \in \text{Rec}_{\Sigma, \delta}(H)$, all further elements of H^∞ are recurring as well, and in particular, there is a recurring J such that $\delta(J) = I$. We shall call this an **I -predecessor** and will use this fact in our proofs.

Definition 2

$$\begin{aligned} \llbracket \Sigma, p_i \rrbracket_{\delta, H} &:= \inf\{I(p_i) ; I \in \text{Rec}_{\Sigma, \delta}(H)\}, \text{ and} \\ \llbracket \Sigma, p_i \rrbracket_{\delta} &:= \inf\{I(p_i) ; \exists H (I \in \text{Rec}_{\Sigma, \delta}(H))\}. \end{aligned}$$

An algebra of pointer systems

In the language of abstract pointer systems, the possibility of complicated referential structures means that the individual proposition cannot be evaluated without its context.

As a consequence, the natural notion of logical operations is not that between propositions, but that between systems. If $\Sigma = \{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$ is a pointer system and $0 \leq i \leq n$, we define a pointer system that corresponds to the negation of p_i with one additional propositional variable p_{-} ,

$$\neg(\Sigma, p_i) := \Sigma \cup \{p_{-} \leftarrow \neg p_i\}.$$

If we have two pointer systems

$$\begin{aligned} \Sigma_0 &= \{p_0 \leftarrow E_{0,0}, \dots, p_{n_0} \leftarrow E_{0,n_0}\}, \text{ and} \\ \Sigma_1 &= \{p_0 \leftarrow E_{1,0}, \dots, p_{n_1} \leftarrow E_{1,n_1}\}, \end{aligned}$$

we make their sets of propositions disjoint by considering a set $\{p_0, \dots, p_{n_0}, p_0^*, \dots, p_{n_1}^*, p_*\}$ of $n_0 + n_1 + 2$ propositional variables. We then set

$$\Sigma_1^* := \{p_0^* \leftarrow E_{1,0}, \dots, p_{n_1}^* \leftarrow E_{1,n_1}\}.$$

With this, we can now define two new pointer systems (with an additional propositional variable p_*):

$$\begin{aligned} (\Sigma_0, p_i) \wedge (\Sigma_1, p_j) &:= \Sigma_0 \cup \Sigma_1^* \cup \{p_* \leftarrow p_i \wedge p_j\}, \\ (\Sigma_0, p_i) \vee (\Sigma_1, p_j) &:= \Sigma_0 \cup \Sigma_1^* \cup \{p_* \leftarrow p_i \vee p_j\}. \end{aligned}$$

Logical properties of Gaifman pointer semantics

Fix a system $\Sigma = \{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$. A proposition p_i is called a **terminal node** if $E_i = _;$ it is called a **source node** if p_i does not occur in any of the expressions E_0, \dots, E_n . This corresponds directly to the properties of i in the dependency graph: p_i is a terminal node if and only if i has no outgoing edges in the dependency graph, and it is a source node if and only if i has no incoming edges in the dependency graph. The **Gaifman-Tarski operator** δ_B is defined as follows:

$$\delta_B(I)(p_i) := \begin{cases} I(E_i) & \text{if } p_i \text{ is not terminal,} \\ I(p_i) & \text{if } p_i \text{ is terminal.} \end{cases}$$

Note that this operator can be described as follows:

“From the clause $p_i \leftarrow E_i$ form the equation Q_i by replacing the occurrences of p_i on the right-hand side of the equality sign with the values $I(p_i)$. If p_i is a terminal node, let $\delta(I)(p_i) := I(p_i)$. Otherwise, let I^* be the unique solution to the system of equations $\{Q_0, \dots, Q_n\}$ and let $\delta(I)(p_i) := I^*(p_i)$.” (*)

This more complicated description will provide the motivation for the forward propagation operator δ_F in the section “Belief semantics with forward propagation”.

The operator δ_B gives rise to a *logical* system, as the semantics defined by δ_B are compatible with the operations in the algebra of pointer systems.

Proposition 3 Let $\Sigma = \{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$ be a pointer system. For any $i \leq n$, we have

$$\llbracket \neg(\Sigma, p_i) \rrbracket_{\delta_B} = \neg \llbracket \Sigma, p_i \rrbracket_{\delta_B}.$$

Proof. In this proof, we shall denote interpretations for the set $\{p_0, \dots, p_n\}$ by capital letters I and J and interpretations for the bigger set $\{p_0, \dots, p_n, p_{-}\}$ by letters \hat{I} and \hat{J} . It is enough to show that if \hat{I} is δ_B -recurring, then there is some δ_B -recurring J such that $\hat{I}(p_{-}) = \neg J(p_i)$. If I is δ_B -recurring, we call J an **I -predecessor** if J is also δ_B -recurring and $\delta_B(J) = I$, and similarly for \hat{I} . It is easy to see that every δ_B -recurring I (or \hat{I}) has an I -predecessor (or \hat{I} -predecessor) which is not necessarily unique.

As δ_B is a **B**-operator, we have that if \hat{J} is δ_B -recurring, then so is $J := \hat{J} \upharpoonright \{p_0, \dots, p_n\}$.

Now let \hat{I} be δ_B -recurring and let \hat{J} be one of its \hat{I} -predecessors. Then by the above, $J := \hat{J} \upharpoonright \{p_0, \dots, p_n\}$ is δ_B -recurring and

$$\hat{I}(p_{-}) = \delta_B(\hat{J})(p_{-}) = \neg \hat{J}(p_i) = \neg J(p_i).$$

q.e.d.

Proposition 4 Let $\Sigma_0 = \{p_0 \leftarrow E_0, \dots, p_n \leftarrow E_n\}$ and $\Sigma_1 = \{p_0 \leftarrow F_0, \dots, p_m \leftarrow F_m\}$ be pointer systems. For any $i, j \leq n$, we have

$$\llbracket (\Sigma_0, p_i) \vee (\Sigma_1, p_j) \rrbracket_{\delta_B} = \llbracket \Sigma_0, p_i \rrbracket_{\delta_B} \vee \llbracket \Sigma_1, p_j \rrbracket_{\delta_B}.$$

Similarly for \vee replaced by \wedge .

Proof. The basic idea is very similar to the proof of Proposition 3, except that we have to be a bit more careful to see how the two systems Σ_0 and Σ_1 can interact in the bigger system. We reserve letters I_0 and J_0 for the interpretations on Σ_0 , I_1 and J_1 for those on Σ_1 and I and J for interpretations on the whole system, including p_* . If $\llbracket \Sigma_0, p_i \rrbracket = 1$, then any δ_B -recurring I must have $I(p_*) = 1$ by the \vee -analogue of the argument given in the proof of Proposition 3. Similarly, for $\llbracket \Sigma_1, p_j \rrbracket = 1$ and the case that $\llbracket \Sigma_0, p_i \rrbracket = \llbracket \Sigma_1, p_j \rrbracket = 0$. This takes care of six of the nine possible cases.

If I_0 and I_1 are δ_B -recurring, then so is the function $I := \delta(I_0) \cup \delta(I_1) \cup \{\langle p_*, I_0(p_i) \vee I_1(p_j) \rangle\}$ (if I_0 is k -periodic and I_1 is ℓ -periodic, then I is at most $k \cdot \ell$ -periodic). In particular, if we have such an I_0 with $I_0(p_i) = \frac{1}{2}$ and an I_1 with $I_1(p_j) \neq 1$, then $I(p_*) = \frac{1}{2}$ (and symmetrically for interchanged rôles). Similarly, if we have recurring interpretations for relevant values 0 and 1 for both small systems, we can put them together to δ_B -recurring interpretations with values 0 and 1 for the big system. This gives the truth value $\frac{1}{2}$ for the disjunction in the remaining three cases.

q.e.d.

H_0	$\delta_B(H_0)$	$\delta_F(H_0)$	H_1	$\delta_B(H_1)$	$\delta_F(H_1)$	H_2	$\delta_B(H_2)$	$\delta_F(H_2)$	H_3	$\delta_B(H_3)$	$\delta_F(H_3)$	H_4
0	$\frac{1}{2}$	0	0	1	0	$\frac{1}{2}$	1	$\frac{1}{2}$	1	1	1	1
1	1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	1	1	1	1	1	1	1

Figure 1: The first three iterations of values of $H_0 = (0, 1, 0, \frac{1}{2}, \frac{1}{2})$ up to the point of stability ($H_3 = (1, 1, 0, 0, 1)$).

Belief semantics with forward propagation

In (GLS07), the authors gave a revision operator that incorporated both backward and forward propagation. The value of $\delta(I)(p_i)$ depended on the values of all $I(p_j)$ such that j is connected to i in the dependency graph. Here, we split the operator in two parts: the backward part which is identical to the Gaifman-Tarski operator, and the forward part which we shall define now.

In analogy to the definition of δ_B , we define δ_F as follows. Given an interpretation I , we transform each clause $p_i \leftarrow E_i$ of the system into an equation $Q_i \equiv I(p_i) = E_i$ where the occurrences of the p_i on the left-hand side of the equation are replaced by their I -values and the ones on the right-hand side are variables. We obtain a system $\{Q_0, \dots, Q_n\}$ of $n + 1$ equations in \mathbb{T} . Note that we cannot mimic the definition of δ_B directly: as opposed to the equations in that definition, the system $\{Q_0, \dots, Q_n\}$ need not have a solution, and if it has one, it need not be unique. We therefore define: if p_i is a source node, then $\delta_F(I)(p_i) := I(p_i)$. Otherwise, let S be the set of solutions to the system of equations $\{Q_0, \dots, Q_n\}$ and let $\delta_F(I)(p_i) := \inf\{I(p_i); I \in S\}$ (remember that $\inf \emptyset = \frac{1}{2}$). Note that this definition is literally the dual to definition (*) of δ_B (i.e., it is obtained from (*) by interchanging “right-hand side” by “left-hand side” and “terminal node” by “source node”).

We now combine δ_B and δ_F to one operator δ_T by defining pointwise

$$\delta_T(I)(p_i) := \delta_F(I)(p_i) \otimes \delta_B(I)(p_i)$$

where \otimes has the following truth table:⁴

\otimes	0	$\frac{1}{2}$	1
0	0	0	$\frac{1}{2}$
$\frac{1}{2}$	0	$\frac{1}{2}$	1
1	$\frac{1}{2}$	1	1

⁴The values for agreement ($0 \otimes 0$, $\frac{1}{2} \otimes \frac{1}{2}$, and $1 \otimes 1$) are obvious choices. In case of complete disagreement ($0 \otimes 1$ and $1 \otimes 0$), you have little choice but give the value $\frac{1}{2}$ of ignorance (otherwise, there would be a primacy of one direction over the other). For reasons of symmetry, this leaves two values $\frac{1}{2} \otimes 0 = 0 \otimes \frac{1}{2}$ and $\frac{1}{2} \otimes 1 = 1 \otimes \frac{1}{2}$ to be decided. We opted here for the most informative truth table that gives classical values the benefit of the doubt. The other options would be the tables

\otimes_0	0	$\frac{1}{2}$	1	\otimes_1	0	$\frac{1}{2}$	1	\otimes_2	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$f\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
1	$\frac{1}{2}$	1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1

Each of these connectives will give rise to a slightly different semantics. We opted for the first connective \otimes , as the semantics based on the other three seem to have a tendency to stabilize on the value $\frac{1}{2}$ very often (the safe option: in case of confusion, opt for ignorance).

Properties of our belief semantics

As mentioned in the introduction, we should not be shocked to hear that a system modelling belief and belief change does not follow basic logical rules such as Propositions 3 and 4. Let us take the particular example of conjunction: the fact that belief is not closed under the standard logical rules for conjunction is known as the *preface paradox* and has been described by Kyburg as “conjunctivitis” (Kyb70). In other contexts (that of the modality of “ensuring that”), we have a problem with simple binary conjunctions (Sch08). Of course, the failure of certain logical rules in reasoning about belief is closely connected to the so-called “errors in reasoning” observed in experimental psychology, e.g., the famous Wason selection task (Was68). What constitutes rational belief in this context is an interesting question for modellers and philosophers alike (Ste97; Chr07; Cou08). Let us focus on some concrete examples to validate our claim that the semantics we propose do agree with intuitive understanding, and thus serve as a quasi-empirical test for our system as a formalization of reasoning in self-referential situations with evidence.

Concrete examples

So far, we have just given an abstract system of belief flow in our pointer systems. In order to check whether our system results in intuitively plausible results, we have to check a few examples. Keep in mind that our goal should be to model human reasoning behaviour in the presence of partially paradoxical situations. In this paper, we can only give a first attempt at testing the adequacy of our system: an empirical test against natural language intuitions on a much larger scale is needed. For this, also cf. our section “Discussion and Future Work”.

The Liar As usual, the liar sentence is interpreted by the system $\Sigma := \{p_0 \leftarrow \neg p_0\}$. Since we have only one propositional variable, interpretations are just elements of $\mathbb{T} = \{0, \frac{1}{2}, 1\}$. It is easy to see that $\delta_B(0) = \delta_F(0) = \delta_T(0) = 1$, $\delta_B(\frac{1}{2}) = \delta_F(\frac{1}{2}) = \delta_T(\frac{1}{2}) = \frac{1}{2}$, and $\delta_B(1) = \delta_F(1) = \delta_T(1) = 0$. This means that the δ_T -behaviour of the liar sentence is equal to the Gaifman-semantics behaviour.

The Miller-Jones Example Consider the following test example from (GLS07):

Professors Jones, Miller and Smith are colleagues in a computer science department. Jones and Miller dislike each other without reservation and are very liberal in telling everyone else that “everything that the other one says is false”. Smith just returned from a trip abroad and needs to find out about two committee meetings on Monday morning. He sends out e-mails to his colleagues and to the department secretary. He asks all three of them about the meeting of the faculty, and

H_0^*	$\delta_B(H_0^*)$	$\delta_F(H_0^*)$	H_1^*	$\delta_B(H_1^*)$	$\delta_F(H_1^*)$	H_2^*	$\delta_B(H_2^*)$	$\delta_F(H_2^*)$	H_3^*	$\delta_B(H_3^*)$	$\delta_F(H_3^*)$	H_4^*	$\delta_B(H_4^*)$	$\delta_F(H_4^*)$	H_5^*
0	1/2	0	0	1	0	1/2	1	1/2	1	1	1	1	1	1	1
1	1	1/2	1	1	1/2	1	1	1/2	1	1	1/2	1	1	1/2	1
0	0	1/2	0	0	1/2	0	0	1/2	0	0	1/2	0	0	1/2	0
1/2	0	1/2	0	0	1/2	0	0	1/2	0	0	0	0	0	0	0
0	1/2	1/2	1/2	1/2	1/2	1/2	1	1/2	1	1	1/2	1	1	1	1
1	1	1/2	1	1	1/2	1	1	1/2	1	1	1/2	1	1	1/2	1
0	0	1/2	0	0	1/2	0	0	1/2	0	0	1/2	0	0	1/2	0
1/2	0	1	1/2	0	1/2	0	0	1/2	0	0	0	0	0	0	0
1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1	1/2	1	1	1	1

Figure 2: The first three iterations of values of $H_0^* = (0, 1, 0, 1/2, 0, 1, 0, 1/2, 1/2)$ up to the point of stability ($H_4^* = (1, 1, 0, 0, 1, 1, 0, 0, 1)$).

Jones and the secretary about the meeting of the library committee (of which Miller is not a member).

Jones replies: “We have the faculty meeting at 10am and the library committee meeting at 11am; by the way, don’t believe anything that Miller says, as he is always wrong.”

Miller replies: “The faculty meeting was cancelled; by the way don’t believe anything that Jones says, as he is always wrong.”

The secretary replies: “The faculty meeting is at 10 am and the library committee meeting is at 11 am. But I am sure that Professor Miller told you already as he is always such an accurate person and quick in answering e-mails: everything Miller says is correct.” (GLS07, p. 408)

Trying to analyse Smith’s reasoning process after he returns from his trip, we can assume that he generally believes the secretary’s opinions, and that he has no prior idea about the truth value of the statements “the faculty meeting is at 10am” and “the library meeting is at 11am” and the utterances of Miller and Jones. We have a vague intuition that tells us that in this hypothetical situation, Smith should at least come to the conclusion that the library meeting will be held at 11am (as there is positive, but no negative evidence). His beliefs about the faculty meeting are less straightforward, as there is some positive evidence, but also some negative evidence, and there is the confusing fact that the secretary supports Miller’s statement despite the disagreement in truth value.

In (GLS07, p. 409), the authors analysed this example with their real-valued model and ended up with a stable solution in which Smith accepted both appointments and took Jones’s side (disbelieving Miller). In our system, we now get the following analysis: A pointer system formulation is given as follows.

$$\begin{aligned}
p_0 &\leftarrow \neg p_1 \wedge \neg p_4, & p_1 &\leftarrow \neg p_0 \wedge p_2 \wedge p_4, \\
p_2 &\leftarrow \neg, & p_3 &\leftarrow p_0 \wedge p_2 \wedge p_4, \\
p_4 &\leftarrow \neg,
\end{aligned}$$

where p_0 is Miller’s utterance, p_1 is Jones’s utterance, p_2 is “the library meeting will take place at 11am”, p_3 is the secretary’s utterance, and p_4 is the “the faculty meeting will take place at 10am”.

We identify our starting hypothesis with $H := (1/2, 1/2, 1/2, 1, 1/2)$ (here, as usual, we identify an interpretation with its sequence of values in the order of the indices of the propositional letters). Then $\delta_B(H) = (1/2, 1/2, 1/2, 1/2, 1/2)$ and $\delta_F(H) = (1/2, 1/2, 1, 1, 1/2)$, so that we get $H' := \delta_T(H) = (1/2, 1/2, 1, 1, 1/2)$. Then, in the second iteration step, $\delta_B(H') =$

$(1/2, 1/2, 1, 1/2, 1/2)$ and $\delta_F(H') = (1/2, 1/2, 1, 1, 1/2)$, so we obtain stability at $\delta_T(H') = H'$.

Examples of nonlogical behaviour

In what follows, we investigate some logical properties of the belief semantics, viz. negation and disjunction, focussing on stable hypotheses. To some extent, our results show that the operator δ_T is rather far from the logical properties of δ_B discussed in Propositions 3 and 4.

Negation Consider the pointer system Σ given by

$$\begin{aligned}
p_0 &\leftarrow \neg p_3, & p_1 &\leftarrow \neg, \\
p_2 &\leftarrow \neg, & p_3 &\leftarrow p_1 \wedge p_2.
\end{aligned}$$

The interpretation $H := (0, 1, 0, 1/2)$ is δ_T -stable, as $\delta_B(H) = (1/2, 1, 0, 0)$, $\delta_F(H) = (0, 1/2, 1/2, 1)$, and thus $\delta_T(H) = H$.

Now let us consider the system $\neg(\Sigma, p_3)$. Remember from the proof of Proposition 3 that stable interpretations for the small system could be extended to stable interpretations for the big system by plugging in the expected value for p_- . So, in this particular case, the stable value for p_3 is $1/2$, so we would expect that by extending H by $H_0(p_-) := 1/2$, we would get another stable interpretation.

But this is not the case, as the table of iterated values given in Figure 1 shows. Note that H_0 is not even recurring.

Disjunction Consider the pointer systems Σ and Σ^* and their disjunction $(\Sigma, p_4) \vee (\Sigma^*, p_1^*)$ given as follows:

$$\begin{aligned}
p_0 &\leftarrow \neg p_3, & p_0^* &\leftarrow \neg p_3^*, \\
p_1 &\leftarrow \neg, & p_1^* &\leftarrow \neg, \\
p_2 &\leftarrow \neg, & p_2^* &\leftarrow \neg, \\
p_3 &\leftarrow p_1 \wedge p_2, & p_3^* &\leftarrow p_1^* \wedge p_2^*, \\
p_* &\leftarrow p_4 \vee p_1^*.
\end{aligned}$$

Note that Σ and Σ^* are the same system up to isomorphism and that Σ is the system from the previous example. We already know that the interpretation $H = (0, 1, 0, 1/2)$ is δ_T -stable (therefore, it is δ_T -stable for both Σ and Σ^* in the appropriate reading).

The natural extension of H to the full system with nine propositional variables would be $H_0^* := (0, 1, 0, 1/2, 0, 1, 0, 1/2, 1/2)$, as p_* should take the value $H(p_4) \vee H(p_1^*) = 1/2 \vee 0 = 1/2$. However, we see in Figure 2 that this interpretation is not stable (or even recurring).

Discussion and future work

Testing the behaviour of our system on the liar sentence and one additional example cannot be enough as an empirical test of the adequacy of our system. After testing more examples and having developed some theoretical insight into the system and its properties, we would consider testing the system experimentally by designing situations in which people reason about beliefs in self-referential situations with evidence, and then compare the predictions of our system to the actual behaviour of human agents.

Such an experimental test should not be done with just one system, but with a class of systems. We have already discussed that our choice of the connective \otimes combining δ_B and δ_F to δ_T was not unique. Similarly, the rules for how to handle multiple solutions (“take the pointwise infimum”) in the case of forward propagation are not the only way to deal with this formally. One natural alternative option would be to split the sequence H^∞ into multiple sequences if there are multiple solutions. For instance, if we are trying to calculate $\delta_F(H)$ and we have multiple solutions to the set of equations, then $\delta_F(H)$ becomes a set of interpretations (possibly giving rise to different recurrences and stabilities, depending on which possibility you follow). There are many variants that could be defined, but the final arbiter for whether these systems are adequate descriptions of reasoning processes will have to be the experimental test.

References

- Thomas Bolander. *Logical Theories for Agent Introspection*. PhD thesis, Technical University of Denmark, 2003.
- David Christensen. *Putting Logic in its place. Formal Constraints on Rational Belief*. Oxford University Press, 2007.
- Carlos Iván Chesñevar and Guillermo Ricardo Simari. A lattice-based approach to computing warranted beliefs in skeptical argumentation frameworks. 2007. In (Vel07, pp. 280–285).
- Marian E. Counihan. *Looking for logic in all the wrong places: an investigation of language, literacy and logic in reasoning*. PhD thesis, Universiteit van Amsterdam, 2008. ILLC Publications DS-2008-10.
- Haim Gaifman. Operational pointer semantics: Solution to self-referential puzzles I. In Moshe Y. Vardi, editor, *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, CA, March 1988*, pages 43–59. Morgan Kaufmann, 1988.
- Haim Gaifman. Pointers to truth. *Journal of Philosophy*, 89(5):223–261, 1992.
- Anil Gupta and Nuel Belnap. *The revision theory of truth*. MIT Press, 1993.
- Sujata Ghosh, Benedikt Löwe, and Erik Scorelle. Belief flow in assertion networks. In Uta Priss, Simon Polovina, and Richard Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications, 15th International Conference on Conceptual Structures, ICCS 2007, Sheffield, UK, July 22-27, 2007, Proceedings*, volume 4604 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2007.
- Alan H. Goldman. A note on the conjunctivity of knowledge. *Analysis*, 36:5–9, 1975.
- Geoff Hulten, David Maxwell Chickering, and David Heckerman. Learning Bayesian networks from dependency networks: A preliminary study. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2003.
- Aaron Hunter and James P. Delgrande. Iterated belief change: A transition system approach. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 460–465. Professional Book Center, 2005.
- Aaron Hunter and James P. Delgrande. An action description language for iterated belief change. 2007. In (Vel07, pp. 2498–2503).
- Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Machine Learning Conference*, pages 441–448. ACM Press, 2005.
- Kristian Kerstin and Luc De Raedt. Bayesian logic programming: Theory and tool. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.
- Henry Kyburg. Conjunctivitis. In Marshall Swain, editor, *Induction, Acceptance, and Rational Belief*, pages 55–82. Reidel, 1970.
- Benedikt Löwe. Revision forever! In Henrik Schärfe, Pascal Hitzler, and Peter Øhrstrøm, editors, *Conceptual Structures: Inspiration and Application, 14th International Conference on Conceptual Structures, ICCS 2006, Aalborg, Denmark, July 16-21, 2006, Proceedings*, volume 4068 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2006.
- Benjamin Schnieder. On what we can ensure. *Synthese*, 162:101–115, 2008.
- Edward Stein. *Without good reason. The rationality debate in philosophy and cognitive science*. Clarendon Library of Logic and Philosophy. Clarendon Press, 1997.
- Manuela M. Veloso, editor. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. AAAI Press, 2007.
- Peter Wason. Reasoning about a rule. *Quarterly Journal of Experimental Psychology*, 20(3):273–281, 1968.

The Role of Logic in AGI Systems: Towards a Lingua Franca for General Intelligence

Helmar Gust and Ulf Krumnack and Angela Schwering and Kai-Uwe Kühnberger

Institute of Cognitive Science, University of Osnabrück, Germany

{hgust|krumnack|aschweri|kkuehnbe}@uos.de

Abstract

Systems for general intelligence require a significant potential to model a variety of different cognitive abilities. It is often claimed that logic-based systems – although rather successful for modeling specialized tasks – lack the ability to be useful as a universal modeling framework due to the fact that particular logics can often be used only for special purposes (and do not cover the whole breadth of reasoning abilities) and show significant weaknesses for tasks like learning, pattern matching, or controlling behavior. This paper argues against this thesis by exemplifying that logic-based frameworks can be used to integrate different reasoning types and can function as a coding scheme for the integration of symbolic and subsymbolic approaches. In particular, AGI systems can be based on logic frameworks.

Introduction

As a matter of fact artificial intelligence currently departs significantly from its origins. The first decades of AI can be characterized as the attempt to use mostly logic-based methods for the development of frameworks and implementations of higher cognitive abilities. A few prominent examples are knowledge representation formalisms like Minsky's frames (Minsky, 1975), semantic networks (Sowa, 1987), McCarthy's situation calculus (McCarthy, 1963), Brachnan's KL-ONE (Brachman and Schmolze, 1985). With the rise and partial success of neurosciences, neuroinformatics, dynamic system theory, and other nature-inspired computing paradigms, logic-based frameworks seem to lose their importance in artificial intelligence. The current success of these new methodologies are at least partially based on the fact that several non-trivial problems are connected with logic-based approaches. Examples of such problems are the profusion of knowledge, the variety of reasoning formalisms, the problem of noisy data, and the lack of cognitive plausibility with respect to learning from sparse data, or reasoning in non-tractable domains.

A number of new research paradigms were proposed in order to overcome some limitations of logical computational frameworks. Many of them are inspired by biological or psychological findings. The following list summarizes some of these new methodologies:

- *Natural Computation*: This cluster contains methods like learning with neural networks and support vector ma-

chines, evolutionary computing, genetic algorithms, dynamic system theory etc.

- *Cognitive Robotics*: Embedded and embodied models of agents focus on perception aspects, motor control, and the idea that the world itself is the best of all possible models. This tradition moves the interest from higher to lower cognitive abilities.
- *Cognitive Architectures*: Integrated frameworks for modeling cognitive abilities, often use many different methodologies in order to model cognitive abilities. There seems to be a tendency for a method pluralism including also ideas from natural computation.
- *Further Approaches*: Further approaches for AI systems are, for example, fuzzy and probabilistic approaches (sometimes combined with logic), semantic networks that are enriched with activation potentials, the modeling of social aspects in multi-agent networks etc.

The mentioned methodologies are usually considered as alternatives to specialized logic-based AI systems. Although methods of, for example, natural computation may have strengths in particular domains, it is clearly far from being obvious that they can be used for modeling higher cognitive abilities. In particular, for AGI systems, which attempt to model the whole spectrum of higher and lower cognitive abilities, the difficulty to find a basis for its underlying methodology seems to be crucial: due to the fact that there are no good ideas how logical frameworks can be translated into such new computational paradigms, a starting point for AGI is the clarification of its methodological basis.

This paper argues for the usage of a non-standard logic-based framework in order to model different types of reasoning and learning in a uniform framework as well as the integration of symbolic and subsymbolic approaches. Analogical reasoning has the potential for an integration of a variety of reasoning formalisms and neural-symbolic integration is a promising research endeavor to allow the learning of logical knowledge with neural networks. The remainder of the paper has the following structure: first, we will discuss the variety of logic and learning formalisms. Second, we will show possible steps towards a logic-based theory that comprises different types of reasoning and learning. Third, we will argue for using logic as the lingua franca for AGI systems based on neural-symbolic integration.

Types of Reasoning	Corresponding Formalisms
Deductions	Classical Logic
Inductions	Inductive Logic Programming
Abductions	Extensions of Logic Programming
Analogical Reasoning	SME, LISA, AMBR, HDTP
Similarity-Based Reasoning	Case-Based Reasoning
Non-Monotonic Reasoning	Answer Set Programming
Frequency-Based Reasoning	Bayesian Reasoning
Vague and Uncertain Reasoning	Fuzzy, Probabilistic Logic
Reasoning in Ontologies	Description Logics
Etc.	Etc.

Table 1: Some types of reasoning and some formalisms

Tensions in Using Logic in AGI Systems

The Variety of Reasoning Types

Logical approaches have been successfully applied to applications like planning, theorem proving, knowledge representation, problem solving, and inductive learning, just to mention some of them. Nevertheless, it is remarkable that for particular domains and applications very special logical theories were proposed. These theories cannot be simply integrated into one uniform framework. Table 1 gives an overview of some important types of reasoning and their corresponding logic formalisms. Although such formalisms can be applied to a variety of different domains, it turns out that the degree of their generalization potential is limited. One will hardly find an approach that integrates inductive, deductive, abductive, fuzzy etc. logic theories in one unified framework.¹

The Variety of Learning Types

A very similar situation can be found by shifting our attention from reasoning issues to learning aspects. The manifold of learning paradigms used in applications leads to the development of various corresponding formalisms. Major distinctions of learning approaches like supervised, unsupervised, and reinforcement learning structure the field, but are themselves just labels for a whole bunch of learning algorithms. Table 2 summarizes some important learning techniques commonly used in AI.

Although the list of machine learning algorithms guarantees that efficient learning strategies do exist for many applications, several non-trivial problems are connected with these theories. The following list summarizes some of them:

- Learning from noisy data is a classical problem for symbolic learning theories. Neural-inspired machine learning mechanisms have certain advantages in this respect.
- There is no learning paradigm that convincingly can learn from sparse, but highly conceptualized data. Whereas natural agents are able to generate inductive hypotheses based on a few data points due to the availability of

¹Clearly, there are exceptions: an example may be Wang’s NARS architecture (Wang, 2006) where the integration of the whole range of higher cognitive abilities are programmatically modeled in a logic-based framework.

	Learning Types	Learning Approaches
Unsupervised Learning	Clustering	Neural Networks, SOMs, ART, RBF network
Supervised Learning	Classification, Learning a Function	Case-based reasoning, k -Nearest Neighbor, Decision Tree Learning
Reinforcement Learning	Policy Learning	Q -Learning, POMDPs, Temporal Difference Learning
Analytical & Inductive Learning	Rule Extraction, Learning in Domain Theories	Inductive Learning, Explanation-Based Learning, KBANNS

Table 2: Some types of learning and some methodological learning approaches

background knowledge, classical machine learning mechanisms usually need rather large numbers of (more or less) unconceptualized examples.

- Learning in artificial systems is quite often realized explicitly in a separate module especially designed for a particular task. Contrary to this idea, cognitive learning is usually a continuous adaptation process taking place on many levels: besides explicit learning, rearranging representations in order to align structural properties of two inputs in a complex reasoning process, resolving clashes in a conceptualization by rewriting the representation, implicitly erasing features of an entity to make it compatible with background knowledge etc. may play an important role for adaptation processes (Kühnberger et al., 2008).
- The gap between symbolic and subsymbolic representations is a problem for AI systems because of complementary strengths and weaknesses. A framework that can extract the strengths of both fields would be desirable.

Steps Towards a Logic-Based Theory for Reasoning and Learning

Sketch of HDTP

A framework that proposes to integrate different reasoning types in one framework is heuristic-driven theory projection (HDTP) described in (Gust, Kühnberger, and Schmid, 2006). HDTP has been proposed for analogical reasoning. Due to the fact that establishing an analogical relation between a given source and target domain often requires the combination of different reasoning types, analogical reasoning is a natural starting point for an integration methodology for AGI systems.

We sketch the basic ideas of HDTP in this subsection.² HDTP established an analogical relation between a source theory Th_S and target theory Th_T (both described in a first-order language \mathcal{L}) by computing a generalization (structural description) Th_G of the given theories. This generalization process is based on the theory of anti-unification (Plotkin,

²For a comprehensive presentation of the theory, the reader is referred to (Gust, Kühnberger, and Schmid, 2006) and (Schwering et al., 2009).

Solar System	Rutherford Atom
$\alpha_1 : \text{mass}(\text{sun}) > \text{mass}(\text{planet})$	$\beta_1 : \text{mass}(\text{nucleus}) > \text{mass}(\text{electron})$
$\alpha_2 : \forall t : \text{distance}(\text{sun}, \text{planet}, t) > 0$	$\beta_2 : \forall t : \text{distance}(\text{nucleus}, \text{electron}, t) > 0$
$\alpha_3 : \text{gravity}(\text{sun}, \text{planet}) > 0$	$\beta_3 : \text{coulomb}(\text{nucleus}, \text{electron}) > 0$
$\alpha_4 : \forall x \forall y : \text{gravity}(x, y) > 0$ $\rightarrow \text{attracts}(x, y)$	$\beta_4 : \forall x \forall y : \text{coulomb}(x, y) > 0$ $\rightarrow \text{attracts}(x, y)$
$\alpha_5 : \forall x \forall y \forall t : \text{attracts}(x, y) \wedge$ $\text{distance}(x, y, t) > 0 \wedge$ $\text{mass}(x) > \text{mass}(y)$ $\rightarrow \text{revolves_around}(y, x)$	
Generalized Theory	
$\gamma_1 : \text{mass}(A) > \text{mass}(B)$	
$\gamma_2 : \forall t : \text{distance}(A, B, t) > 0$	
$\gamma_3 : F(A, B) > 0$	
$\gamma_4 : \forall x \forall y : F(x, y) > 0 \rightarrow \text{attracts}(x, y)$	

Table 3: A formalization of the Rutherford analogy with obvious analogous structures (Krumnack et al., 2008)

1970). Anti-unification is the dual constructions of unification: if input terms t_1 and t_2 are given, the output is a generalized term t such that for substitutions Θ_1 and Θ_2 it holds: $t_1 = t\Theta_1$ and $t_2 = t\Theta_2$. It is well-known that for first-order anti-unification a generalization always exists, there are at most finitely many generalizations, and there exists a unique least generalization (Plotkin, 1970).

In order to apply the idea of anti-unification to analogies, it is necessary to extend the anti-unification framework in several respects. We will use the Rutherford analogy formalized in Table 3 as a simple example for motivating HDTP:

1. Not only terms but also formulas (theories) need to be anti-unified.
2. Whereas the generalization of α_1 and β_1 to γ_1 uses only first-order variables, the anti-unification of α_3 and β_3 requires the introduction of second-order variables (Table 3).
3. In order to productively generate the conclusion that electrons revolve around the nucleus, it is necessary to project α_5 to the source domain and generate a formula γ_5 in Th_G with

$$\exists A \exists B \forall t : \text{attracts}(A, B) \wedge \text{distance}(A, B, t) > 0 \wedge \text{mass}(A) > \text{mass}(B) \rightarrow \text{revolves_around}(B, A)$$

In the following, we mention some properties of the extensions mentioned in the above list. The generalization of term anti-unification to the anti-unification of formulas is rather straightforward and will be omitted here.³ A crucial point is the fact that not only first-order but also second-order generalizations need to be computed, corresponding to the introduction of second-order variables. If two terms $t_1 = f(a, b)$ and $t_2 = g(a, b)$ are given, a natural second-order generalization would be $F(a, b)$ (where F is a function variable) with substitutions $\Theta_1 = \{F \leftarrow f\}$ and $\Theta_2 = \{F \leftarrow g\}$. Then: $t_1 = f(a, b) = F(a, b)\Theta_1$ and $t_2 = g(a, b) = F(a, b)\Theta_2$. It is known that unrestricted second-order anti-unifications can lead to infinitely many anti-instances (Hasker, 1995). In (Krumnack et al.,

³Cf. (Krumnack et al., 2008) for more information.

2007), it is shown that a restricted form of higher-order anti-unification resolves this computability problem and is appropriate for analogy making.

Definition 1 *Restricted higher-order anti-unification is based on the following set of basic substitutions (\mathcal{V}_n denotes an infinite set of variables of arity $n \in \mathbb{N}$):*

- A renaming $\rho^{F, F'}$ replaces a variable $F \in \mathcal{V}_n$ by another variable $F' \in \mathcal{V}_n$ of the same argument structure:

$$F(t_1, \dots, t_n) \xrightarrow{\rho^{F, F'}} F'(t_1, \dots, t_n).$$

- A fixation ϕ_C^V replaces a variable $F \in \mathcal{V}_n$ by a function symbol $f \in \mathcal{C}_n$ of the same argument structure:

$$F(t_1, \dots, t_n) \xrightarrow{\phi_C^V} f(t_1, \dots, t_n).$$

- An argument insertion $\iota_{V, i}^{F, F'}$ with $0 \leq i \leq n$, $F \in \mathcal{V}_n$, $G \in \mathcal{V}_k$ with $k \leq n - i$, and $F' \in \mathcal{V}_{n-k+1}$ is defined by

$$F(t_1, \dots, t_n) \xrightarrow{\iota_{V, i}^{F, F'}} F'(t_1, \dots, t_{i-1}, G(t_i, \dots, t_{i+k-1}), t_{i+k}, \dots, t_n).$$

- A permutation $\pi_\alpha^{F, F'}$ with $F, F' \in \mathcal{V}_n$ and bijective $\alpha : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ rearranges the arguments of a term:

$$F(t_1, \dots, t_n) \xrightarrow{\pi_\alpha^{F, F'}} F'(t_{\alpha(1)}, \dots, t_{\alpha(n)}).$$

As basic substitutions one can rename a second-variable, instantiate a variable, insert an argument, or permute arguments. Finite compositions of these basic substitutions are allowed. It can be shown that every first-order substitution can be coded with restricted higher-order substitution and that there are at most finitely many anti-instances in the higher-order case (Krumnack et al., 2007).

A last point concerns the transfer of knowledge from the source to the target domain. In the Rutherford example, this is the projection of α_5 to the target domain governed by the analogical relation computed so far. Such projections allow to creatively introduce new concepts on the target side. The importance of these transfers for modeling creativity and productivity cannot be overestimated.

Integrating Different Types of Reasoning and Learning with HDTP

As should be already clear from the rough introduction to HDTP, analogy making involves several types of reasoning. Not only that the result of the process is the establishment of an analogical relation between Th_S and Th_T , but as a side-effect a generalization is computed that can be interpreted as a learning result: the generalized theory Th_G together with appropriate substitutions cover not only the input theories, but potentially also some further theories. With respect to the Rutherford analogy the result is a central body system. Therefore, already at the level of the core theory, a form of inductive inference is computed.⁴

Another type of inference is involved in cases where the domain theories Th_S and Th_T are not in a form such that an association of corresponding terms and formulas is possible, although the theories can be analogically associated with each other. Assume that β_2 in Table 3 is replaced by

$$\beta'_2 : \forall t : distance(electron, nucleus, t) > 0.$$

A simple association of α_2 and β'_2 is no longer possible, because the argument positions do not coincide. In order to resolve this problem, HDTP uses a theorem prover to rewrite the axioms and deduce a representation that is more appropriate for analogy making. In our example, such a piece of background knowledge may be the formula

$$\forall x \forall y \forall t : distance(x, y, t) = distance(y, x, t)$$

Because the system is based on a first-order logical input, and for every first-order theory there are infinitely many possible axiomatizations, such cases may occur often. In (Krumnack et al., 2008), an algorithm is presented that shows how such a re-representation of a theory can be implemented into the HDTP core framework.

The remarks so far make clear that HDTP already covers different types of reasoning:

- Analogical inferences are used to establish the analogical relation between source and target.
- Anti-unification computes generalizations that can be interpreted as general hypotheses about the structure of the instance theories.
- Deductive inferences are used to rewrite the input for applicability of anti-unification.
- Vague reasoning is implicitly covered by the approach, because analogies are intrinsically fuzzy. There are no right or wrong analogies, rather there are psychologically preferred and psychologically less preferred analogies. HDTP models this aspect by using heuristics.

In alternative analogy models other types of reasoning were proposed for the integration into the analogy making process. As an example abductive reasoning is mentioned that can be modeled in the structure mapping engine (Falkenhainer, Forbus, and Gentner, 1989).

⁴The generalization has similarities to an induction step. Nevertheless, it is important to notice that anti-unification is governed by two structured theories and not by many examples like in the case of classical inductive reasoning.

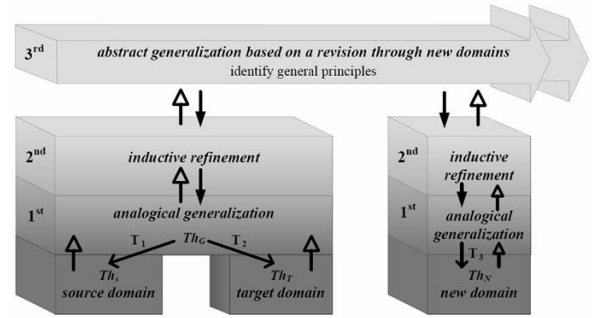


Figure 1: A graphical representation of the analogy making process involving different forms of reasoning and learning

Remarks on Learning with HDTP

As mentioned above not only the variety of different types of reasoning, but also the variety of different types of learning jeopardize the integration of learning into an AGI methodology, as well as learning from sparse and noisy data. The analogy making process presented here gives us some hints for possible solutions of some of these problems. First, notice that analogies do not require many data points. In fact, the input is given by just two theories (in the regular case). This suffices to learn a new conceptualization of the target and to find a generalization of both, source and target. Second, in (Kühnberger et al., 2008) the authors argue that learning from inconsistencies can be seen as a basic principle for adaptation processes in the cognitive architecture I-Cog: clashes occurring in reasoning, learning, or representation processes can trigger adaptation procedures that resolve such clashes. HDTP can be interpreted as one example where such adaptation processes can be specified on the algorithmic level of establishing analogies.

We explain some further issues in the analogy making process with respect to learning using Figure 1.

1. Level: The source and the target domain are compared via analogy to identify common structures between the domains. The commonalities are generalized to a theory Th_G . The analogy making process can transfer knowledge from the source to the target yielding a new conceptualization of the target. This results in a learning effect on the target side.
2. Level: The formulas projected from the source to the target need to be tested, because the analogical knowledge transfer might be true only for prototypical situations. The process runs through a number of different refinement steps and yields a parameter setting specifying in which range an analogy holds.
3. Level: The aim of this level is the identification of general principles. This type of learning requires the comparison (typically analogical comparison) of many different domain theories. At this level, the learning process starts with an intuitive hypothesis about a general principle and compares this iteratively with other domains to gain more confidence.

	Symbolic Approaches	Subsymbolic Approaches
Methods	(Mostly) logical and/or algebraic	(Mostly) analytic
Strengths	Productivity, Recursion Principle, Compositionality	Robustness, Learning, Parsimony, Adaptivity
Weaknesses	Consistency Constraints, Lower Cognitive Abilities	Opacity, Higher Cognitive Abilities
Applications	Reasoning, Problem Solving, Planning etc.	Learning, Motor Control, Vision etc.
CogSci Relation	Not Biologically Inspired	Biologically Inspired
Other Features	Crisp	Fuzzy

Table 4: Differences between symbolic and subsymbolic theories

The three levels proposed above outline the different mechanisms that occur. Analogical learning in this sense is therefore a process in which different learning mechanisms interact and are iteratively repeated to refine and correct knowledge on the various abstraction levels.⁵

Logic as Lingua Franca for AGI Systems

In this section, we sketch ideas of how logic as the underlying representation formalism can be used for integrating subsymbolic devices to a logical reasoning system.

Building Neural-Symbolic Learning Systems

There is an obvious gap between symbolic and subsymbolic theories. As a matter of fact there is not only a methodological difference between these approaches, but furthermore strengths and weaknesses of the two paradigms are complementary distributed. Table 4 mentions some important distinctions between these two types of modeling options. In particular, for an AGI system intended to cover a large part of the breadth of cognitive abilities, a natural idea would be to bring these two research traditions together.

There has been the research endeavor “neural-symbolic integration” attempting to resolve this tension between symbolic and subsymbolic approaches.⁶ The idea is to transform a highly structured input (e.g. a logical theory) into a flat input appropriate for connectionist learning. It has been shown that learning theories of propositional logic with neural networks can be achieved by using different frameworks like the “core method” or KBANNs (Hölldobler and Kalinke, 1994; Towell and Shavlik, 1994). In recent years, extensions to predicate logic were proposed (Bader, Hitzler, and Hölldobler, 2008).

We sketch some ideas of learning predicate logic with neural networks based on topos theory (Gust, Kühnberger, and Geibel, 2007). First, we introduce some concepts. A topos \mathcal{T} is a category where all finite diagrams have limits (and colimits), any two objects have an exponential object,

⁵This distinguishes HDTP from other learning approaches based on analogy. For example, the SME tradition (Falkenhainer, Forbus, and Gentner, 1989) focuses on alignment and transfer, whereas abstraction and interactions of different mechanisms do not play a crucial role.

⁶A good overview of neural-symbolic integration can be found in (Hammer and Hitzler, 2007).

and there exists a subobject classifier (Goldblatt, 1984). As a consequence, \mathcal{T} has an initial and terminal object, as well as finite products and coproducts. The prototypical example for a topos is the category \mathcal{SET} . The objects of \mathcal{SET} are sets, connected by set theoretic functions (called arrows). A product $a \times b$ can be identified with the well-known Cartesian product of two sets a and b , and an exponent a^b with the set of functions $f : b \rightarrow a$. The terminal object $!$ is the one-element set $\{0\}$ with the property that for all sets a there is exactly one arrow from a into $\{0\}$. The truth value object $\Omega = \{0, 1\}$ and the subobject classifier $true: ! \rightarrow \Omega$ mapping 0 to 1 generalizes characteristic functions and therefore interpretations of predicates.

We summarize how a topos can be used for neural learning of a logical theory T given in a first-order language \mathcal{L} .

- T is translated into a variable-free representation in a topos \mathcal{T} . The result is a representation of logic expressions by commuting diagrams, where objects are sets and arrows are set theoretic functions (in case we work in \mathcal{SET}). Logical terms can be interpreted as mappings from the terminal object into the universe U and logical 2-ary connectives as mappings $\Omega \times \Omega \rightarrow \Omega$. Quantified formulas correspond to an operation mapping (complex) predicates to (complex) predicates.
- An algorithm is generating equations of the form $f \circ g = h$ and inequations of the form $f \neq g$ corresponding to equations and inequations of arrows in the \mathcal{T} , i.e. set theoretic functions in \mathcal{SET} .
- As representations for objects and arrows it is possible to choose vectors of the vector space \mathbb{R}^n . The resulting equations and inequations can be used in order to train a feedforward neural network by backpropagation.
- The network learns a representation of objects and arrows of the underlying topos (which themselves represent symbols and expressions of \mathcal{L}), such that the truth conditions of the underlying axioms of T are satisfied. In other words, the network learns a model of T .

Although the approach still has many problems to solve, the very possibility to code axioms of a logical theory T , such that a neural network can learn a model of T , can be considered as the missing link between symbolic and subsymbolic representations. Applied to AGI systems, this means that logic can play the role of a lingua franca for general intelligence, without neglecting one of the two separated worlds of symbolic and subsymbolic computations. Rather it is possible to integrate both worlds into one architecture. A proposal for such an architecture (I-Cog) integrating both devices – the analogy engine and the neural-symbolic integration device – can be found in (Kühnberger et al., 2008).

Related Work

Analogies have been playing an important role in cognitive science and cognitively inspired AI for a long time. Classical frameworks are, for example, the symbolic approach SME (Falkenhainer, Forbus, and Gentner, 1989), the connectionist system LISA (Hummel and Holyoak, 1997), or the hybrid approach AMBR (Kokinov and Petrov, 2001). A

good overview of analogy models can be found in (Gentner, Holyoak, and Kokinov, 2001). There are numerous classical papers on neural-symbolic integration, e.g. (Barnden, 1989) as one of the early ones. More recent work is concerned with extensions of propositional logic (D’Avila Garcez, Broda, and Gabbay, 2002), and the modeling of predicate logic using the “core method” (Bader, Hitzler, and Hölldobler, 2008).

Conclusions

This paper argues for the usage of logic as the basis for an AGI system without neglecting other nature-inspired approaches for modeling intelligent behavior. Although there is a zoo of logical formalisms that are sometimes hard to integrate with each other, we claim that directions towards such an integration of a variety of different reasoning types already exist. As an example we proposed the analogy engine HDTP in order to integrate such diverse types of reasoning like analogical, deductive, inductive, and vague reasoning. Furthermore, non-trivial learning mechanisms can be detected by adapting and fine-tuning the analogical relation. Finally, this paper claims that even the interaction between symbolic theories and subsymbolic theories can be achieved by the usage of techniques developed in neural-symbolic integration.

Obviously, many issues remain open. Besides the challenge of an empirical justification of the presented framework, other issues for future work need to be addressed. Currently it is impossible to solve challenging problems (e.g. benchmark problems) for theorem provers with neural-symbolic integration. Similarly, many questions of analogy making remain open and are not solved yet. One example is the integration of large knowledge bases, in particular, the retrieval of relevant knowledge, another one is the scalability of analogy making to applications in the large. Nevertheless, taking these ideas together it turns out that a variety of different cognitive abilities can be addressed in a uniform framework using logic as a mediating tool.

References

- Bader, S., Hitzler, P., and Hölldobler, S. 2008. Connectionist model generation: A first-order approach. *Neurocomputing*, 71:2420–2432.
- Barnden, J.A. 1989. Neural net implementation of complex symbol processing in a mental model approach to syllogistic reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 568-573.
- Brachman, R. and Schmolze, J. 1985. An Overview of KL-ONE Knowledge Representation System, *Cognitive Science* 9(2):171–216.
- D’Avila Garcez, A., Broda, K., and Gabbay, D. 2002. *Neural-Symbolic Learning Systems: Foundations and Applications*. Berlin Heidelberg, Springer.
- Falkenhainer, B., Forbus, K., and Gentner, D. 1989. The structure-mapping engine: Algorithm and example, *Artificial Intelligence*, 41:1-63.
- Gentner, D., Holyoak, K., and Kokinov, B. 2001. *The Analogical Mind. Perspectives from Cognitive Science*, Cambridge, MA: MIT Press.
- Goldblatt, R. 1984. *Topoi, the categorical analysis of logic*. North-Holland, Amsterdam.
- Gust, H., Kühnberger, K.-U., and Geibel, P. 2007. Learning Models of Predicate Logical Theories with Neural Networks Based on Topos Theory. In: P. Hitzler and B. Hammer (eds.): *Perspectives of Neural-Symbolic Integration*, SCI 77, Springer, pp. 233-264.
- Gust, H., Kühnberger, K.-U., and Schmid, U. 2006. Metaphors and Heuristic-Driven Theory Projection. *Theoretical Computer Science*, 354:98-117.
- Hammer, B. and Hitzler, P. (eds.) 2007. *Perspectives of Neural-Symbolic Integration*. Springer, Berlin.
- Hasker, R. 1995. The replay of program derivations. PhD thesis, Champaign, IL, USA.
- Hölldobler, S. and Kalinke, Y. 1994. Ein massiv paralleles Modell für die Logikprogrammierung. *Proceedings of the Tenth Logic Programming Workshop*, WLP 94:89-92.
- Hummel, J. and Holyoak, K. 1997. Distributed representations of structure: A theory of analogical access and mapping, *Psychological Review* 104(3):427–466.
- Kokinov, B. and Petrov, A. 2001. Integrating Memory and Reasoning in Analogy-Making: The AMBR Model, in D. Gentner, K. Holyoak, B. Kokinov (eds.): *The Analogical Mind*. Perspectives from Cognitive Science, Cambridge Mass. (2001).
- Krumnack, U., Schwering, A., Kühnberger, K.-U., and Gust, H. 2007. Restricted Higher-Order Anti-Unification for Analogy Making. *20th Australian Joint Conference on Artificial Intelligence*, Springer, pp. 273-282.
- Krumnack, U., Gust, H., Kühnberger, K.-U., and Schwering, A. 2008. Re-representation in a Logic-Based Model for Analogy Making. *21st Australian Joint Conference on Artificial Intelligence*.
- Kühnberger, K.-U., Geibel, P., Gust, H., Krumnack, U., Ovchinnikova, E., Schwering, A., and Wandmacher, T. 2008. Inconsistencies in an Integrated Cognitive Architecture. In: P. Wang, B. Goertzel, and S. Franklin (eds.): *Artificial General Intelligence 2008. Proceedings of the First AGI Conference*, IOS Press, pp. 212-223.
- McCarthy, J. 1963 Situations, actions, and causal laws. *Stanford Artificial Intelligence Project Memo 2*, Stanford University, 1963.
- Minsky, M. 1975. A Framework for Representing Knowledge. In P. Winston (ed.): *The Psychology of Computer Vision*, New York, pp. 211–277.
- Plotkin, G. 1970. A note on inductive generalization. *Machine Intelligence* 5:153-163.
- Schwering, A., Krumnack, U., Kühnberger, K.-U. and Gust, H. 2009. Syntactic Principles of HDTP. To appear in *Cognitive Systems Research*.
- Sowa, J. 1987. Semantic Networks. In S. Shapiro (ed.): *Encyclopedia of Artificial Intelligence*, John Wiley & Sons.
- Towell, G. and Shavlik, J. 1994. Knowledge-Based Artificial Neural Networks. *Artificial Intelligence* 70(1-2):119-165.
- Wang, P. 2006. *Rigid Flexibility: The Logic of Intelligence*, Springer, 2006.

The robotics path to AGI using SERVO STACKS

J. Storrs Hall

Institute for Molecular Manufacturing
Laporte, PA, USA

Abstract

The case is made that the path to AGI through cognitive and developmental robotics is compelling. Beyond the familiar argument that it keeps researchers honest by forcing their systems to cope with the real world, it encourages them to recapitulate the evolutionary developmental path which gave rise to intelligence in humans. Insights from this perspective are embodied in the SERVO STACKS cognitive architecture with several salient features. The brain evolved as a body controller and thus is based largely on computational structures appropriate to physical process control. Evolution typically copies existing structure and modifies it minimally to meet a new demand. We should therefore expect the higher cognitive functions to be performed by body controllers pressed into service as brain controllers.

Introduction and Motivation

The brain evolved as a body controller; except for size, the human brain is structurally similar to that of other primates. Neocortex is surprisingly homogeneous, suggesting a general computing fabric instead of hard-wired functional modules. This uniformity has been cited as a strong argument for a common computational function (Mountcastle 1978). An argument can thus be made that evolution took a computing substrate evolved for body control and simply extended it to support the complex cognitive functions of symbolic ratiocination characteristic of human intelligence.

The SERVO STACKS architecture is an attempt to use this insight as a guide in developing a general artificial intelligence. For example, there is some evidence that the neural structures which manipulate grammar and language developed from, or indeed overlap, the ones which produce finely nuanced and sequenced manipulations by the hands. Thus we feel that insight into the former may be gained by investigating the latter, particularly to the extent of identifying mechanisms that might be responsible for compliant dexterity and seamless fluidity in both domains.

Copyright © 2008, The Second Conference on Artificial General Intelligence (AGI-09.org). All rights reserved.

It is common in higher-level cognitive architectures for there to be a chain something like “sensory input to interpretation to cogitation to action sequencing to motor output.” In evolution, however, the structure of a simpler animal is usually augmented by adding a controller to the top the entire lower-form structure, giving rise to a laminar structure with crosstalk between input and output at each level. This form is clearly reflected in the control architectures of modern robotics as pioneered in (Brooks 1986).

To extend this laminar control architecture to be a more general-purpose cognitive system, we reinterpret the afferent and efferent signals at each level, together with their crosstalk in both directions, as a feedback-loop servo controller. The resulting stack of servos then represents a classic abstraction hierarchy, with each successive servo “thinking” about what is happening in terms of a more general “language” as we ascend the stack.

For example, we might have lower-level servos concerned with pressure patterns on the skin and muscle activation strengths; a higher level with joint angles and speeds; higher yet with step-taking and foot placement; then room-to-room navigation; then the task of serving coffee; then the entertainment of guests, and so forth.

Given this general form for a cognitive architecture, the key open questions are

- Can a single general model of servo be developed that is appropriate to a broad range of these levels? If so, the notion that the brain grows by copying and modifying existing servos, both evolutionarily and in individual mental development, would gain increased cogency.
- The servos in a working model of mind will not form a simple linear stack but a complex network, which must support varying patterns of communication for different overall cognitive tasks and states. How is this done?
- For mental growth and learning, new servos, with new internal “languages”, must be formed to implement the ability to think in new abstractions about new concepts. How are they created, programmed, and connected?

After examining these we will return to the question of a specific architecture.

A Symbolic Servo

Upon seeing the algorithm of Newell and Simon’s General Problem Solver, John McCarthy quipped, “It’s a symbolic servo.”¹

A standard process-control servo receives a control input signal s (the “setpoint”) and a feedback signal f from the process (called the “plant”) to be controlled. These two signals must be commensurate, e.g. both a measure of engine speed. The servo then adjusts some controllable parameter p of the plant based on a function of the control and feedback signals.

If the servo has a memory of enough cases, arranged as triples (f_0, f_1, p) (e.g. the value of the feedback signal before and after setting the output to p), “case-based” control can be obtained by using p from the tuple where f_0 is closest to current feedback f , and f_1 is nearest the current control input s . For well-behaved plants, even a sparsely populated memory can be interpolated with standard numerical regression techniques.

The SERVO STACKS model specifies a controller whose memory is (s, p, d, f) , where s is the setpoint, $f = f_0$ is the original value of the feedback signal, $d = f_1 - f_0$ the differential of the trajectory, and p is the process control signal sent to the plant. The use of a differential as a trajectory link (instead of explicitly storing the successive value) is prompted by the observed difficulty of following well-practiced behaviors backwards, and because matching differentials affords a cheap generalization over translation, and with a nod to the “difference operators” in GPS (Newell and Simon 1963). It also brings us within shouting distance of the standard transfer function formulation of linear dynamical systems in control theory, although we have yet to take practical advantage of this.

We refer to this as a “sigma” (*situation / goal / memory / action*) to distinguish it from standard formulations of servo or state machine. In our model, each signal has a strength as well as a value; at zero strength, it acts as a “don’t-care”, and at partial strength it acts as a bias in case the other inputs incompletely determine a memory record, but can be overridden by stronger signals on other inputs. More precisely, each signal is a vector whose components individually have strengths.

If sufficiently populated, the stored trajectories form a manifold in the memory space, which is equivalent to an equation (or system of them) which forms a model of the observed behaviors. Typically the manifold will be a p -valued function of (f, s) -space, and generally interpreted as a function selected by s in f -space.

If some of the dimensions of the space are time derivatives of others, it forms a dynamical systems model (equivalent to a system of differential equations). In the absence of such derivatives, the sigma can either be

clocked to obtain state-machine behavior or allowed to run free, in which case it will find fixed points of its function.

A sigma can be used in a variety of modes, depending on its inputs:

1. Homeostatic servo mode: as above, s is driven by a control signal from above, f is driven by the feedback signal from below, d is sent back up as a feedback. p is output to drive the plant, and may optionally be added into d as part of the feedback.
2. Sequencing control mode: s is driven by control from above, $d + f$ is output to drive f , and $p + d$ is sent back up as feedback.
3. Simulate mode: s is driven by control from above, $d + f$ is output to drive f , and $p + d$ is sent back up as feedback.
4. Recognize mode: f and d are driven by the signal from below and its derivative; s and $p + d$ are output to be sent back up as feedback. s may also be fed back weakly to itself, and/or driven weakly from above for a priming effect.

Signals and Representation

Analogy (Hofstadter 1995) and blending (Fauconnier and Turner 2003) have both been suggested as key basic operations that a cognitive architecture must implement. It is instructive to note that vectors of real numbers support both these operations as simple geometric combinations, while representing concepts as complex as a recognizable sketch of a human face (Churchland 1986; 2005). Although at higher levels of abstraction it is surely the case that more complex structures and operations will be necessary to support analogy and blending, it seems reasonable to begin with a primitive representation that provides a head start.

The physical signals in Servo Stacks are fixed-length numeric vectors of length n ($n = 1024$ in current experiments), but support an arbitrary number of notional signals. The components of the notional signals are given a superimposed coding as sums of randomly-chosen base vectors in the physical signal space. Any two such vectors have a high probability of being nearly orthogonal (Kanerva 1988). Any port on any sigma can be connected to any other port, with a high probability that notional signals unknown to the receiver will simply be ignored (by virtue of being orthogonal to the active manifold). Notional signals can be generated locally without the need of an overall registry, and can be combined simply by adding physical signals. Hereinafter, we shall ignore the encoding and refer only to the notional signal vectors of arbitrary length.

Such vectors can obviously record the position in configuration space of an arm, or be interpreted as a raster (including the areal functions of dynamic neural field theory (Erlhagen and Bicho 2006)), or specify force values for muscles. However, throughout most of the architecture, they will typically record the activation of,

¹As related by Marvin Minsky at AAAI FSS 2001, North Falmouth, MA

and signals to be sent to, sets of other sigmas. In the sense that such a vector can be thought of as representing a situation as recognized or understood by the other sigmas, it functions as a frame (Minsky 1975). In cases where that is an appropriate interpretation, the sigma itself acts as a frame-system, predicting new situations as the result of actions or suggesting actions given desired situations.

It is perhaps important to emphasize that there is not one big vector space in which everything is represented, as in some connectionist theories. Each sigma has its own language, in the sense of an interpretation of the vectors. More precisely, each pair of sigmas which communicate have a common interpretation for the signals that pass between them, but these interpretations vary completely from one area to another.

Stacks and Networks

Servos are commonly arranged in stacks in complex mechanisms. Upper-level servos may send control signals directly to lower ones (the attitude-control system in a fly-by-wire aircraft commanding aileron servos) or indirectly through a process parameter (room thermostats in a home hydronic heating system run hot-water pumps, relying on a homeostasis provided by a boiler thermostat). Leading roboticists Albus (Albus 1992) and Brooks (Brooks 1986) have based their flagship architectures on hierarchies of FSAs.

Thus the SERVO STACKS model, as a layered network of elements that can act as sequencing or homeostatic controllers, is straightforwardly adapted into these roles. A key difference between this and a conventional view of a control hierarchy, however, is that sigmas by their nature provide a sensing and translation function. The setpoint and feedback signals s and f are necessarily in a different “language” than p , and thus, in a directly connected stack, from the s and f of the sigma below.

A servo stack is as reasonable a model for a sensory pathway as for a motor one. It is becoming increasingly recognized that sensing is neurophysiologically an active, not a passive process (Wolpert, Ghahramani, and Jordan 1995). In vision, for example, there are examples of homeostasis, such as iris dilation for retina illumination levels, and sequencing control, as of saccading in intermediate object recognition to trace boundaries and salient features.

Holonic recognition, including key features such as priming and context sensitivity, is readily provided for in SERVO STACKS by weakly driving the s input from above. A key feature of the superimposed coding of signals is that the signal representing a percept can have components corresponding to several possibilities, and the interpretation process may provide (efferent!) feedback from semantic constraints.

At the lower sensorimotor levels, the reconfigurability of the sigmas is not important, and indeed there may be a level below which they are hard-wired and cognitively

impenetrable. However, at higher levels where sigmas represent concepts such as words and goal-directed actions, reconfigurability is crucial to the use of the network as a fabric of representation. The same set of sigmas can be used to recognize an action, imagine doing it, predict its effects, and actually to perform it.

Recursion and Concepts

We posit a mechanism similar to Minsky’s *k-line* (Minsky 1980), which can record all the network elements active at a given point, but also able to record their connections: which port on each is driving which other ports at what strength. These *active subnet configurations* (hereinafter ASCs) can be named and passed as values between higher-level sigmas, which can perform operations on them such as substitution: the ASCs for “pick up the red block”, “red block”, and “blue ball” can be combined in such a way as to form an ASC for “pick up the blue ball”.

The formation of ASCs is at least neurally plausible. A generally broadcast signal could cause all active elements to associate a code (one component of the notional vector) with their current activity or connectivity, and a similar broadcast could cause that activity and connectivity to be reestablished. A number of associative datastructure manipulation techniques are similarly plausible for operations between ASCs (Foster 1976; Potter 1991; Hall 1994).

Since ASCs are capable of recursion (in the linguistics sense, i.e. capable of being combined to form complex structures from simple ones) and other typically symbolic manipulations, they form a “language of thought” in the sense of Fodor (Fodor 1975; 1978).

Perhaps the most important question one can ask of a cognitive architecture is how it represents concepts. In the common quip, something is a duck if it walks like a duck and quacks like a duck. Rather than representing a duck as some static datastructure essentially equivalent to a dictionary entry, an ASC represents a duck with an active *working machine* that is capable of recognizing a duck, simulating a duck, and even imitating a duck. This is the active subnet which activates and connects all the sigmas which store duck-relevant trajectories.

Note again that ASCs are manipulated, not by some exogenous program, but by sigmas whose memories p are records of ASC-manipulation control signals. The s and f signals to such sigmas are typically the output of more complex processing and recognition. The higher-level sigmas which manipulate ASCs are no different in principle from any others – manipulating one’s own thoughts must be learned and practiced.

Play, Practice, and Imitation

Evidence ranging from visually deprived kittens (Wiesel and Hubel 1963) to language-deprived children (SACKS 1989) indicates that appropriate sensory experience is necessary for the development of cognitive ability across

a wide range of levels of organization. The same phenomenon would affect our sigma, which would be incompetent at its task in the absence of a populated memory.

For a sigma to become competent at recognition or control, its memory must be populated with enough trajectories to form a reasonable model of the dynamics of the space implied by its signals. Underpopulated sigmas can improve their models by indulging in play: driving outputs so that the resulting state falls into vacant regions of the space. The plant to be driven in such play can be either the actual robot body, through the entire sensorimotor stack below the playing sigma, or simulations, in which case the stack is disconnected by putting some cutset of intermediate sigmas into simulate mode.

By far the major mode of human learning is imitation. After a sigma records the experience of someone else doing an action, the action may be imitated substituting oneself for the original actor in the ASC. This will rarely be perfect, but it gives the student mind a scaffolding upon which to improve by practice.

An Architecture

The testbed for SERVO STACKS is an upper-torso anthropoid robot in the genre of Cog (Brooks et al. 1999). The robot is intended to be able to learn enough skills and concepts to be roughly the equivalent of a SHRDLU (Winograd 1972), but with physical cameras, manipulators, wooden blocks and table, and hearing and responding in spoken language.

Given the demanding computational and memory requirements of the SERVO STACKS model, particularly the associative memory of the sigmas, it seems likely that processing power will form a substantial bottleneck for the near future. We consider this appropriate, however: any biologically inspired theory of the mind must take into account the substantial apparent processing power of the brain (Merkle 1987). Any such theory whose computational requirements fit available computer hardware too neatly seems suspiciously *ad hoc*.

Autogenous Kernel

We are primarily concerned with the ability of a mind to learn and grow. We adopt the basic architecture of a self-extending system from (Hall 1999), which specifies an “autogenous kernel” with irreducibly basic self-construction capabilities, which builds, in successive layers, a series of extensions that have both more general capabilities and more sophisticated self-construction abilities.

In a cognitive architecture, the kernel consists of some basic sensorimotor sigmas, pre-programmed with records that allow for infant-like activities like waving arms and learning to track them with eyes. Perhaps more importantly, it contains higher-level sigmas pre-programmed with records that allow them to do basic

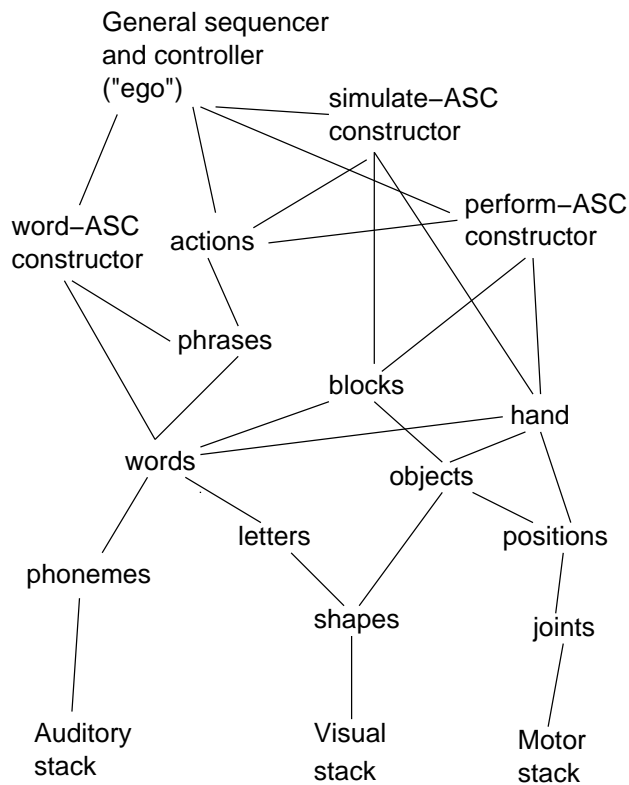


Figure 1: Kernel robot cognitive architecture.

ASC manipulation.

The key issue in learning is the provenance of the mappings of signals and connectivity of the sigmas. For example, consider the sigma that maps from 3-D space to the configuration space of a multi-jointed arm, constituting the forward or inverse kinematics function depending on how it is used. This works after it is populated – where did the 3-D space come from originally, however?

The process starts with a handful of data-mining heuristics that are applied more or less at random to all the signals in the system. In early experiments these are Kohonen map formation, other standard dimensionality reduction techniques such as PCA, and hierarchical clustering using affinity propagation (Frey and Dueck 2007).

Sigmas generated this way are run in prediction mode and compete in an agoric/genetic ecosystem (Hall, Steinberg, and Davison 1998) on the basis of how well their predictions match experience. Those that succeed are linked together by the supply-chain dynamics of the agoric phase of the ecosystem and continue to compete in the marketplace. (The top-level inflow of funds in this system corresponds to motivations, and is currently a very simplistic stub that provides for answering questions and performing requested actions.)

Sensorimotor stack confluence

A key feature of the SERVO STACKS model is that the stacks implementing any given sensorimotor modality (such as vision) are not separate but merge into other stacks at relatively low levels. In the testbed robot architecture, the main stacks are vision, hearing, and manipulation.

- Vision and manipulation converge at a low level to allow a fairly tight hand-eye coordination control loop. The upper end of this confluence feeds into an object-model stack.
- At a somewhat higher level, vision and hearing converge at the level of letters/words, feeding into a language stack.
- The language stack and object model converge at a level such that model semantics are active in sentence parsing, as in SHRDLU.

When the robot hears the sentence, “Dutch the blue blog,” the sigmas which store word and syntax-related trajectories – words are trajectories through phonemes, sentences are trajectories through words – are configured to perform a spreading-activation parse similar to a Jones APN (Jones 1983; Jones and Driscoll 1985). There is enough feedback in this process to force the recognition of “Touch the blue block.” The recognition itself will involve setting up the performance sigmas in simulation mode (at a high level only) to verify that the meaning is within the robot’s experience. This will amplify the salience of “touch” and diminish that of “dutch”, etc.

Memory

There is no separate module labelled “memory” in the architecture. Long-term memories are represented by the trajectory memories in all the sigmas and the mappings from signals to ASCs. Short-term or working memory is represented by the content of the active signals (including which ASCs are active).

Memories within a given sigma are managed, in our early experiments, by clustering and weighting heuristics (and by having each sigma have a fixed, limited capacity). Segmentation of trajectories into distinct traces is frankly *ad hoc* and is an area of active investigation.

Related Work

SERVO STACKS falls squarely in the field of cognitive robotics (Clark and Grush 1999; Sloman et al. 2006). It shares in particular a strong concern for ontogenetic development with such projects as iCub (Tsagarakis et al. 2007). It is distinguished from many of the specific efforts in developmental robotics (Lungarella et al. 2003), however, by focussing on function, representation, and organization at a higher level than a neural network model (Shanahan 2006) or even a dynamic neural fields model (Erlhagen and Bicho 2006).

Minsky and Papert’s original “Society of the Minds” cognitive architecture (Minsky 1977) was considerably more neurally inspired (and more oriented toward mental growth) than the majority of the “agent-based” architectures which followed – although the latter typically had the advantage of being more well-specified and indeed actually being implemented. The present work is strongly inspired by SoM but differs from it in several significant ways, particularly in the notion that ASCs can be handed from agent to agent as values. (Note that the first published mention of a rule-based controller reconfigurable as a simulator is in Marvin Minsky’s Princeton dissertation (Minsky 1954)!)

Our model follows Grush’s theory (Grush 2004) of representation as emulation based on forward modelling in Kalman filters and presumed equivalent functionality in the brain in several ways.

The notion of simply using the full record of experience as a model is generally referred to in AI as “case-based reasoning” (Kolodner 1992). Some of the neural-network approaches that allow one-shot learning are the original Hopfield network (with its Hebbian programming algorithm) (Hopfield 1982) and Aleksander’s G-RAM neural unit model (Aleksander 1990). SERVO STACKS might be implemented in either of these; for example, similar in spirit to our use of sigmas as associative-memory based sequencers is Orponen’s programming language for Hopfield nets (Orponen and Prost 1996). However, for any digital simulation, an exhaustive search of a vector list is as fast as one single iteration of a Hopfield relaxation, so we choose instead to concentrate on the abstract properties of proximity in n-spaces and assume that conventional techniques can be developed to implement them efficiently (Shakhnarovich, Darrell, and Indyk 2006; Garcia, Debreuve, and Barlaud 2008).

References

- Albus, J. S. 1992. Rcs: A reference model architecture for intelligent control. *IEEE Computer* 25(5):56–59.
- Aleksander, I. 1990. Neural systems engineering: towards a unified design discipline? *Computing and Control* 1:259–265.
- Brooks, R. A.; Breazeal, C.; Marjanovic, M.; Scassellati, B.; and Williamson, M. M. 1999. The cog project: Building a humanoid robot. *Lecture Notes in Computer Science* 1562:52–87.
- Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation* RA-2:14–23.
- Churchland, P. M. 1986. Some reductive strategies in cognitive neurobiology. *Mind* 95(379):279–309.
- Churchland, P. M. 2005. Vector completion, relevant abduction, and the capacity for ‘globally sensitive’ inference. In Raftopoulos, A., ed., *Cognitive Penetrability of Perception: Attention, Action, Strategies, and*

- Bottom-Up Constraints*. Nova Science Publishers. 1–12.
- Clark, A., and Grush, R. 1999. Towards a cognitive robotics. *Adaptive Behavior* 1(7):5–16.
- Erlhagen, W., and Bicho, E. 2006. The dynamic neural field approach to cognitive robotics. *Journal of Neural Engineering* 3:36–54.
- Fauconnier, G., and Turner, M. 2003. Conceptual blending, form and meaning. *Recherches en communication* 19(19):57–86.
- Fodor, J. 1975. *The Language of Thought*. Thomas Y. Crowell Company.
- Fodor, J. 1978. Tom Swift and his procedural grandmother. *Cognition* 6:204–224.
- Foster, C. C. 1976. *Content Addressable Parallel Processors*. Wiley.
- Frey, B. J., and Dueck, D. 2007. Clustering by passing messages between data points. *Science* 315:972–976.
- Garcia, V.; Debreuve, E.; and Barlaud, M. 2008. Fast k nearest neighbor search using gpu.
- Grush, R. 2004. The emulation theory of representation: Motor control, imagery and perception. *Behavioral and Brain Sciences* 27(3):377–442.
- Hall, J. S.; Steinberg, L.; and Davison, B. D. 1998. Combining agoric and genetic methods in stochastic design. *Nanotechnology* 9(3):274–284.
- Hall, J. S. 1994. *Associative Processing: Architectures, Algorithms, Applications*. Ph.D. Dissertation, Rutgers University.
- Hall, J. S. 1999. Architectural considerations for self-replicating manufacturing systems. *Nanotechnology* 10(3):323–330.
- Hofstadter, D. 1995. *Fluid Concepts and Creative Analogies*. New York: Basic Books.
- Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79. Washington, DC: National Academy Press.
- Jones, M. A., and Driscoll, A. S. 1985. Movement in active production networks. In *Meeting of the Association for Computational Linguistics*, 161–166.
- Jones, M. A. 1983. Activation-based parsing. In *IJCAI-VIII*, 678–682.
- Kanerva, P. 1988. *Sparse Distributed Memory*. Cambridge, MA: MIT Press.
- Kolodner, J. L. 1992. An introduction to case-based reasoning. *AI Review* 6:3–34.
- Lungarella, M.; Metta, G.; Pfeifer, R.; and Sandini, G. 2003. Developmental robotics: a survey. *Connection Science* 15(4):151–190.
- Merkle, R. 1987. Reverse engineering the brain. In *Proc. AIAA Computers in Aerospace VI*, 375.
- Minsky, M. 1954. *Neural-Analog Networks and the Brain-Model Problem*. Ph.D. Dissertation, Princeton University.
- Minsky, M. 1975. A framework for representing knowledge. In Winston, P. H., ed., *The Psychology of Computer Vision*. McGraw-Hill.
- Minsky, M. 1977. Plain talk about neurodevelopmental epistemology. In *IJCAI-V*, 1083–1092.
- Minsky, M. 1980. K-lines: A theory of memory. *Cognitive Science* 4(2):117–133.
- Mountcastle, V. B. 1978. An organizing principle for cerebral function: The unit model and the distributed system. In Edelman, G. M., and Mountcastle, V. B., eds., *The Mindful Brain*. MIT Press.
- Newell, A., and Simon, H. 1963. GPS: A program that simulates human thought. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. New York, NY: McGraw-Hill. 279–296.
- Orponen, P., and Prost, F. 1996. Parallel programming on hopfield nets. In *Proc. Finnish AI Conference*, 5–12.
- Potter, J. L. 1991. *Associative Computing: A Programming Paradigm for Massively Parallel Computers*. Perseus Publishing.
- Sacks, O. 1989. *Seeing Voices, A Journey into the World of the Deaf*. Berkeley, CA: University of California Press.
- Shakhnarovich, G.; Darrell, T.; and Indyk, P. 2006. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press.
- Shanahan, M. 2006. A cognitive architecture that combines internal simulation with a global workspace. *Consciousness and Cognition* 15:433–449.
- Sloman, A.; Wyatt, J.; Hawes, N.; Chappell, J.; and Kruijff, G.-J. M. 2006. Long term requirements for cognitive robotics. In *Proceedings CogRob2006, The Fifth International Cognitive Robotics Workshop. The AAAI-06 Workshop on Cognitive Robotics*.
- Tsagarakis, N. G.; Metta, G.; Sandini, G.; Vernon, D.; Beira, R.; Becchi, F.; Righetti, L.; Santos-Victor, J.; Ijspeert, A. J.; Carrozza, M. C.; and Caldwell, D. G. 2007. icub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics* 21(10):1151–1175.
- Wiesel, T. N., and Hubel, D. H. 1963. Single-cell responses in striate cortex of kittens deprived of vision in one eye. *J Neurophysiol* 26:1003–17.
- Winograd, T. 1972. *Understanding Natural Language*. Academic Press.
- Wolpert, D. M.; Ghahramani, Z.; and Jordan, M. I. 1995. An internal model for sensorimotor integration. *Science* 269(5232):1880–1882.

A Unifying Framework for Analysis and Evaluation of Inductive Programming Systems*

Martin Hofmann and Emanuel Kitzelmann and Ute Schmid

Faculty Information Systems and Applied Computer Science

University of Bamberg, Germany

{martin.hofmann, emanuel.kitzelmann, ute.schmid}@uni-bamberg.de

Abstract

In this paper we present a comparison of several inductive programming (IP) systems. IP addresses the problem of learning (recursive) programs from incomplete specifications, such as input/output examples. First, we introduce conditional higher-order term rewriting as a common framework for inductive logic and inductive functional program synthesis. Then we characterise the several ILP systems which belong either to the most recently researched or currently to the most powerful IP systems within this framework. In consequence, we propose the inductive functional system IGOR II as a powerful and efficient approach to IP. Performance of all systems on a representative set of sample problems is evaluated and shows the strength of IGOR II.

Introduction

Inductive programming (IP) is concerned with the synthesis of programs or algorithms from incomplete specifications, such as input/output (I/O) examples. Focus is on the synthesis of *declarative*, i.e., logic, functional, or functional logic programs. Research in IP provides better insights in the cognitive skills of human programmers. Furthermore, powerful and efficient IP systems can enhance software systems in a variety of domains—such as automated theorem proving and planning—and offer novel approaches to knowledge based software engineering and model driven software development, as well as end user programming support in the XSL domain (Hofmann 2007). Depending on the target language, IP systems can be classified as inductive logic programming (ILP), inductive functional programming (IFP) or inductive functional logic programming (IFLP).

Beginnings of IP research addressed inductive synthesis of functional programs from small sets of positive I/O examples only (Biermann et al. 1984). One of the most influential classical systems was THESYS (Summers 1977) which synthesised linear recursive LISP programs by rewriting I/O pairs into traces and folding of traces based on recurrence detection. Currently, induction of functional programs is covered by the analytical approaches IGOR I (Kitzelmann and Schmid 2006), and IGOR II (Kitzelmann 2007) and by

the evolutionary/search-based approaches ADATE (Olsson 1995) and MAGICHASKELLER (Katayama 2005). Analytical approaches work example-driven, so the structure of the given I/O pairs is used to guide the construction of generalised programs. Search-based approaches first construct one or more hypothetical programs, evaluate them against the I/O examples and then work on with the most promising hypotheses.

In the last decade, some inductive logic programming (ILP) systems were presented with focus on learning recursive logic *programs* in contrast to learning classifiers: FFOIL (Quinlan 1996), GOLEM (Muggleton and Feng 1990), PROGOL (Muggleton 1995), and DIALOGS-II (Flener 1996). Synthesis of functional logic programs is covered by the system FLIP (Hernández-Orallo et al. 1998).

IP can be viewed as a special branch of machine learning because programs are constructed by inductive generalisation from examples. Therefore, as for classification learning, each approach can be characterised by its restriction and preference bias (Mitchell 1997). However, IP approaches cannot be evaluated with respect to some covering measure or generalisation error since (recursive) programs must treat *all* I/O examples correctly to be an acceptable hypothesis.

Current IP systems not only differ with respect to the target language and the synthesis strategies but also with respect to the information which has to or can be presented and the scope of programs which can be synthesised. Unfortunately, up to now there is neither a systematic empirical evaluation of IP systems nor a general vocabulary for describing and comparing the different approaches in a systematic way (see (Hofmann, Kitzelmann, and Schmid 2008) for a preliminary evaluation of some systems and (Flener and Yilmaz 1999) for a treatment of ILP systems). We believe that both is necessary for further progress in the field: Only if the relative strengths and weaknesses of the different systems become transparent, more powerful and efficient approaches can be designed by exploiting the strengths of the given approaches and tackling their weaknesses.

We first present conditional combinatory term rewriting as a framework for describing IP systems. Afterwards, several systems are characterised and compared in this framework and their performance is evaluated on a set of representative sample problems and shows the strength of IGOR II. We conclude with ideas on further research.

*Research was supported by the German Research Community (DFG), grant SCHM 1239/6-1.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

A Unified Framework for IP

Conditional Constructor Systems

We shortly introduce term rewriting and conditional constructor systems as, e.g., described in (Baader and Nipkow 1998; Terese 2003). For a signature, i.e., a set of function symbols Σ and a set of variables \mathcal{X} we denote the set of all terms over Σ and \mathcal{X} by $\mathcal{T}_\Sigma(\mathcal{X})$ and the (sub)set of ground (variable free) terms by \mathcal{T}_Σ . We distinguish function symbols that denote datatype *constructors* from those denoting (user-)defined functions. Thus $\Sigma = \mathcal{C} \cup \mathcal{F}, \mathcal{C} \cap \mathcal{F} = \emptyset$ where \mathcal{C} contains the constructors and \mathcal{F} the defined function symbols respectively. We uniformly represent an induced program in a functional style as a set R of recursive equations (rules) over a signature Σ . The equations are applied as simplification (or rewrite) rules (as known from functional programming) from left to right, i.e., they form a *term rewriting system*. The lefthand side (lhs) l of a rewrite rule $l \rightarrow r$ has the form $F(p_1, \dots, p_n)$, called *function head*, where $F \in \mathcal{F}$ is the name of the function implemented by (amongst others) this rule, i.e., a defined function symbol, and the $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$ are built up from constructors and variables only. We call terms from $\mathcal{T}_\mathcal{C}(\mathcal{X})$ *constructor terms*. The sequence of the p_i is called *pattern*. This format of rules or equations is known as *pattern matching* in functional languages such as HASKELL. In the term rewriting setting, a TRS with this form is called *constructor (term rewriting) system (CS)*. All variables of the righthand side (rhs) must also occur in the lhs, i.e. they must be *bound* (by the lhs). If no rule applies to a term the term is in *normal form*.

Each rewrite rule may be augmented with a *condition* that must be met to apply the *conditional* rule. A term rewriting system or constructor system is called conditional term rewriting system or *conditional constructor system (CCS)* respectively if it contains at least one conditional rule. A condition is an ordered conjunction of equality constraints $v_i = u_i$ with $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$. Each u_i must be grounded if the lhs of the rule is instantiated and if all equalities $v_j = u_j$ with $j < i$ evaluate to true, then u_i evaluates to some ground normal form. For the v_i must hold (i) either the same as for the u_i or (ii) v_i may contain unbound variables but then it must be a constructor term. In the first case also v_i evaluates to some ground normal form and the equality evaluates to true if both normal forms are equal. In the second case the equality evaluates to true if v_i and the ground normal form of u_i unify. Then the free variables in v_i are bound and may be used in the following conjuncts and the rhs of the rule. We write conditional rules in the form: $l \rightarrow r \Leftarrow v_1 = u_1, \dots, v_n = u_n$. Figure 1(1) shows an example. Rules without a condition are called *unconditional*. If we apply a defined function to ground constructor terms $F(i_1, \dots, i_n)$, we call the i_i *inputs* of F . If such an application normalises to a ground constructor term o we call o *output*. A CCS is *terminating* if all rewriting processes end up in a normal form. In order to implement functions the outputs are required to be unique for each particular input vector. This is the case if the TRS is *confluent*.

To lift a CCS into the higher-order context and extend it to a (conditional) combinatory rewrite system

((C)CRS) (Terese 2003) we introduce meta-variables $\mathcal{X}_M = X, Y, Z, \dots$ such that $\mathcal{X} = \mathcal{X}_M \cup \mathcal{X}_T$ with $\mathcal{X}_M \cap \mathcal{X}_T = \emptyset$ where \mathcal{X}_T includes all first-order variables over terms. Meta-variables occur as $X(t_1, \dots, t_n)$ and allow for generalisation over functions with arity n . To preserve the properties of a CS, we need to introduce an abstraction operator $[-]$ to bind variables locally to a context. The term $[A]t$ is called abstraction and the occurrences of the variable A in t are bound. For example the recursive rule for the well-known function *map* would look like $map([A]Z(A), cons(B, C)) \rightarrow cons(Z(B), map([A]Z(A), C))$ and would match following term $map([A]square(A), cons(1, nil))$.

Target Languages in the CCRS Framework

To compare all systems under equal premises, the different occurrences of declarative languages are put into the CCRS framework¹. Considering functional target languages, the underlying concepts are either based on abstract theories (equational theory (Hernández-Orallo et al. 1998), CS (Kitzelmann 2007)), or concrete functional languages (ML (Olsson 1995), HASKELL (Katayama 2005)). Applying the CCRS framework to IFP or IFLP systems is straightforward, since they all share the basic principles and functional semantics. This is in particular the case with IFLP, which provided the theoretical framework for FLIP. However contrarily to IFLP, we additionally want to qualify for expressing the basic constructs of functional languages in the CCRS framework and both apply it to existing systems for a well-founded analysis and evaluation.

In addition to pattern matching and the functional operational semantics of CS, CCS can express constructs as *if*-, *case*-, and *let-expressions* in a rewriting context. The *if-then* part of an *if*-expression can be modeled by a condition $v = u$ following case (i) in the previous section. A *case*-expression is modeled following case (ii), where $v \in \mathcal{T}_\mathcal{C}(\mathcal{X})$ and $v \notin \mathcal{X}$. If $v \in \mathcal{X}$, case (ii) models a local variable declaration as in a *let*-expression. Fig. 1(2) shows a CCRS for the HASKELL program containing a *let*-expression.

In the context of IP, only logic target programs where the output is uniquely determined by the input are considered. Such programs usually are expressed as “functional” predicates such as *multlast* in Fig. 1(3). Transforming Horn clauses containing functional predicates into CCSs is a generalisation of representing Horn clauses as conditional identities (Baader and Nipkow 1998).

IP in the CCRS Framework

Let us now formalise the IP problem in the CCRS setting. Given a CCRS, both, the set of defined function symbols \mathcal{F} and the set of rules R , be further partitioned into disjoint subsets $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$ and $R = E^+ \cup E^- \cup BK$, respectively. \mathcal{F}_T are the function symbols of the functions to be synthesised, also called *target functions*. \mathcal{F}_B are the

¹Note the subset relationship between that CS, CCS, and CCRS. So, if the higher-order context is of no matter we use the term CCS, otherwise CCRS.

<p>(1) CCRS</p> <pre> mlast([]) -> [] mlast([A]) -> [A] mlast([A,B C]) -> [D,D C] <= [D C] = mlast([B C]) </pre>	<p>(2) Functional (Haskell)</p> <pre> mlast([]) = [] mlast([A]) = [A] mlast([A,B C]) = let [D C] = mlast([B C]) in [D,D C] </pre>
<p>(3) Logic (Prolog)</p> <pre> mlast([], []). mlast([A], [A]). mlast([A,B C], [D,D C]) :- mlast([B C], [D C]). </pre>	

Figure 1: Equivalent programs of *multlast*

symbols of predefined functions that can be used for synthesis. These can either be built in or defined by the user in the background knowledge BK . \mathcal{F}_I is a pool of function variables that can be used for defining invented functions on the fly. E^+ is the set of *positive examples* or *evidence* and E^- the set of *negative examples*, both containing a finite number of I/O pairs as unconditional rewrite rules $F(t_1, \dots, t_n) \rightarrow r$, where $F \in \mathcal{F}_T$ and $t_1, \dots, t_n, r \in \mathcal{T}_C(\mathcal{X}_T)$. The rules in E^- are interpreted as inequality constraints. BK is a finite set of conditional or unconditional rules $F(t_1, \dots, t_n) \rightarrow r \Leftarrow v_1 = u_1 \wedge \dots \wedge v_n = u_n$ defining auxiliary concepts that can be used for synthesising the target function, where $F \in \mathcal{F}_B$, $t_1, \dots, t_n \in \mathcal{T}_C(\mathcal{X})$, and $r, u_i, v_i \in \mathcal{T}_B(\mathcal{X})$. Furthermore, it is requested that for each symbol $f \in \mathcal{F}_T$ (\mathcal{F}_B), there is at least one rule in R_T (R_B) with function name f .

With such a given CCRS, the IP task can be now described as follows: find a finite set R_T of rules $F(t_1, \dots, t_n) \rightarrow r \Leftarrow v_1 = u_1 \wedge \dots \wedge v_n = u_n$ (or program for short) where $F \in \mathcal{F}$, $t_1, \dots, t_n \in \mathcal{T}_C(\mathcal{X})$, and $r, u_i, v_i \in \mathcal{T}(\mathcal{X})$, such that it covers all positive examples ($R_T \cup BK \models E^+$, *posterior sufficiency* or *completeness*) and none of the negative examples ($R_T \cup BK \not\models E^-$, *posterior satisfiability* or *consistency*). In general, this is done by discriminating between different inputs using patterns on the lhs or conditions modelling *case-expressions* and computing the correct output on the rhs. To compute the output constructors, recursive calls, functions from the background knowledge, local variable declarations, and invented functions can be used. An invented function is hereby a function which symbol occurs only in \mathcal{F}_I , i. e. is neither a target function nor defined in the BK and is defined by the synthesis system on the fly.

However, there is usually an infinite number of programs satisfying these conditions, e. g. E^+ itself, and therefore two further restrictions are imposed: A restriction on the terms constructed, the so called *restriction bias* and a restriction on which terms or rules are chosen, the *preference bias*.

The *restriction bias* allows only a specific subset of the terms defined for u_i, v_i, l, r in a rule $F(t_1, \dots, t_n) \rightarrow r \Leftarrow u_1 = v_1 \wedge \dots \wedge u_n = v_n$, e.g., prohibiting nested or mutual recursion or demanding the rhs to follow a certain scheme.

The *preference bias* imposes a partial ordering on terms, lhs, rhs, conditions or whole programs defined by the CCS framework and the restriction bias. A correct program synthesised by a specific system is optimal w. r. t. this ordering and satisfying completeness and consistency.

Table 1: Systems' characteristics summary

	\mathcal{C}	\mathcal{F}_T	\mathcal{F}_B	\mathcal{F}_I	E^+	E^-	BK	\mathcal{X}_M
ADATE	•	{·}	•	•	•	•	•	∅
FLIP	•	•	•	∅	◦	◦, ∅	•	∅
FFOIL	<i>c</i>	•	⊃	∅	◦	◦, ∅	◦	∅
GOLEM	•	{·}	•	∅	◦	◦	•	∅
IGOR I	•	{·}	∅	•	◦	∅	∅	∅
IGOR II	•	•	•	•	◦	∅	◦	∅
MAGH.	•	{·}	•	∅	•	•	•	◦

• unrestricted / conditional rules ◦ restricted / unconditional rules
 {·} singleton set ∅ empty set
c constants ⊃ built in predicates

Systems Description in the CCRS Framework

We will consider only FFOIL, GOLEM, FLIP, MAGIC-HASKELLER IGOR I and IGOR II. The prominent systems FFOIL and GOLEM shall provide a baseline, as representatives of ILP systems, against which the other systems can be compared. The others belong either to the most recent or currently to the most powerful IP systems and attest the current focus of research on IFLP and IFP.

PROGOL and DIALOGS-II have been excluded, because they make heavily use of background knowledge which goes beyond the notion the other systems have. DIALOGS-II as an interactive system collects much more evidence which is not expressible in I/O examples, because it virtually allows for Horn clauses in E^+ and E^- . Similarly the mode declarations of PROLOG. To allow for learning programs (unlike learning classifiers), mode declarations specify positions of recursive calls and correct data type decompositions would be necessary, which makes the task of IP uninteresting.

Table 1 summarises the classification of the mentioned systems into the CCRS framework and Table 2 the systems' restriction bias and the expressiveness of the conditions in a rule. The following paragraphs sketching the covering and search strategy, the search space and the preference bias.

ADATE as an evolutionary computation system employs search techniques that are inspired by basic biological principles of evolution, like for instance mutation and crossover. Starting from a trivial initial program, offsprings are generated which are tested against the I/O examples and rated using criteria as time and memory usage as preference bias. Only the "fittest" programs are developed further as the search progresses until eventually one program covers all I/O examples and it is aborted by the user. In this way, ADATE searches the space of all programs, in a subset of ML, globally, covering the examples via a generate and test.

FFOIL (Quinlan 1996) is an early representative of a functional top-down learning system. Starting with an empty set of conditions, it continues adding *gainful* literals to a rule until it covers no negative examples anymore. Which conditions to add is determined by a information-based heuristic called *foil gain*. It favours literals with a high information gain, i.e., which discriminate notably between positive and

negative evidence when added to a rule. All examples explained by this rule are removed from E^+ and another rule is learnt until E^+ is empty. If no candidate literals are gainful enough, all so called *determinate* literals are added. A determinate literal does not deteriorate the foil gain of the current rule, but introduces new variables. So FFOIL traverses the space of Horn clauses heuristically lead by the *foil gain* following a greedy sequential covering strategy.

FLIP (Hernández-Orallo et al. 1998) is a representative of the IFLP approach. It starts from all possible consistent restricted generalisations (CRG) deriveable from the positive example equations. A restricted generalisation (RG) is a generalisation without introducing new variables on the rhs of the equation. A RG is consistent if it does not cover any negative example when interpreted as a rewrite rule.

Informally, narrowing combines resolution from logic programming and term reduction from functional programming. FLIP uses its inverse, similar as inverse resolution, called inverse narrowing to solve the induction problem.

FLIP's core algorithm is based CRG and inverse narrowing which induce a space of hypothesis programs. It is searched heuristically using a combination of minimum description length and coverage of positive examples as preference bias, following a sequential covering strategy.

GOLEM (Muggleton and Feng 1990) uses a bottom-up, or example driven approach based on Plotkin's framework of relative least general generalisation (*rlgg*) (Plotkin 1971). This avoids searching a large hypothesis space for consistent hypothesis as, e.g, FFOIL, but rather constructs a unique clause covering a subset of the provided examples relative to the given background knowledge. However, such a search space explodes and makes search nearly intractable.

Therefore, to generate a single clause, GOLEM first randomly picks pairs of positive examples, computes their *rlggs* and chooses the one with the highest coverage, i.e., with the greatest number of positive examples covered. By randomly choosing additional examples and computing the *rlgg* of the clause and the new example, the clause is further generalised. This generalisation is repeated using the clause with the highest coverage until generalisation does not yield a higher coverage. To generate further clauses GOLEM uses the sequential covering approach. The preference bias is defined as the clause covering most of the positive and no negative examples in a lattice over clauses constructed by computing the *rlggs* of two examples.

MAGICHASKELLER (Katayama 2005) is a comparable new search-based synthesiser which generates HASKELL programs. Exploiting type-constraints, it searches the space of λ -expressions for the smallest program satisfying the user's specification. The expressions are created from user provided functions and data-type constructors via function composition, function application, and λ -abstraction (anonymous functions). The system's preference bias can be characterised as a breadth-first search over the length of the candidate programs guided by the type of the target function.

Therefore it prefers the smallest program constructable from the provided functions that satisfies the user's constraints.

IGOR I is a modern extension of the seminal THESYS system (Summers 1977) adopting its two-step approach. In a first step I/O examples are rewritten to traces which explain each output given the respective input based on a datatype theory. All traces are integrated into one conditional expression computing exactly the output for the inputs as given in the examples as a non-recursive program. In a second step, this initial program is generalised into recursive equations by searching for syntactic regularities in this term.

Synthesis is still restricted to structural problems, where only the structure of the arguments matters, but not their contents, such as in list reversing (and contrary to *member*). Nevertheless, the scope of synthesisable programs is considerably larger. For instance, tree-recursive functions and functions with hidden parameters can be induced. Most notably, programs consisting of a calling function and an arbitrary set of further recursive functions can be induced.

IGOR II is, contrarily to others, specialised to learn *recursive programs*. To do this reliably, partitioning of input examples, i.e., the introduction of patterns and predicates, and the synthesis of expressions computing the specified outputs, are strictly separated. Partitioning is done systematically and completely instead of randomly (GOLEM) or by a greedy search (FFOIL). All subsets of a partition are created in parallel, i.e., IGOR II follows a "simultaneous" covering approach. Also the search for expressions is complete, still remaining tractable even for relative complex programs because construction of hypotheses is data-driven. IGOR II combines analytical program synthesis with search.

Fewer case distinctions, most specific patterns, and fewer recursive calls or calls to background functions are preferred. Thus, the initial hypothesis is a single rule per target function. Initial rules are least general generalisations (*lggs*) (Plotkin 1971) of the example equations, i.e., patterns are *lggs* of the example inputs, *rhss* are *lggs* of the outputs w.r.t. the substitutions for the pattern, and conditions are empty. Successor hypotheses have to be computed, if unbound variables occur in *rhss*. Three ways of getting successor hypotheses are applied: (1) Partitioning of the inputs by replacing one pattern by a set of disjoint more specific patterns or by adding a predicate to the condition. (2) Replacing the rhs by a (recursive) call of a defined function, where finding the argument of the function call is treated as a new induction problem. (3) Replacing the rhs *subterms* in which unbound variables occur by a call to new subprograms. In cases (2) and (3) auxiliary functions are invented, abducting I/O-examples for them.

Forecast As far as one can generally already say, the "old" systems GOLEM and FFOIL are hampered by their greedy sequential covering strategy. Consequently, partial rules are never revised and lead to local optima, and thus losing dependencies between rules. This is especially the case with FFOIL learning predicates or finding a separate rule for the

Table 2: Overview of systems’ restriction bias

	$F(i_1, \dots, i_n)/lhs$	rhs	v_i/u_i
ADATE	$i_i \in \mathcal{X}_{\mathcal{T}}$	$\mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	ilc
FLIP	CRG of E^+	inverse narrowing of CRG of E^+	—
FFOIL	$i_i \in \mathcal{X}_{\mathcal{T}}$	$\mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}}) \cup \{true, false\}$	il
GOLEM	$i_i \in \mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	$\mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}}) \cup \{true, false\}$	ilc
IGOR I	$i_i \in \mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	$\mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	—
IGOR II	$i_i \in \mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	$\mathcal{T}_{\mathcal{C}}(\mathcal{X}_{\mathcal{T}})$	i
MAGH.	composition of functions from BK , higher-order via paramorphisms		

i if l let c case

base case, where the foil gain may be misleading. FFOIL is heavily biased towards constructing the next clause to cover the most frequent function value in the remaining tuples.

Where FFOIL has only very general lhss, GOLEM and FLIP try to be more flexible in discriminating the inputs there, but not effective enough. Random sampling is too unreliable for an optimal partition of the inputs, especially for more complex data structures or programs with many rules.

FLIP generates the lhss using the CRG on basis of common subterms on the lhs and rhs of the examples. Necessary function-carrying subterms on both sides may be generalised and the lhss may tend to be overly general. Also, neither overlap of the lhss is prohibited, nor are the rules ordered. Consequently, one input may be matched by several rules resulting in a wrong output. The rhs are constructed via inverse narrowing inducing a huge search space, so with increasing complexity of examples the search becomes more and more intractable when relying solely on heuristics.

MAGICHASKELLER is a promising example of including higher-order features into IP and shows how functions like *map* or *filter* can be applied effectively, when used advisedly, as some kind of program pattern or scheme. Nevertheless, MAGICHASKELLER and ADATE exhibits the usual pros and cons common to all search-based approaches: The more extensive the BK library, the more powerful the synthesised programs are, the greater is the search space and the longer are the runs. However, contrarily to GOLEM, it is not misled by partial solutions and shows again that only a complete search can be satisfactory for IP.

IGOR I and IGOR II will have problems were many examples are required (*mult/add* & *alldds*), but will be in other respects very fast.

Empirical Results

As problems we have chosen some of those occurring in the accordant papers and some to bring out the specific strengths and weaknesses. They have the usual semantics on lists: *multlast* replaces all elements with the last and *shiftr* makes a right-shift of all elements in a list. Therefore it is necessary to access the last element for further computations. Further functions are *lasts* which applies *last* on a list of lists, *isort* which is insertion-sort, *alldds* checks for odd numbers, and *weave* alternates elements from two lists into one. For *odd/even* and *mult/add* both functions need to be learnt at once. The functions in *odd/even* are mutually recursive and need more than two rules, *lasts*, *multlast*, *isort*, *reverse*,

Table 3: Systems’ runtimes on different problems in seconds

Problems	Systems						
	ADATE	FFOIL	FLIP	GOLEM	IGOR I	IGOR II	MAGH.
<i>lasts</i>	365.62	0.7 [⊥]	×	1.062	0.051	5.695	19.43
<i>last</i>	1.0	0.1	0.020	< 0.001	0.005	0.007	0.01
<i>member</i>	2.11	0.1 [⊥]	17.868	0.033	—	0.152	1.07
<i>odd/even</i>	—	< 0.1 [⊥]	0.130	—	—	0.019	—
<i>multlast</i>	5.69	< 0.1	448.900 [⊥]	< 0.001	0.331	0.023	0.30
<i>isort</i>	83.41	×	×	0.714	—	0.105	0.01
<i>reverse</i>	30.24	—	—	—	0.324	0.103	0.08
<i>weave</i>	27.11	0.2	134.240 [⊥]	0.266	0.001 [⊥]	0.022	⊙
<i>shiftr</i>	20.14	< 0.1 [⊥]	448.550 [⊥]	0.298	0.041	0.127	157.32
<i>mult/add</i>	—	8.1 [⊥]	×	—	—	⊙	—
<i>alldds</i>	466.86	0.1 [⊥]	×	0.016 [⊥]	0.015 [⊥]	⊙	×

— not tested × stack overflow ⊙ time out ⊥ wrong

mult/add, *alldds* suggest to use function invention, but only *reverse* is explicitly only solvable with. *lasts* and *alldds* also split up in more than two rules if no function invention is applied. To solve *member* pattern matching is required, because equality is not provided. The function *weave* is especially interesting, because it demands either for iterating over more than one argument resulting in more than one base case, or swapping the arguments at each recursive call.

Because FFOIL and GOLEM usually perform better with more examples, whereas FLIP, MAGICHASKELLER and IGOR II do better with less, each system got as much examples as necessary up to certain complexity, but then exhaustively, so no specific cherry-picking was allowed.

For synthesising *isort* all systems had a function to insert into a sorted list, and the predicate $<$ as background knowledge. FLIP needed an additional function *if* to relate the *insert* function with the $<$. For all systems except FLIP and MAGICHASKELLER the definition of the background knowledge was extensional. IGOR II was allowed to use variables and for GOLEM additionally the accordant negative examples were provided. MAGICHASKELLER had paramorphic functions to iterate over a data type in BK . Note that we did not test a system with a problem which it per se cannot solve due to its restriction bias. This is indicated with ‘—’ instead of a runtime. A timeout after ten minutes is indicated with ⊙. Table 3 shows the runtimes of the different systems on the example problems.

All tests have been conducted under Ubuntu 7.10 on a Intel Dual Core 2.33 GHz with 4GB memory. Following versions have been used: FLIP v0.7, FFOIL 1.0, GOLEM version of August 1992, the latest version of IGOR I, IGOR II version 2.2, ADATE version 0.50 and MAGICHASKELLER 0.8.3-1. The input files can be obtained under <http://www.cogsys.wiai.uni-bamberg.de/effalip/download.html>.

As the empirical results affirm the previous considerations, FFOIL fails with nearly all problems, and *multlast* was only solved with more examples. This can easily be explained with the greedy foil gain and a sequential cover-

ing strategy. Due to GOLEM's random sampling, the best result of ten runs have been chosen.

Long run times and the failures of FLIP testify for the intractable search space induced by the inverse narrowing operator. Wrong programs are due to overlapping lhss and its generalisation strategy of the inputs. Despite its randomisation, GOLEM overtrumps FLIP due to its capability of introducing `let`-expressions (cf. *multlast*). IGOR I and IGOR II need function invention to balance this weak-point.

On *reverse* and *isort* MAGICHASKELLER demonstrates the power of higher-order functions. Although it does not invent auxiliary functions, *reverse* was solved using its paramorphism over lists which provides some kind of accumulator. The paramorphisms are also the reason why MAGICHASKELLER fails with *weave*, since swapping the inputs with each recursive call does not fit in the schema induced by the paramorphism for lists.

These results showed, that the field of IP is not yet fully researched, and there are improvements discovered since the "golden times" of ILP and still to be discovered. Basically, ILP systems need a vast number of I/O examples which is usually impractical for a normal user to provide. Contrarily, IFP systems get along with much less examples but are still much more reliable in their results than ILP systems. Among the IFP it is significant analytic approaches rule out ADATE or MAGICHASKELLER on more complex problems where the search space increases. Also the ability of generically inventing functions is a big advantage.

Conclusions and Further Work

Based on a uniform description of some well-known IP systems and as result of our empirical evaluation of IP systems on a set of representative sample problems, we could show that the analytical approach of IGOR II is highly promising. IGOR II can induce a large scope of recursive programs, including mutual recursion using a straight-forward technique for function invention. Background knowledge can be provided in a natural way. As consequence of IGOR II's generalisation principle, induced programs are guaranteed to terminate and to be the least generalisations. Although construction of hypotheses is not restricted by some greedy heuristics, induction is highly time efficient. Furthermore, IGOR II works with minimal information provided by the user. It needs only a small set of positive I/O examples together with the data type specification of the target function and no further information such as schemes.

Due to the nature of specification by example, IP systems in general, cannot scale up to complex software development problems. However, integrating IP principles in the software engineering process might relieve developers from specifying or coding specific functions. Although IGOR II cannot tackle problems of complex size, it can tackle problems which are intellectually complex and therefore might offer support to inexperienced programmers.

Function invention for the outmost function without prior definition of the positions of recursive calls will be our greatest future challenge. Furthermore, we plan to include the introduction of `let`-expressions and higher-order functions (such as `map`, `reduce`, `filter`).

References

- Baader, F., and Nipkow, T. 1998. *Term Rewriting and All That*. United Kingdom: Cambridge University Press.
- Biermann, A. W.; Kodratoff, Y.; and Guiho, G. 1984. *Automatic Program Construction Techniques*. NY, Free Press.
- Flener, P., and Yilmaz, S. 1999. Inductive synthesis of recursive logic programs: Achievements and prospects. *J. Log. Program.* 41(2-3):141–195.
- Flener, P. 1996. Inductive logic program synthesis with Dialogs. In Muggleton, S., ed., *Proc. of the 6th International Workshop on ILP*, 28–51. Stockholm University
- Hernández-Orallo, J., and Ramírez-Quintana, M. J. 1998. Inverse narrowing for the induction of functional logic programs. In Freire-Nistal, et al., eds., *Joint Conference on Declarative Programming*, 379–392.
- Hofmann, M.; Kitzelmann, E.; and Schmid, U. 2008. Analysis and evaluation of inductive programming systems in a higher-order framework. In *31st German Conference on Artificial Intelligence*, LNAI. Springer-Verlag.
- Hofmann, M. 2007. *Automatic Construction of XSL Templates – An Inductive Programming Approach*. VDM Verlag, Saarbrücken.
- Katayama, S. 2005. Systematic search for lambda expressions. In *Trends in Functional Programming*, 111–126.
- Kitzelmann, E., and Schmid, U. 2006. Inductive synthesis of functional programs: An explanation based generalization approach. *J. of ML Research* 7:429–454.
- Kitzelmann, E. 2007. Data-driven induction of recursive functions from input/output-examples. In Kitzelmann, E., and Schmid, U., eds., *Proc. of the ECML/PKDD 2007 Workshop on Approaches and Applications of Inductive Programming*, 15–26.
- Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill Higher Education.
- Muggleton, S., and Feng, C. 1990. Efficient induction of logic programs. In *Proc. of the 1st Conference on Algorithmic Learning Theory*, 368–381. Ohmsma, Tokyo, Japan.
- Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):245–286.
- Olsson, R. J. 1995. Inductive functional programming using incremental program transformation. *Artificial Intelligence* 74(1):55–83.
- Plotkin, G. 1971. A further note on inductive generalization. In *Machine Intelligence*, vol. 6. Edinb. Univ. Press.
- Quinlan, J. R., and Cameron-Jones, R. M. 1993. FOIL: A midterm report. In *Machine Learning: ECML-93, Proc.*, vol. 667, 3–20. Springer-Verlag.
- Quinlan, J. R. 1996. Learning first-order definitions of functions. *Journal of AI Research* 5:139–161.
- Summers, P. D. 1977. A methodology for LISP program construction from examples. *Journal ACM* 24:162–175.
- Terese. 2003. *Term Rewriting Systems*, vol. 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Feature Markov Decision Processes

Marcus Hutter

RSISE @ ANU and SML @ NICTA

Canberra, ACT, 0200, Australia

marcus@hutter1.net www.hutter1.net

Abstract

General purpose intelligent learning agents cycle through (complex, non-MDP) sequences of observations, actions, and rewards. On the other hand, reinforcement learning is well-developed for small finite state Markov Decision Processes (MDPs). So far it is an art performed by human designers to extract the right state representation out of the bare observations, i.e. to reduce the agent setup to the MDP framework. Before we can think of mechanizing this search for suitable MDPs, we need a formal objective criterion. The main contribution of this article is to develop such a criterion. I also integrate the various parts into one learning algorithm. Extensions to more realistic dynamic Bayesian networks are developed in the companion article [Hut09].

Introduction

Background & motivation. Artificial General Intelligence (AGI) is concerned with designing agents that perform well in a wide range of environments [GP07; LH07]. Among the well-established “narrow” AI approaches, arguably Reinforcement Learning (RL) pursues most directly the same goal. RL considers the general agent-environment setup in which an agent interacts with an environment (acts and observes in cycles) and receives (occasional) rewards. The agent’s objective is to collect as much reward as possible. Most if not all AI problems can be formulated in this framework.

The simplest interesting environmental class consists of finite state fully observable Markov Decision Processes (MDPs) [Put94; SB98], which is reasonably well understood. Extensions to continuous states with (non)linear function approximation [SB98; Gor99], partial observability (POMDP) [KLC98; RPPCd08], structured MDPs (DBNs) [SDL07], and others have been considered, but the algorithms are much more brittle.

In any case, a lot of work is still left to the designer, namely to extract the right state representation (“features”) out of the bare observations. Even if *potentially* useful representations have been found, it is usually not clear which one will turn out to be better, except in situations where we already know a perfect model. Think of a mobile robot equipped with a camera plunged into an unknown environment. While we can imagine which image features are po-

tentially useful, we cannot know which ones will actually be useful.

Main contribution. Before we can think of mechanically searching for the “best” MDP representation, we need a formal objective criterion. Obviously, at any point in time, if we want the criterion to be effective it can only depend on the agent’s past experience. The main contribution of this article is to develop such a criterion. Reality is a non-ergodic partially observable uncertain unknown environment in which acquiring experience can be expensive. So we want/need to exploit the data (past experience) at hand optimally, cannot generate virtual samples since the model is not given (need to be learned itself), and there is no reset-option. In regression and classification, penalized maximum likelihood criteria [HTF01, Chp.7] have successfully been used for semi-parametric model selection. It is far from obvious how to apply them in RL. Ultimately we do not care about the observations but the rewards. The rewards depend on the states, but the states are arbitrary in the sense that they are model-dependent functions of the data. Indeed, our derived Cost function cannot be interpreted as a usual model+data code length.

Relation to other work. As partly detailed later, the suggested Φ MDP model could be regarded as a scaled-down practical instantiation of AIXI [Hut05; Hut07], as a way to side-step the open problem of learning POMDPs, as extending the idea of state-aggregation from planning (based on bi-simulation metrics [GDG03]) to RL (based on code length), as generalizing U-Tree [McC96] to arbitrary features, or as an alternative to PSRs [SLJ⁺03] for which proper learning algorithms have yet to be developed.

Notation. Throughout this article, \log denotes the binary logarithm, ϵ the empty string, and $\delta_{x,y} = \delta_{xy} = 1$ if $x = y$ and 0 else is the Kronecker symbol. I generally omit separating commas if no confusion arises, in particular in indices. For any x of suitable type (string, vector, set), I define string $x = x_{1:l} = x_1 \dots x_l$, sum $x_+ = \sum_j x_j$, union $x_* = \bigcup_j x_j$, and vector $\mathbf{x}_\bullet = (x_1, \dots, x_l)$, where j ranges over the full range $\{1, \dots, l\}$ and $l = |x|$ is the length or dimension or size of x . \hat{x} denotes an estimate of x . $P(\cdot)$ denotes a probability over states and rewards or parts thereof. I do not distinguish between random variables X and realizations x , and abbreviation $P(x) := P[X = x]$ never leads to confusion. More specifically, $m \in \mathbb{N}$ denotes the number of states, $i \in \{1, \dots, m\}$

any state index, $n \in \mathbb{N}$ the current time, and $t \in \{1, \dots, n\}$ any time. Further, due to space constraints at several places I gloss over initial conditions or special cases where inessential. Also $0 * \text{undefined} = 0 * \text{infinity} = 0$.

Feature Markov Decision Process (Φ MDP)

This section describes our formal setup. It consists of the agent-environment framework and maps Φ from observation-action-reward histories to MDP states. I call this arrangement “Feature MDP” or short Φ MDP.

Agent-environment setup. I consider the standard agent-environment setup [RN03] in which an *Agent* interacts with an *Environment*. The agent can choose from actions $a \in \mathcal{A}$ (e.g. limb movements) and the environment provides (regular) observations $o \in \mathcal{O}$ (e.g. camera images) and real-valued rewards $r \in \mathcal{R} \subseteq \mathbb{R}$ to the agent. The reward may be very scarce, e.g. just +1 (-1) for winning (losing) a chess game, and 0 at all other times [Hut05, Sec.6.3]. This happens in cycles $t = 1, 2, 3, \dots$: At time t , after observing o_t , the agent takes action a_t based on history $h_t := o_1 a_1 r_1 \dots o_{t-1} a_{t-1} r_{t-1} o_t$. Thereafter, the agent receives reward r_t . Then the next cycle $t+1$ starts. The agent’s objective is to maximize his long-term reward. Without much loss of generality, I assume that \mathcal{A} , \mathcal{O} , and \mathcal{R} are finite. Implicitly I assume \mathcal{A} to be small, while \mathcal{O} may be huge.

The agent and environment may be viewed as a pair or triple of interlocking functions of the history $\mathcal{H} := (\mathcal{O} \times \mathcal{A} \times \mathcal{R})^* \times \mathcal{O}$:

$$\begin{aligned} \text{Env} : \mathcal{H} \times \mathcal{A} \times \mathcal{R} &\rightsquigarrow \mathcal{O}, & o_n &= \text{Env}(h_{n-1} a_{n-1} r_{n-1}), \\ \text{Agent} : \mathcal{H} &\rightsquigarrow \mathcal{A}, & a_n &= \text{Agent}(h_n), \\ \text{Env} : \mathcal{H} \times \mathcal{A} &\rightsquigarrow \mathcal{R}, & r_n &= \text{Env}(h_n a_n). \end{aligned}$$

where \rightsquigarrow indicates that mappings \rightarrow might be stochastic.

The goal of AI is to design agents that achieve high (expected) reward over the agent’s lifetime.

(Un)known environments. For known $\text{Env}()$, finding the reward maximizing agent is a well-defined and formally solvable problem [Hut05, Chp.4], with computational efficiency being the “only” matter of concern. For most real-world AI problems $\text{Env}()$ is at best partially known.

Narrow AI considers the case where function $\text{Env}()$ is either known (like in blocks world), or essentially known (like in chess, where one can safely model the opponent as a perfect minimax player), or $\text{Env}()$ belongs to a relatively small class of environments (e.g. traffic control).

The goal of AGI is to design agents that perform well in a large range of environments [LH07], i.e. achieve high reward over their lifetime with as little as possible assumptions about $\text{Env}()$. A minimal necessary assumption is that the environment possesses *some* structure or pattern.

From real-life experience (and from the examples below) we know that usually we do not need to know the complete history of events in order to determine (sufficiently well) what will happen next and to be able to perform well. Let $\Phi(h)$ be such a “useful” summary of history h .

Examples. In full-information *games* (like chess) with static opponent, it is sufficient to know the current state of the game (board configuration) to play well (the history

plays no role), hence $\Phi(h_t) = o_t$ is a sufficient summary (Markov condition). Classical *physics* is essentially predictable from position and velocity of objects at a single time, or equivalently from the locations at two consecutive times, hence $\Phi(h_t) = o_t o_{t-1}$ is a sufficient summary (2nd order Markov). For *i.i.d. processes* of unknown probability (e.g. clinical trials \simeq Bandits), the frequency of observations $\Phi(h_n) = (\sum_{t=1}^n \delta_{o_t o})_{o \in \mathcal{O}}$ is a sufficient statistic. In a *POMDP planning* problem, the so-called belief vector at time t can be written down explicitly as some function of the complete history h_t (by integrating out the hidden states). $\Phi(h_t)$ could be chosen as (a discretized version of) this belief vector, showing that Φ MDP generalizes POMDPs. Obviously, the *identity* $\Phi(h) = h$ is always sufficient but not very useful, since $\text{Env}()$ as a function of \mathcal{H} is hard to impose to “learn”.

This suggests to look for Φ with small codomain, which allow to learn/estimate/approximate Env by $\widehat{\text{Env}}$ such that $o_t \approx \widehat{\text{Env}}(\Phi(h_{t-1}))$ for $t = 1 \dots n$.

Example. Consider a robot equipped with a camera, i.e. o is a pixel image. Computer vision algorithms usually extract a set of features from o_{t-1} (or h_{t-1}), from low-level patterns to high-level objects with their spatial relation. Neither is it possible nor necessary to make a precise prediction of o_t from summary $\Phi(h_{t-1})$. An approximate prediction must and will do. The difficulty is that the similarity measure “ \approx ” needs to be context dependent. Minor image nuances are irrelevant when driving a car, but when buying a painting it makes a huge difference in price whether it’s an original or a copy. Essentially only a bijection Φ would be able to extract *all* *potentially* interesting features, but such a Φ defeats its original purpose.

From histories to states. It is of utmost importance to properly formalize the meaning of “ \approx ” in a general, domain-independent way. Let $s_t := \Phi(h_t)$ summarize all relevant information in history h_t . I call s a state or feature (vector) of h . “Relevant” means that the future is predictable from s_t (and a_t) alone, and that the relevant future is coded in $s_{t+1} s_{t+2} \dots$. So we pass from the complete (and known) history $o_1 a_1 r_1 \dots o_n a_n r_n$ to a “compressed” history $s a r_{1:n} \equiv s_1 a_1 r_1 \dots s_n a_n r_n$ and seek Φ such that s_{t+1} is (approximately a stochastic) function of s_t (and a_t). Since the goal of the agent is to maximize his rewards, the rewards r_t are always relevant, so they (have to) stay untouched (this will become clearer below).

The Φ MDP. The structure derived above is a classical Markov Decision Process (MDP), but the primary question I ask is not the usual one of finding the value function or best action or comparing different models of a given state sequence. I ask how well can the state-action-reward sequence generated by Φ be modeled as an MDP compared to other sequences resulting from different Φ .

Φ MDP Coding and Evaluation

I first review optimal codes and model selection methods for i.i.d. sequences, subsequently adapt them to our situation, and show that they are suitable in our context. I state my Cost function for Φ and the Φ selection principle.

I.i.d. processes. Consider i.i.d. $x_1 \dots x_n \in \mathcal{X}^n$ for finite $\mathcal{X} = \{1, \dots, m\}$. For known $\theta_i = \mathbb{P}[x_t = i]$ we have $\mathbb{P}(x_{1:n}|\theta) = \theta_{x_1} \dots \theta_{x_n}$. It is well-known that there exists a code (e.g. arithmetic or Shannon-Fano) for $x_{1:n}$ of length $-\log \mathbb{P}(x_{1:n}|\theta)$, which is asymptotically optimal with probability one.

For unknown θ we may use a frequency estimate $\hat{\theta}_i = n_i/n$, where $n_i = |\{t : x_t = i\}|$. Then $-\log \mathbb{P}(x_{1:n}|\hat{\theta}) = n H(\hat{\theta})$, where $H(\hat{\theta}) := -\sum_{i=1}^m \hat{\theta}_i \log \hat{\theta}_i$ is the Entropy of $\hat{\theta}$ ($0 \log 0 := 0 =: 0 \log \frac{0}{0}$). We also need to code (n_i) , which naively needs $\log n$ bits for each i . One can show that it is sufficient to code each $\hat{\theta}_i$ to accuracy $O(1/\sqrt{n})$, which requires only $\frac{1}{2} \log n + O(1)$ bits each. Hence the overall code length of $x_{1:n}$ for unknown frequencies is

$$\text{CL}(x_{1:n}) = \text{CL}(\mathbf{n}) := n H(\mathbf{n}/n) + \frac{m'-1}{2} \log n \quad (1)$$

for $n > 0$ and 0 else, where $\mathbf{n} = (n_1, \dots, n_m)$ and $n = n_+ = n_1 + \dots + n_m$ and $m' = |\{i : n_i > 0\}| \leq m$ is the number of non-empty categories. The code is optimal (within $+O(1)$) for all i.i.d. sources. It can be rigorously derived from many principles: MDL, MML, combinatorial, incremental, and Bayesian [Grü07].

In the following I will ignore the $O(1)$ terms and refer to (1) simply as *the* code length. Note that $x_{1:n}$ is coded exactly (lossless). Similarly (see MDP below) sampling models more complex than i.i.d. may be considered, and the one that leads to the shortest code is selected as the best model [Grü07].

MDP definitions. Recall that a sequence $sa r_{1:n}$ is said to be sampled from an MDP $(\mathcal{S}, \mathcal{A}, T, R)$ iff the probability of s_t only depends on s_{t-1} and a_{t-1} ; and r_t only on s_{t-1} , a_{t-1} , and s_t . That is, $\mathbb{P}(s_t | h_{t-1} a_{t-1}) = \mathbb{P}(s_t | s_{t-1}, a_{t-1}) =: T_{s_{t-1} s_t}^{a_{t-1}}$ and $\mathbb{P}(r_t | h_t) = \mathbb{P}(r_t | s_{t-1}, a_{t-1}, s_t) =: R_{s_{t-1} s_t}^{a_{t-1} r_t}$. For simplicity of exposition I assume a deterministic dependence of r_t on s_t only, i.e. $r_t = R_{s_t}$. In our case, we can identify the state-space \mathcal{S} with the states s_1, \dots, s_n “observed” so far. Hence $\mathcal{S} = \{s^1, \dots, s^m\}$ is finite and typically $m \ll n$, i.e. states repeat. Let $s \xrightarrow{a} s'(r')$ be shorthand for “action a in state s resulted in state s' (reward r')”. Let $\mathcal{T}_{ss'}^{ar'} := \{t \leq n : s_{t-1} = s, a_{t-1} = a, s_t = s', r_t = r'\}$ be the set of times $t-1$ at which $s \xrightarrow{a} s'$, and $n_{ss'}^{ar'} := |\mathcal{T}_{ss'}^{ar'}|$ their number ($n_{++}^+ = n$).

Coding MDP sequences. For some fixed s and a , consider the subsequence $s_{t_1} \dots s_{t_{n'}}$ of states reached from s via a ($s \xrightarrow{a} s_{t_i}$), i.e. $\{t_1, \dots, t_{n'}\} = \mathcal{T}_{s s'}^{a*}$, where $n' = n_{s s'}^{a+}$. By definition of an MDP, this sequence is i.i.d. with s' occurring $n_{s s'}^{a+}$ times. By (1) we can code this sequence in $\text{CL}(n')$ bits. The whole sequence $s_{1:n}$ consists of $|\mathcal{S} \times \mathcal{A}|$ i.i.d. sequences, one for each $(s, a) \in \mathcal{S} \times \mathcal{A}$. We can join their codes and get a total code length

$$\text{CL}(s_{1:n} | a_{1:n}) = \sum_{s, a} \text{CL}(n_{s \bullet}^{a+}) \quad (2)$$

Similarly to the states we code the rewards. There are different “standard” reward models. I consider only the simplest case of a small discrete reward set \mathcal{R} like $\{0, 1\}$ or $\{-1, 0, +1\}$ here and defer generalizations to \mathbb{R} and a discussion of variants to the Φ DBN model [Hut09]. By the

MDP assumption, for each state s' , the rewards at times $\mathcal{T}_{+s'}^{**}$ are i.i.d. Hence they can be coded in

$$\text{CL}(r_{1:n} | s_{1:n}, a_{1:n}) = \sum_{s'} \text{CL}(n_{+s'}^{**}) \quad (3)$$

bits. I have been careful to assign zero code length to non-occurring transitions $s \xrightarrow{a} s' r'$ so that large but sparse MDPs don't get penalized too much.

Reward \leftrightarrow state trade-off. Note that the code for r depends on s . Instead we may interpret the construction as follows: Ultimately we/the agent cares about the reward, so we want to measure how well we can predict the rewards, which we do with (3). But this code depends on s , so we need a code for s too, which is (2). To see that we need both parts consider two extremes.

A simplistic state transition model (small $|\mathcal{S}|$) results in a short code for s . For instance, for $|\mathcal{S}| = 1$, nothing needs to be coded and (2) is identically zero. But this obscures potential structure in the reward sequence, leading to a long code for r .

On the other hand, the more detailed the state transition model (large $|\mathcal{S}|$) the easier it is to predict and hence compress r . But a large model is hard to learn, i.e. the code for s will be large. For instance for $\Phi(h) = h$, no state repeats and the frequency-based coding breaks down.

Φ selection principle. Let us define the *Cost* of $\Phi : \mathcal{H} \rightarrow \mathcal{S}$ on h_n as the length of the Φ MDP code for sr given a:

$$\text{Cost}(\Phi | h_n) := \text{CL}(s_{1:n} | a_{1:n}) + \text{CL}(r_{1:n} | s_{1:n}, a_{1:n}), \quad (4)$$

where $s_t = \Phi(h_t)$ and $h_t = oar_{1:t-1} o_t$

The discussion above suggests that the minimum of the joint code length, i.e. the *Cost*, is attained for a Φ that keeps all and only relevant information for predicting rewards. Such a Φ may be regarded as best explaining the rewards. So we are looking for a Φ of minimal cost:

$$\Phi^{best} := \arg \min_{\Phi} \{\text{Cost}(\Phi | h_n)\} \quad (5)$$

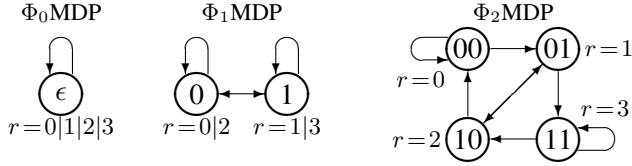
The state sequence generated by Φ^{best} (or approximations thereof) will usually only be approximately MDP. While $\text{Cost}(\Phi | h)$ is an optimal code only for MDP sequences, it still yields good codes for approximate MDP sequences. Indeed, Φ^{best} balances closeness to MDP with simplicity. The primary purpose of the simplicity bias is *not* computational tractability, but generalization ability [LH07; Hut05].

A Tiny Example

The purpose of the tiny example in this section is to provide enough insight into how and why Φ MDP works to convince the reader that our Φ selection principle is reasonable. Consider binary observation space $\mathcal{O} = \{0, 1\}$, quaternary reward space $\mathcal{R} = \{0, 1, 2, 3\}$, and a single action $\mathcal{A} = \{0\}$. Observations o_t are independent fair coin flips, i.e. Bernoulli($\frac{1}{2}$), and reward $r_t = 2o_{t-1} + o_t$ a deterministic function of the two most recent observations.

Considered features. As features Φ I consider $\Phi_k : \mathcal{H} \rightarrow \mathcal{O}^k$ with $\Phi_k(h_t) = o_{t-k+1} \dots o_t$ for various $k = 0, 1, 2, \dots$ which regard the last k observations as “relevant”. Intuitively Φ_2

is the best observation summary, which I confirm below. The state space $\mathcal{S} = \{0,1\}^k$ (for sufficiently large n). The Φ MDPs for $k=0,1,2$ are as follows.



Φ_2 MDP with all non-zero transition probabilities being 50% is an exact representation of our data source. The missing arrow (directions) are due to the fact that $s = o_{t-1}o_t$ can only lead to $s' = o'_t o'_{t+1}$ for which $o'_t = o_t$. Note that Φ MDP does not “know” this and has to learn the (non)zero transition probabilities. Each state has two successor states with equal probability, hence generates (see previous paragraph) a Bernoulli($\frac{1}{2}$) state subsequence and a constant reward sequence, since the reward can be computed from the state = last two observations. Asymptotically, all four states occur equally often, hence the sequences have approximately the same length $n/4$.

In general, if s (and similarly r) consists of $x \in \mathcal{N}$ i.i.d. subsequences of equal length n/x over $y \in \mathcal{N}$ symbols, the code length (2) (and similarly (3)) is

$$\begin{aligned} \text{CL}(s|\mathbf{a}; x_y) &= n \log y + x \frac{|\mathcal{S}|-1}{2} \log \frac{n}{x}, \\ \text{CL}(r|\mathbf{s}, \mathbf{a}; x_y) &= n \log y + x \frac{|\mathcal{R}|-1}{2} \log \frac{n}{x} \end{aligned}$$

where the extra argument x_y just indicates the sequence property. So for Φ_2 MDP we get

$\text{CL}(s|\mathbf{a}; 4_2) = n + 6 \log \frac{n}{4}$ and $\text{CL}(r|\mathbf{s}, \mathbf{a}; 4_1) = 6 \log \frac{n}{4}$. The log-terms reflect the required memory to code (or the time to learn) the MDP structure and probabilities. Since each state has only 2 realized/possible successors, we need n bits to code the state sequence. The reward is a deterministic function of the state, hence needs no memory to code given s .

The Φ_0 MDP throws away all observations (left figure above), hence $\text{CL}(s|\mathbf{a}; 1_1) = 0$. While the reward sequence is *not* i.i.d. (e.g. $r_{t+1} = 3$ cannot follow $r_t = 0$), Φ_0 MDP has no choice regarding them as i.i.d., resulting in $\text{CL}(s|\mathbf{a}; 1_4) = 2n + \frac{3}{2} \log n$.

The Φ_1 MDP model is an interesting compromise (middle figure above). The state allows a partial prediction of the reward: State 0 allows rewards 0 and 2; state 1 allows rewards 1 and 3. Each of the two states creates a Bernoulli($\frac{1}{2}$) state successor subsequence and a binary reward sequence, wrongly presumed to be Bernoulli($\frac{1}{2}$). Hence $\text{CL}(s|\mathbf{a}; 2_2) = n + \log \frac{n}{2}$ and $\text{CL}(r|\mathbf{s}, \mathbf{a}; 2_2) = n + 3 \log \frac{n}{2}$.

Summary. The following table summarizes the results for general $k=0,1,2$ and beyond:

$\text{Cost}(\Phi_0 h)$	$\text{Cost}(\Phi_1 h)$	$\text{Cost}(\Phi_2 h)$	$\text{Cost}(\Phi_{k \geq 2} h)$
$2n + \frac{3}{2} \log n$	$2n + 4 \log \frac{n}{2}$	$n + 12 \log \frac{n}{4}$	$n + \frac{2^k + 2}{2^{k-1}} \log \frac{n}{2^k}$

For large n , Φ_2 results in the shortest code, as anticipated. The “approximate” model Φ_1 is just not good enough to beat the vacuous model Φ_0 , but in more realistic examples some approximate model usually has the shortest code. In [Hut09] I show on a more complex example how Φ^{best} will store long-term information in a POMDP environment.

Cost(Φ) Minimization

I have reduced the reinforcement learning problem to a formal Φ -optimization problem. I briefly explain what we have gained by this reduction, and provide some general information about problem representations, stochastic search, and Φ neighborhoods. Finally I present a simplistic but concrete algorithm for searching context tree MDPs.

Φ search. I now discuss how to find good summaries Φ . The introduced generic cost function $\text{Cost}(\Phi|h_n)$, based on only the known history h_n , makes this a well-defined task that is completely decoupled from the complex (ill-defined) reinforcement learning objective. This reduction should not be under-estimated. We can employ a wide range of optimizers and do not even have to worry about overfitting. The most challenging task is to come up with creative algorithms proposing Φ 's.

There are many optimization methods: Most of them are search-based: random, blind, informed, adaptive, local, global, population based, exhaustive, heuristic, and other search methods [AL97]. Most are or can be adapted to the structure of the objective function, here $\text{Cost}(\cdot|h_n)$. Some exploit the structure more directly (e.g. gradient methods for convex functions). Only in very simple cases can the minimum be found analytically (without search).

General maps Φ can be represented by/as programs for which variants of Levin search [Sch04; Hut05] and genetic programming are the major search algorithms. Decision trees/lists/grids are also quite powerful, especially rule-based ones in which logical expressions recursively divide domain \mathcal{H} into “true/false” regions [San08] that can be identified with different states.

Φ neighborhood relation. Most search algorithms require the specification of a neighborhood relation or distance between candidate Φ . A natural “minimal” change of Φ is splitting and merging states (state refinement and coarsening). Let Φ' split some state $s^a \in \mathcal{S}$ of Φ into $s^b, s^c \notin \mathcal{S}$

$$\Phi'(h) := \begin{cases} \Phi(h) & \text{if } \Phi(h) \neq s^a \\ s^b \text{ or } s^c & \text{if } \Phi(h) = s^a \end{cases}$$

where the histories in state s^a are distributed among s^b and s^c according to some splitting rule (e.g. randomly). The new state space is $\mathcal{S}' = \mathcal{S} \setminus \{s^a\} \cup \{s^b, s^c\}$. Similarly Φ' merges states $s^b, s^c \in \mathcal{S}$ into $s^a \notin \mathcal{S}$ if

$$\Phi'(h) := \begin{cases} \phi(h) & \text{if } \Phi(h) \neq s^a \\ s^a & \text{if } \Phi(h) = s^b \text{ or } s^c \end{cases}$$

where $\mathcal{S}' = \mathcal{S} \setminus \{s^b, s^c\} \cup \{s^a\}$. We can regard Φ' as being a neighbor of or similar to Φ .

Stochastic Φ search. Stochastic search is the method of choice for high-dimensional unstructured problems. Monte Carlo methods can actually be highly effective, despite their simplicity [Liu02]. The general idea is to randomly choose a neighbor Φ' of Φ and replace Φ by Φ' if it is better, i.e. has smaller Cost. Even if $\text{Cost}(\Phi'|h) > \text{Cost}(\Phi|h)$ we may keep Φ' , but only with some (in the cost difference exponentially) small probability. Simulated annealing is a version which minimizes $\text{Cost}(\Phi|h)$. Apparently, Φ of small cost are (much) more likely to occur than high cost Φ .

Context tree example. The Φ_k in the example of the previous section depended on the last k observations. Let us generalize this to a context dependent variable length: Consider a finite complete suffix free set of strings (= prefix tree of reversed strings) $\mathcal{S} \subset \mathcal{O}^*$ as our state space (e.g. $\mathcal{S} = \{0, 01, 011, 111\}$ for binary \mathcal{O}), and define $\Phi_{\mathcal{S}}(h_n) := s$ iff $o_{n-|s|+1:n} = s \in \mathcal{S}$, i.e. s is the part of the history regarded as relevant. State splitting and merging works as follows: For binary \mathcal{O} , if history part $s \in \mathcal{S}$ of h_n is deemed too short, we replace s by $0s$ and $1s$ in \mathcal{S} , i.e. $\mathcal{S}' = \mathcal{S} \setminus \{s\} \cup \{0s, 1s\}$. If histories $1s, 0s \in \mathcal{S}$ are deemed too long, we replace them by s , i.e. $\mathcal{S}' = \mathcal{S} \setminus \{0s, 1s\} \cup \{s\}$. Large \mathcal{O} might be coded binary and then treated similarly. The idea of using suffix trees as state space is from [McC96]. For small \mathcal{O} we have the following simple Φ -optimizer:

Φ Improve($\Phi_{\mathcal{S}}, h_n$)

- [Randomly choose a state $s \in \mathcal{S}$;
- Let p and q be uniform random numbers in $[0, 1]$;
- if $(p > 1/2)$ then split s i.e. $\mathcal{S}' = \mathcal{S} \setminus \{s\} \cup \{os : o \in \mathcal{O}\}$
- else if $\{os : o \in \mathcal{O}\} \subseteq \mathcal{S}$
- then merge them, i.e. $\mathcal{S}' = \mathcal{S} \setminus \{os : o \in \mathcal{O}\} \cup \{s\}$;
- if $(\text{Cost}(\Phi_{\mathcal{S}'}|h_n) - \text{Cost}(\Phi_{\mathcal{S}}|h_n) > \log(q))$ then $\mathcal{S} := \mathcal{S}'$;
- [**return** ($\Phi_{\mathcal{S}}$);

Exploration & Exploitation

Having obtained a good estimate $\hat{\Phi}$ of Φ^{best} in the previous section, we can/must now determine a good action for our agent. For a finite MDP with known transition probabilities, finding the optimal action is routine. For estimated probabilities we run into the infamous exploration-exploitation problem, for which promising approximate solutions have recently been suggested [SL08]. At the end of this section I present the overall algorithm for our Φ MDP agent.

Optimal actions for known MDPs. For a known finite MDP $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, the maximal achievable (“optimal”) expected future discounted reward sum, called (Q) Value (of action a) in state s , satisfies the following (Bellman) equations [SB98]

$$Q_s^{*a} = \sum_{s'} T_{ss'}^a [R_{ss'}^a + \gamma V_{s'}^*] \quad \text{and} \quad V_s^* = \max_a Q_s^{*a} \quad (6)$$

where $0 < \gamma < 1$ is a discount parameter, typically close to 1. See [Hut05, Sec.5.7] for proper choices. The equations can be solved in polynomial time by a simple iteration process or various other methods [Put94]. After observing o_{n+1} , the optimal next action is

$$a_{n+1} := \arg \max_a Q_{s_{n+1}}^{*a}, \quad \text{where} \quad s_{n+1} = \Phi(h_{n+1}) \quad (7)$$

Estimating the MDP. We can estimate the transition probability T by

$$\hat{T}_{ss'}^a := \frac{n_{ss'}^{a+}}{n_{s+}^{a+}} \quad \text{if} \quad n_{s+}^{a+} > 0 \quad \text{and} \quad 0 \quad \text{else.} \quad (8)$$

It is easy to see that the Shannon-Fano code of $s_{1:n}$ based on $P_{\hat{T}}(s_{1:n}|a_{1:n}) = \prod_{t=1}^n \hat{T}_{s_{t-1}s_t}^{a_{t-1}}$ plus the code of the non-zero transition probabilities $\hat{T}_{ss'}^a > 0$ to relevant accuracy $O(1/\sqrt{n_{s+}^{a+}})$ has length (2), i.e. the frequency estimate (8)

is consistent with the attributed code length. The expected reward can be estimated as

$$\hat{R}_{ss'}^a := \sum_{r' \in \mathcal{R}} \hat{R}_{ss'}^{ar'}, \quad \hat{R}_{ss'}^{ar'} := \frac{n_{ss'}^{ar'}}{n_{ss'}^{a+}} \quad (9)$$

Exploration. Simply replacing T and R in (6) and (7) by their estimates (8) and (9) can lead to very poor behavior, since parts of the state space may never be explored, causing the estimates to stay poor.

Estimate \hat{T} improves with increasing n_{s+}^{a+} , which can (only) be ensured by trying all actions a in all states s sufficiently often. But the greedy policy above has no incentive to explore, which may cause the agent to perform very poorly: The agent stays with what he *believes* to be optimal without trying to solidify his belief. Trading off exploration versus exploitation optimally is computationally intractable [Hut05; PVHR06; RP08] in all but extremely simple cases (e.g. Bandits). Recently, polynomially optimal algorithms (Rmax, E3, OIM) have been invented [KS98; SL08]: An agent is more explorative if he expects a high reward in the unexplored regions. We can “deceive” the agent to believe this by adding another “absorbing” high-reward state $s^e \in \mathcal{S}$, not in the range of $\Phi(h)$, i.e. never observed. Henceforth, \mathcal{S} denotes the extended state space. For instance $+$ in (8) now includes s^e . We set

$$n_{ss^e}^a = 1, \quad n_{s^e s}^a = \delta_{s^e s}, \quad R_{ss^e}^a = R_{max}^e \quad (10)$$

for all s, a , where exploration bonus R_{max}^e is polynomially (in $(1-\gamma)^{-1}$ and $|\mathcal{S} \times \mathcal{A}|$) larger than $\max \mathcal{R}$ [SL08].

Now compute the agent’s action by (6)-(9) but for the extended \mathcal{S} . The optimal policy p^* tries to find a chain of actions and states that likely leads to the high reward absorbing state s^e . Transition $\hat{T}_{ss^e}^a = 1/n_{s+}^a$ is only “large” for small n_{s+}^a , hence p^* has a bias towards unexplored (state,action) regions. It can be shown that this algorithm makes only a polynomial number of sub-optimal actions.

The overall algorithm for our Φ MDP agent is as follows.

Φ MDP-Agent(\mathcal{A}, \mathcal{R})

- [Initialize $\Phi \equiv \epsilon$; $\mathcal{S} = \{\epsilon\}$; $h_0 = a_0 = r_0 = \epsilon$;
- for $n = 0, 1, 2, 3, \dots$
- [Choose e.g. $\gamma = 1 - 1/(n+1)$;
- Set $R_{max}^e = \text{Polynomial}((1-\gamma)^{-1}, |\mathcal{S} \times \mathcal{A}|) \cdot \max \mathcal{R}$;
- While waiting for o_{n+1} { $\Phi := \Phi$ Improve(Φ, h_n)};
- Observe o_{n+1} ; $h_{n+1} = h_n a_n r_n o_{n+1}$;
- $s_{n+1} := \Phi(h_{n+1})$; $\mathcal{S} := \mathcal{S} \cup \{s_{n+1}\}$;
- Compute action a_{n+1} from Equations (6)-(10);
- Output action a_{n+1} ;
-] [Observe reward r_{n+1} ;

Improved Cost Function

As discussed, we ultimately only care about (modeling) the rewards, but this endeavor required introducing and coding states. The resulted $\text{Cost}(\Phi|h)$ function is a code length of not only the rewards but also the “spurious” states. This likely leads to a too strong penalty of models Φ with large state spaces \mathcal{S} . The proper Bayesian formulation developed in this section allows to “integrate” out the states. This leads

to a code for the rewards only, which better trades off accuracy of the reward model and state space size.

For an MDP with transition and reward probabilities $T_{ss'}^a$ and $R_{ss'}^{ar'}$, the probabilities of the state and reward sequences are

$$P(s_{1:n}|a_{1:n}) = \prod_{t=1}^n T_{s_{t-1}s_t}^{a_{t-1}}, \quad P(r_{1:n}|s_{1:n}a_{1:n}) = \prod_{t=1}^n R_{s_{t-1}s_t}^{a_{t-1}r_t}$$

The probability of $\mathbf{r}|\mathbf{a}$ can be obtained by taking the product and marginalizing \mathbf{s} :

$$P_U(r_{1:n}|a_{1:n}) = \sum_{s_{1:n}} \prod_{t=1}^n U_{s_{t-1}s_t}^{a_{t-1}r_t} = \sum_{s_n} [U^{a_0r_1} \dots U^{a_{n-1}r_n}]_{s_0s_n}$$

where for each $a \in \mathcal{A}$ and $r' \in \mathcal{R}$, matrix $U^{ar'} \in \mathbb{R}^{m \times m}$ is defined as $[U^{ar'}]_{ss'} \equiv U_{ss'}^{ar'} := T_{ss'}^a R_{ss'}^{ar'}$. The right n -fold matrix product can be evaluated in time $O(m^2n)$. This shows that \mathbf{r} given \mathbf{a} and U can be coded in $-\log P_U$ bits. The unknown U needs to be estimated, e.g. by the relative frequency $\hat{U}_{ss'}^{ar'} := n_{ss'}^{ar'} / n_{s+}^{a+}$. The $M := m(m-1)|\mathcal{A}|(|\mathcal{R}|-1)$ (independent) elements of \hat{U} can be coded to sufficient accuracy in $\frac{1}{2}M \log n$ bits. Together this leads to a code for $\mathbf{r}|\mathbf{a}$ of length

$$\text{ICost}(\Phi|h_n) := -\log P_U(r_{1:n}|a_{1:n}) + \frac{1}{2}M \log n \quad (11)$$

In practice, M can and should be chosen smaller like done in the original Cost function, where we have used a restrictive model for R and considered only non-zero transitions in T .

Conclusion

I have developed a formal criterion for evaluating and selecting good “feature” maps Φ from histories to states and presented the feature reinforcement learning algorithm Φ MDP-Agent(). The computational flow is $h \rightsquigarrow \Phi \rightsquigarrow (\hat{T}, \hat{R}) \rightsquigarrow (\hat{V}, \hat{Q}) \rightsquigarrow a$. The algorithm can easily and significantly be accelerated: Local search algorithms produce sequences of “similar” Φ , which naturally suggests to compute/update $\text{Cost}(\Phi|h)$ and the value function V incrementally. The primary purpose of this work was to introduce and explore Φ -selection on the conveniently simple (but impractical) unstructured finite MDPs. The results of this work set the stage for the more powerful Φ DBN model developed in the companion article [Hut09] based on Dynamic Bayesian Networks. The major open problems are to develop smart Φ generation and smart stochastic search algorithms for Φ^{best} , and to determine whether minimizing (11) is the right criterion.

References

[AL97] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley-Interscience, Chichester, England, 1997.

[GDG03] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1–2):163–223, 2003.

[Gor99] G. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1999.

[GP07] B. Goertzel and C. Pennachin, editors. *Artificial General Intelligence*. Springer, 2007.

[Grü07] P. D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, Cambridge, 2007.

[HTF01] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[Hut05] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005. 300 pages, <http://www.hutter1.net/ai/uaibook.htm>.

[Hut07] M. Hutter. Universal algorithmic intelligence: A mathematical top→down approach. In *Artificial General Intelligence*, pages 227–290. Springer, Berlin, 2007.

[Hut09] M. Hutter. Feature dynamic Bayesian networks. In *Artificial General Intelligence (AGI'09)*. Atlantis Press, 2009.

[KLC98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[KS98] M. J. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA, 1998.

[LH07] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds & Machines*, 17(4):391–444, 2007.

[Liu02] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2002.

[McC96] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1996.

[Put94] M. L. Puterman. *Markov Decision Processes — Discrete Stochastic Dynamic Programming*. Wiley, New York, NY, 1994.

[PVHR06] P. Poupart, N. A. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proc. 23rd International Conf. on Machine Learning (ICML'06)*, volume 148, pages 697–704, Pittsburgh, PA, 2006. ACM.

[RN03] S. J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[RP08] S. Ross and J. Pineau. Model-based Bayesian reinforcement learning in large structured domains. In *Proc. 24th Conference in Uncertainty in Artificial Intelligence (UAI'08)*, pages 476–483, Helsinki, 2008. AUAI Press.

[RPPCd08] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008(32):663–704, 2008.

[San08] S. Sanner. *First-Order Decision-Theoretic Planning in Structured Relational Environments*. PhD thesis, Department of Computer Science, University of Toronto, 2008.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sch04] J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.

[SDL07] A. L. Strehl, C. Diuk, and M. L. Littman. Efficient structure learning in factored-state MDPs. In *Proc. 27th AAAI Conference on Artificial Intelligence*, pages 645–650, Vancouver, BC, 2007. AAAI Press.

[SL08] I. Szita and A. Lőrincz. The many faces of optimism: a unifying approach. In *Proc. 12th International Conference (ICML 2008)*, volume 307, Helsinki, Finland, June 2008.

[SLJ+03] S. Singh, M. Littman, N. Jong, D. Pardoe, and P. Stone. Learning predictive state representations. In *Proc. 20th International Conference on Machine Learning (ICML'03)*, pages 712–719, 2003.

Feature Dynamic Bayesian Networks

Marcus Hutter

RSISE @ ANU and SML @ NICTA

Canberra, ACT, 0200, Australia

marcus@hutter1.net www.hutter1.net

Abstract

Feature Markov Decision Processes (Φ MDPs) [Hut09] are well-suited for learning agents in general environments. Nevertheless, unstructured (Φ)MDPs are limited to relatively simple environments. Structured MDPs like Dynamic Bayesian Networks (DBNs) are used for large-scale real-world problems. In this article I extend Φ MDP to Φ DBN. The primary contribution is to derive a cost criterion that allows to automatically extract the most relevant features from the environment, leading to the “best” DBN representation. I discuss all building blocks required for a complete general learning algorithm.

Introduction

Agents. The agent-environment setup in which an *Agent* interacts with an *Environment* is a very general and prevalent framework for studying intelligent learning systems [RN03]. In cycles $t = 1, 2, 3, \dots$, the environment provides a (regular) *observation* $o_t \in \mathcal{O}$ (e.g. a camera image) to the agent; then the agent chooses an *action* $a_t \in \mathcal{A}$ (e.g. a limb movement); finally the environment provides a real-valued *reward* $r_t \in \mathbb{R}$ to the agent. The reward may be very scarce, e.g. just +1 (-1) for winning (losing) a chess game, and 0 at all other times [Hut05, Sec.6.3]. Then the next cycle $t+1$ starts. The agent’s objective is to maximize his reward.

Environments. For example, *sequence prediction* is concerned with environments that do not react to the agents actions (e.g. a weather-forecasting “action”) [Hut03], *planning* deals with the case where the environmental function is known [RPPCd08], *classification* and *regression* is for conditionally independent observations [Bis06], *Markov Decision Processes* (MDPs) assume that o_t and r_t only depend on a_{t-1} and o_{t-1} [SB98], POMDPs deal with *Partially Observable MDPs* [KLC98], and *Dynamic Bayesian Networks* (DBNs) with structured MDPs [BDH99].

Feature MDPs [Hut09]. Concrete real-world problems can often be modeled as MDPs. For this purpose, a *designer* extracts relevant features from the history (e.g. position and velocity of all objects), i.e. the *history* $h_t = a_1 o_1 r_1 \dots a_{t-1} o_{t-1} r_{t-1} o_t$ is summarized by a *feature* vector $s_t := \Phi(h_t)$. The feature vectors are regarded as *states* of an MDP and are assumed to be (approximately) Markov.

Artificial General Intelligence (AGI) [GP07] is concerned with designing *agents that perform well in a very large*

range of environments [LH07], including all of the mentioned ones above and more. In this general situation, it is not a priori clear what the useful features are. Indeed, any observation in the (far) past may be relevant in the future. A solution suggested in [Hut09] is to learn Φ itself.

If Φ keeps too much of the history (e.g. $\Phi(h_t) = h_t$), the resulting MDP is too large (infinite) and cannot be learned. If Φ keeps too little, the resulting state sequence is not Markov. The *Cost* criterion I develop formalizes this trade-off and is minimized for the “best” Φ . At any time n , the best Φ is the one that minimizes the Markov code length of $s_1 \dots s_n$ and $r_1 \dots r_n$. This reminds but is actually quite different from MDL, which minimizes model+data code length [Grü07].

Dynamic Bayesian networks. The use of “unstructured” MDPs [Hut09], even our Φ -optimal ones, is clearly limited to relatively simple tasks. Real-world problems are structured and can often be represented by dynamic Bayesian networks (DBNs) with a reasonable number of nodes [DK89]. Bayesian networks in general and DBNs in particular are powerful tools for modeling and solving complex real-world problems. Advances in theory and increase in computation power constantly broaden their range of applicability [BDH99; SDL07].

Main contribution. The primary contribution of this work is to extend the Φ *selection principle* developed in [Hut09] for MDPs to the conceptually much more demanding DBN case. The major extra complications are approximating, learning and coding the rewards, the dependence of the Cost criterion on the DBN structure, learning the DBN structure, and how to store and find the optimal value function and policy.

Although this article is self-contained, it is recommended to read [Hut09] first.

Feature Dynamic Bayesian Networks (Φ DBN)

In this section I recapitulate the definition of Φ MDP from [Hut09], and adapt it to DBNs. While formally a DBN is just a special case of an MDP, exploiting the additional structure efficiently is a challenge. For generic MDPs, typical algorithms should be polynomial and can at best be linear in the number of states $|\mathcal{S}|$. For DBNs we want algorithms that are polynomial in the number of features m . Such DBNs

have exponentially many states ($2^{O(m)}$), hence the standard MDP algorithms are exponential, not polynomial, in m . Deriving poly-time (and poly-space!) algorithms for DBNs by exploiting the additional DBN structure is the challenge. The gain is that we can handle exponentially large structured MDPs efficiently.

Notation. Throughout this article, \log denotes the binary logarithm, and $\delta_{x,y} = \delta_{xy} = 1$ if $x=y$ and 0 else is the Kronecker symbol. I generally omit separating commas if no confusion arises, in particular in indices. For any z of suitable type (string,vector,set), I define string $z = z_{1:l} = z_1 \dots z_l$, sum $z_+ = \sum_j z_j$, union $z_* = \bigcup_j z_j$, and vector $\mathbf{z}_* = (z_1, \dots, z_l)$, where j ranges over the full range $\{1, \dots, l\}$ and $l = |z|$ is the length or dimension or size of z . \hat{z} denotes an estimate of z . The characteristic function $\mathbb{1}_B = 1$ if $B=\text{true}$ and 0 else. $P(\cdot)$ denotes a probability over states and rewards or parts thereof. I do not distinguish between random variables Z and realizations z , and abbreviation $P(z) := P[Z = z]$ never leads to confusion. More specifically, $m \in \mathbb{N}$ denotes the number of features, $i \in \{1, \dots, m\}$ any feature, $n \in \mathbb{N}$ the current time, and $t \in \{1, \dots, n\}$ any time. Further, due to space constraints at several places I gloss over initial conditions or special cases where inessential. Also $0 * \text{undefined} = 0 * \text{infinity} = 0$.

Φ MDP definition. A Φ MDP consists of a 7 tuple $(\mathcal{O}, \mathcal{A}, \mathcal{R}, \text{Agent}, \text{Env}, \Phi, \mathcal{S}) = (\text{observation space, action space, reward space, agent, environment, feature map, state space})$. Without much loss of generality, I assume that \mathcal{A} and \mathcal{O} are finite and $\mathcal{R} \subseteq \mathbb{R}$. Implicitly I assume \mathcal{A} to be small, while \mathcal{O} may be huge.

Agent and Env are a pair or triple of interlocking functions of the history $\mathcal{H} := (\mathcal{O} \times \mathcal{A} \times \mathcal{R})^* \times \mathcal{O}$:

$$\begin{aligned} \text{Env} : \mathcal{H} \times \mathcal{A} \times \mathcal{R} &\rightsquigarrow \mathcal{O}, & o_n &= \text{Env}(h_{n-1} a_{n-1} r_{n-1}), \\ \text{Agent} : \mathcal{H} &\rightsquigarrow \mathcal{A}, & a_n &= \text{Agent}(h_n), \\ \text{Env} : \mathcal{H} \times \mathcal{A} &\rightsquigarrow \mathcal{R}, & r_n &= \text{Env}(h_n a_n). \end{aligned}$$

where \rightsquigarrow indicates that mappings \rightarrow might be stochastic. The informal goal of AI is to design an Agent() that achieves high (expected) reward over the agent's lifetime in a large range of Env()ironments.

The feature map Φ maps histories to states

$$\Phi : \mathcal{H} \rightarrow \mathcal{S}, \quad s_t = \Phi(h_t), \quad h_t = o a r_{1:t-1} o_t \in \mathcal{H}$$

The idea is that Φ shall extract the ‘‘relevant’’ aspects of the history in the sense that ‘‘compressed’’ history $s a r_{1:n} \equiv s_1 a_1 r_1 \dots s_n a_n r_n$ can well be described as a sample from some MDP $(\mathcal{S}, \mathcal{A}, T, R) = (\text{state space, action space, transition probability, reward function})$.

(Φ) Dynamic Bayesian Networks are structured (Φ)MDPs. The state space is $\mathcal{S} = \{0,1\}^m$, and each state $s \equiv \mathbf{x} \equiv (x^1, \dots, x^m) \in \mathcal{S}$ is interpreted as a feature vector $\mathbf{x} = \Phi(h)$, where $x^i = \Phi^i(h)$ is the value of the i th binary feature. In the following I will also refer to x^i as feature i , although strictly speaking it is its value. Since non-binary features can be realized as a list of binary features, I restrict myself to the latter.

Given $\mathbf{x}_{t-1} = \mathbf{x}$, I assume that the features $(x_t^1, \dots, x_t^m) = \mathbf{x}'$ at time t are independent, and that each x'^i depends only

on a subset of ‘‘parent’’ features $\mathbf{u}^i \subseteq \{x^1, \dots, x^m\}$, i.e. the transition matrix has the structure

$$T_{\mathbf{x}\mathbf{x}'}^a = P(\mathbf{x}_t = \mathbf{x}' | \mathbf{x}_{t-1} = \mathbf{x}, a_{t-1} = a) = \prod_{i=1}^m P^a(x'^i | \mathbf{u}^i) \quad (1)$$

This defines our **Φ DBN model**. It is just a Φ MDP with special \mathcal{S} and T . Explaining Φ DBN on an example is easier than staying general.

Φ DBN Example

Consider an instantiation of the simple vacuum world [RN03, Sec.3.6]. There are two rooms, A and B , and a vacuum Robot that can observe whether the room he is in is *Clean* or *Dirty*; *Move* to the other room, *Suck*, i.e. clean the room he is in; or do *Nothing*. After 3 days a room gets dirty again. Every clean room gives a reward 1, but a moving or sucking robot costs and hence reduces the reward by 1. Hence $\mathcal{O} = \{A, B\} \times \{C, D\}$, $\mathcal{A} = \{N, S, M\}$, $\mathcal{R} = \{-1, 0, 1, 2\}$, and the dynamics Env() (possible histories) is clear from the above description.

Dynamics as a DBN. We can model the dynamics by a DBN as follows: The state is modeled by 3 features. Feature $R \in \{A, B\}$ stores in which room the robot is, and feature $A/B \in \{0, 1, 2, 3\}$ remembers (capped at 3) how long ago the robot has cleaned room A/B last time, hence $\mathcal{S} = \{0, 1, 2, 3\} \times \{A, B\} \times \{0, 1, 2, 3\}$. The state/feature transition is as follows:

if $(x^R = A$ and $a = S)$ then $x'^A = 0$ else $x'^A = \min\{x^A + 1, 3\}$;
if $(x^R = B$ and $a = S)$ then $x'^B = 0$ else $x'^B = \min\{x^B + 1, 3\}$;
if $a = M$ (if $x^R = B$ then $x'^R = A$ else $x'^R = B$) else $x'^R = x^R$;

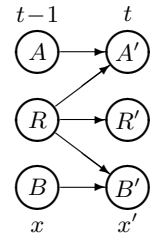
A DBN can be viewed as a two-layer Bayesian network [BDH99]. The dependency structure of our example is depicted in the right diagram.

Each feature consists of a (left,right)-pair of nodes, and a node $i \in \{1, 2, 3 = m\} \hat{=} \{A, R, B\}$ on the right is connected to all and only the parent features \mathbf{u}^i on the left. The reward is

$$r = \mathbb{1}_{x^A < 3} + \mathbb{1}_{x^B < 3} - \mathbb{1}_{a \neq N}$$

The features map $\Phi = (\Phi^A, \Phi^R, \Phi^B)$ can also be written down explicitly. It depends on the actions and observations of the last 3 time steps.

Discussion. Note that all nodes x'^i can implicitly also depend on the chosen action a . The optimal policies are repetitions of action sequence S, N, M or S, M, N . One might think that binary features $x^{A/B} \in \{C, D\}$ are sufficient, but this would result in a POMDP (Partially Observable MDP), since the cleanness of room A is not observed while the robot is in room B . That is, \mathbf{x}' would not be a (probabilistic) function of \mathbf{x} and a alone. The quaternary feature $x^A \in \{0, 1, 2, 3\}$ can easily be converted into two binary features, and similarly x^B . The purely deterministic example can easily be made stochastic. For instance, *Sucking* and *Moving* may fail with a certain probability. Possible, but more complicated is to model a probabilistic transition from *Clean* to *Dirty*. In the randomized versions the agent needs to use its observations.



ΦDBN Coding and Evaluation

I now construct a code for $s_{1:n}$ given $a_{1:n}$, and for $r_{1:n}$ given $s_{1:n}$ and $a_{1:n}$, which is optimal (minimal) if $s_{1:n}r_{1:n}$ given $a_{1:n}$ is sampled from some MDP. It constitutes our cost function for Φ and is used to define the Φ selection principle for DBNs. Compared to the MDP case, reward coding is more complex, and there is an extra dependence on the graphical structure of the DBN.

Recall [Hut09] that a sequence $z_{1:n}$ with counts $\mathbf{n} = (n_1, \dots, n_m)$ can within an additive constant be coded in

$$\text{CL}(\mathbf{n}) := n H(\mathbf{n}/n) + \frac{m'-1}{2} \log n \text{ if } n > 0 \text{ and } 0 \text{ else} \quad (2)$$

bits, where $n = n_+ = n_1 + \dots + n_m$ and $m' = |\{i : n_i > 0\}| \leq m$ is the number of non-empty categories, and $H(\mathbf{p}) := -\sum_{i=1}^m p_i \log p_i$ is the entropy of probability distribution \mathbf{p} . The code is optimal (within $+O(1)$) for all i.i.d. sources.

State/Feature Coding. Similarly to the Φ MDP case, we need to code the temporal “observed” state=feature sequence $\mathbf{x}_{1:n}$. I do this by a frequency estimate of the state/feature transition probability. (Within an additive constant, MDL, MML, combinatorial, incremental, and Bayesian coding all lead to the same result). In the following I will drop the prime in (\mathbf{u}^i, a, x^i) tuples and related situations if/since it does not lead to confusion. Let $\mathcal{T}_{\mathbf{u}^i x^i}^{ia} = \{t \leq n : \mathbf{u}_{t-1} = \mathbf{u}^i, a_{t-1} = a, x_t^i = x^i\}$ be the set of times $t-1$ at which features that influence x^i have values \mathbf{u}^i , and action is a , and which leads to feature i having value x^i . Let $n_{\mathbf{u}^i x^i}^{ia} = |\mathcal{T}_{\mathbf{u}^i x^i}^{ia}|$ their number ($n_{\mathbf{u}^i}^{i+} = n \forall i$). I estimate each feature probability separately by $\hat{P}^a(x^i | \mathbf{u}^i) = n_{\mathbf{u}^i x^i}^{ia} / n_{\mathbf{u}^i}^{i+}$. Using (1), this yields

$$\begin{aligned} \hat{P}(\mathbf{x}_{1:n} | a_{1:n}) &= \prod_{t=1}^n \hat{T}_{\mathbf{x}_{t-1} \mathbf{x}_t}^{a_{t-1}} = \prod_{t=1}^n \prod_{i=1}^m \hat{P}^{a_{t-1}}(x_t^i | \mathbf{u}_{t-1}^i) \\ &= \dots = \exp \left[\sum_{i, \mathbf{u}^i, a} n_{\mathbf{u}^i}^{ia} H \left(\frac{n_{\mathbf{u}^i}^{ia}}{n_{\mathbf{u}^i}^{i+}} \right) \right] \end{aligned}$$

The length of the Shannon-Fano code of $\mathbf{x}_{1:n}$ is just the logarithm of this expression. We also need to code each non-zero count $n_{\mathbf{u}^i x^i}^{ia}$ to accuracy $O(1/\sqrt{n_{\mathbf{u}^i}^{ia}})$, which each needs $\frac{1}{2} \log(n_{\mathbf{u}^i}^{ia})$ bits. Together this gives a complete code of length

$$\text{CL}(\mathbf{x}_{1:n} | a_{1:n}) = \sum_{i, \mathbf{u}^i, a} \text{CL}(n_{\mathbf{u}^i}^{ia}) \quad (3)$$

The rewards are more complicated.

Reward structure. Let $R_{\mathbf{x}\mathbf{x}'}^a$ be (a model of) the observed reward when action a in state \mathbf{x} results in state \mathbf{x}' . It is natural to assume that the structure of the rewards $R_{\mathbf{x}\mathbf{x}'}^a$ is related to the transition structure $T_{\mathbf{x}\mathbf{x}'}^a$. Indeed, this is not restrictive, since one can always consider a DBN with the union of transition and reward dependencies. Usually it is assumed that the “global” reward is a *sum* of “local” rewards $R_{\mathbf{u}^i x^i}^{ia}$, one for each feature i [KP99]. For simplicity of exposition I assume that the local reward R^i only depends on the feature value x^i and not on \mathbf{u}^i and a . Even this is not restrictive

and actually may be advantageous as discussed in [Hut09] for MDPs. So I assume

$$R_{\mathbf{x}\mathbf{x}'}^a = \sum_{i=1}^m R_{x^i}^{ia} =: R(\mathbf{x}')$$

For instance, in the example in the previous section, two local rewards ($R_{x^A}^A = \mathbb{1}_{x^A < 3}$ and $R_{x^B}^B = \mathbb{1}_{x^B < 3}$) depend on \mathbf{x}' only, but the third reward depends on the action ($R^R = -\mathbb{1}_{a \neq N}$).

Often it is assumed that the local rewards are directly observed or known [KP99], but we neither want nor can do this here: Having to specify many local rewards is an extra burden for the environment (e.g. the teacher), which preferably should be avoided. In our case, it is not even possible to pre-specify a local reward for each feature, since the features Φ^i themselves are learned by the agent and are not statically available. They are agent-internal and not part of the Φ DBN interface. In case multiple rewards *are* available, they can be modeled as part of the regular observations o , and r only holds the overall reward. The agent must and can learn to interpret and exploit the local rewards in o by himself.

Learning the reward function. In analogy to the MDP case for R and the DBN case for T above it is tempting to estimate $R_{x^i}^i$ by $\sum_r r' n_{+x^i}^{ir'} / n_{+x^i}^{i+}$ but this makes no sense. For instance if $r_t = 1 \forall t$, then $\hat{R}_{x^i}^i \equiv 1$, and $\hat{R}_{\mathbf{x}\mathbf{x}'}^a \equiv m$ is a gross mis-estimation of $r_t \equiv 1$. The localization of the global reward is somewhat more complicated. The goal is to choose $R_{x^1}^1, \dots, R_{x^m}^m$ such that $r_t = R(\mathbf{x}_t) \forall t$.

Without loss we can set $R_0^i \equiv 0$, since we can subtract a constant from each local reward and absorb them into an overall constant w_0 . This allows us to write

$$R(\mathbf{x}) = w_0 x^0 + w_1 x^1 + \dots + w_m x^m = \mathbf{w}^\top \mathbf{x}$$

where $w_i := R_1^i$ and $x^0 \equiv 1$.

In practice, the Φ DBN model will not be perfect, and an approximate solution, e.g. a least squares fit, is the best we can achieve. The square loss can be written as

$$\text{Loss}(\mathbf{w}) := \sum_{t=1}^n (R(\mathbf{x}_t) - r_t)^2 = \mathbf{w}^\top A \mathbf{w} - 2\mathbf{b}^\top \mathbf{w} + c \quad (4)$$

$$A_{ij} := \sum_{t=1}^n x_t^i x_t^j, \quad b_i := \sum_{t=1}^n r_t x_t^i, \quad c := \sum_{t=1}^n r_t^2$$

Note that A_{ij} counts the number of times feature i and j are “on” (=1) simultaneously, and b_i sums all rewards for which feature i is on. The loss is minimized for

$$\hat{\mathbf{w}} := \arg \min_{\mathbf{w}} \text{Loss}(\mathbf{w}) = A^{-1} \mathbf{b}, \quad \hat{R}(\mathbf{x}) = \hat{\mathbf{w}}^\top \mathbf{x}$$

which involves an inversion of the $(m+1) \times (m+1)$ matrix A . For singular A we take the pseudo-inverse.

Reward coding. The quadratic loss function suggests a Gaussian model for the rewards:

$$P(r_{1:n} | \hat{\mathbf{w}}, \sigma) := \exp(-\text{Loss}(\hat{\mathbf{w}})/2\sigma^2) / (2\pi\sigma^2)^{n/2}$$

Maximizing this w.r.t. the variance σ^2 yields the maximum likelihood estimate

$$-\log P(r_{1:n} | \hat{\mathbf{w}}, \hat{\sigma}) = \frac{n}{2} \log(\text{Loss}(\hat{\mathbf{w}})) - \frac{n}{2} \log \frac{n\epsilon}{2\pi}$$

where $\hat{\sigma}^2 = \text{Loss}(\hat{\mathbf{w}})/n$. Given $\hat{\mathbf{w}}$ and $\hat{\sigma}$ this can be regarded as the (Shannon-Fano) code length of $r_{1:n}$ (there are actually a few subtleties here which I gloss over). Each weight \hat{w}_k and $\hat{\sigma}$ need also be coded to accuracy $O(1/\sqrt{n})$, which needs $(m+2)\frac{1}{2}\log n$ bits total. Together this gives a complete code of length

$$\begin{aligned} \text{CL}(r_{1:n}|\mathbf{x}_{1:n}a_{1:n}) &= \\ &= \frac{n}{2} \log(\text{Loss}(\hat{\mathbf{w}})) + \frac{m+2}{2} \log n - \frac{n}{2} \log \frac{ne}{2\pi} \end{aligned} \quad (5)$$

ΦDBN evaluation and selection is similar to the MDP case. Let G denote the graphical structure of the DBN, i.e. the set of parents $\text{Pa}^i \subseteq \{1, \dots, m\}$ of each feature i . (Remember \mathbf{u}^i are the parent values). Similarly to the MDP case, the cost of (Φ, G) on h_n is defined as

$$\text{Cost}(\Phi, G|h_n) := \text{CL}(\mathbf{x}_{1:n}|a_{1:n}) + \text{CL}(r_{1:n}|\mathbf{x}_{1:n}, a_{1:n}), \quad (6)$$

and the best (Φ, G) minimizes this cost.

$$(\Phi^{best}, G^{best}) := \arg \min_{\Phi, G} \{\text{Cost}(\Phi, G|h_n)\}$$

A general discussion why this is a good criterion can be found in [Hut09]. In the following section I mainly highlight the difference to the MDP case, in particular the additional dependence on and optimization over G .

DBN Structure Learning & Updating

This section briefly discusses minimization of (6) w.r.t. G given Φ and even briefer minimization w.r.t. Φ . For the moment regard Φ as given and fixed.

Cost and DBN structure. For general structured local rewards $R_{\mathbf{u}^i, x^i}^i$, (3) and (5) both depend on G , and (6) represents a novel DBN structure learning criterion that includes the rewards.

For our simple reward model $R_{x^i}^i$, (5) is independent of G , hence only (3) needs to be considered. This is a standard MDL criterion, but I have not seen it used in DBNs before. Further, the features i are independent in the sense that we can search for the optimal parent sets $\text{Pa}^i \subseteq \{1, \dots, m\}$ for each feature i separately.

Complexity of structure search. Even in this case, finding the optimal DBN structure is generally hard. In principle we could rely on off-the-shelf heuristic search methods for finding good G , but it is probably better to use or develop some special purpose optimizer. One may even restrict the space of considered graphs G to those for which (6) can be minimized w.r.t. G efficiently, as long as this restriction can be compensated by “smarter” Φ .

A brute force exhaustive search algorithm for Pa^i is to consider all 2^m subsets of $\{1, \dots, m\}$ and select the one that minimizes $\sum_{\mathbf{u}^i, a} \text{CL}(\mathbf{n}_{\mathbf{u}^i}^i)$. A reasonable and often employed assumption is to limit the number of parents to some small value p , which reduces the search space size to $O(m^p)$.

Indeed, since the Cost is exponential in the maximal number of parents of a feature, but only linear in n , a Cost minimizing Φ can usually not have more than a logarithmic number of parents, which leads to a search space that is pseudo-polynomial in m .

Heuristic structure search. We could also replace the well-founded criterion (3) by some heuristic. One such heuristic has been developed in [SDL07]. The mutual information is another popular criterion for determining the dependency of two random variables, so we could add j as a parent of feature i if the mutual information of x^j and x^i is above a certain threshold. Overall this takes time $O(m^2)$ to determine G . An MDL inspired threshold for binary random variables is $\frac{1}{2n} \log n$. Since the mutual information treats parents independently, \hat{T} has to be estimated accordingly, essentially as in naive Bayes classification [Lew98] with feature selection, where x^i represents the class label and \mathbf{u}^i are the features selected \mathbf{x} . The improved Tree-Augmented naive Bayes (TAN) classifier [FGG97] could be used to model synchronous feature dependencies (i.e. within a time slice). The Chow-Liu [CL68] minimum spanning tree algorithm allows determining G in time $O(m^3)$. A tree becomes a forest if we employ a lower threshold for the mutual information.

Φ search is even harder than structure search, and remains an art. Nevertheless the reduction of the complex (ill-defined) reinforcement learning problem to an internal feature search problem with well-defined objective is a clear conceptual advance.

In principle (but not in practice) we could consider the set of *all* (computable) functions $\{\Phi : \mathcal{H} \rightarrow \{0,1\}\}$. We then compute $\text{Cost}(\Phi|h)$ for every finite subset $\Phi = \{\Phi^{i_1}, \dots, \Phi^{i_m}\}$ and take the minimum (note that the order is irrelevant).

Most practical search algorithms require the specification of some neighborhood function, here for Φ . For instance, stochastic search algorithms suggest and accept a neighbor of Φ with a probability that depends on the Cost reduction. See [Hut09] for more details. Here I will only present some very simplistic ideas for features and neighborhoods.

Assume binary observations $\mathcal{O} = \{0,1\}$ and consider the last m observations as features, i.e. $\Phi^i(h_n) = o_{n-i+1}$ and $\Phi(h_n) = (\Phi^1(h_n), \dots, \Phi^m(h_n)) = o_{n-m+1:n}$. So the states are the same as for Φ_m MDP in [Hut09], but now $\mathcal{S} = \{0,1\}^m$ is structured as m binary features. In the example here, $m = 5$ lead to a perfect Φ DBN. We can add a new feature o_{n-m} ($m \rightsquigarrow m+1$) or remove the last feature ($m \rightsquigarrow m-1$), which defines a natural neighborhood structure.

Note that the context trees of [McC96; Hut09] are more flexible. To achieve this flexibility here we either have to use smarter features within our framework (simply interpret $s = \Phi_S(h)$ as a feature vector of length $m = \lceil \log |\mathcal{S}| \rceil$) or use smarter (non-tabular) estimates of $P^a(x^i|\mathbf{u}^i)$ extending our framework (to tree dependencies).

For general purpose intelligent agents we clearly need more powerful features. Logical expressions or (non)accepting Turing machines or recursive sets can map histories or parts thereof into true/false or accept/reject or in/out, respectively, hence naturally represent binary features. Randomly generating such expressions or programs with an appropriate bias towards simple ones is a universal feature generator that eventually finds the optimal feature map. The idea is known as Universal Search [Gag07].

Value & Policy Learning in Φ DBN

Given an estimate $\hat{\Phi}$ of Φ^{best} , the next step is to determine a good action for our agent. I mainly concentrate on the difficulties one faces in adapting MDP algorithms and discuss state of the art DBN algorithms. Value and policy learning in known finite state MDPs is easy provided one is satisfied with a polynomial time algorithm. Since a DBN is just a special (structured) MDP, its (Q) Value function respects the same Bellman equations [Hut09, Eq.(6)], and the optimal policy is still given by $a_{n+1} := \operatorname{argmax}_a Q_{x_{n+1}}^{*a}$. Nevertheless, their solution is now a nightmare, since the state space is exponential in the number of features. We need algorithms that are polynomial in the number of features, i.e. logarithmic in the number of states.

Value function approximation. The first problem is that the optimal value and policy do not respect the structure of the DBN. They are usually complex functions of the (exponentially many) states, which cannot even be stored, not to mention computed [KP99]. It has been suggested that the value can often be approximated well as a sum of local values similarly to the rewards. Such a value function can at least be stored.

Model-based learning. The default quality measure for the approximate value is the ρ -weighted squared difference, where ρ is the stationary distribution.

Even for a fixed policy, value iteration does *not* converge to the best approximation, but usually converges to a fixed point close to it [BT96]. Value iteration requires ρ explicitly. Since ρ is also too large to store, one has to approximate ρ as well. Another problem, as pointed out in [KP00], is that policy iteration may not converge, since different policies have different (misleading) stationary distributions. Koller and Parr [KP00] devised algorithms for general factored ρ , and Guestrin et al. [GKPV03] for max-norm, alleviating this problem. Finally, general policies cannot be stored exactly, and another restriction or approximation is necessary.

Model-free learning. Given the difficulties above, I suggest to (re)consider a very simple class of algorithms, without suggesting that it is better. The above model-based algorithms exploit \hat{T} and \hat{R} directly. An alternative is to sample from \hat{T} and use model-free “Temporal Difference (TD)” learning algorithms based only on this internal virtual sample [SB98]. We could use TD(λ) or Q -value variants with linear value function approximation.

Beside their simplicity, another advantage is that neither the stationary distribution nor the policy needs to be stored or approximated. Once approximation \hat{Q}^* has been obtained, it is trivial to determine the optimal (w.r.t. \hat{Q}^*) action via $a_{n+1} = \operatorname{argmax}_a Q_{x_{n+1}}^{*a}$ for any state of interest (namely x_{n+1}) exactly.

Exploration. Optimal actions based on approximate rather than exact values can lead to very poor behavior due to lack of exploration. There are polynomially optimal algorithms (Rmax,E3,OIM) for the exploration-exploitation dilemma.

For model-based learning, extending E3 to DBNs is straightforward, but E3 needs an oracle for planning in a given DBN [KK99]. Recently, Strehl et al. [SDL07] accomplished the same for Rmax. They even learn the DBN struc-

ture, albeit in a very simplistic way. Algorithm OIM [SL08], which I described in [Hut09] for MDPs, can also likely be generalized to DBNs, and I can imagine a model-free version.

Incremental Updates

As discussed two sections ago, most search algorithms are local in the sense that they produce a chain of “slightly” modified candidate solutions, here Φ ’s. This suggests a potential speedup by computing quantities of interest incrementally.

Cost. Computing $\text{CL}(x|a)$ in (3) takes at most time $O(m2^k|A|)$, where k is the maximal number of parents of a feature. If we remove feature i , we can simply remove/subtract the contributions from i in the sum. If we add a new feature $m+1$, we only need to search for the best parent set u^{m+1} for this new feature, and add the corresponding code length. In practice, many transitions don’t occur, i.e. $n_{u^i x^i}^{ia} = 0$, so $\text{CL}(x|a)$ can actually be computed much faster in time $O(|\{n_{u^i x^i}^{ia} > 0\}|)$, and incrementally even faster.

Rewards. When adding a new feature, the current local reward estimates may not change much. If we reassign a fraction $\alpha \leq 1$ of reward to the new feature x^{m+1} , we get the following ansatz¹.

$$\hat{R}(x^1, \dots, x^{m+1}) = (1-\alpha)\hat{R}(x) + w_{m+1}x^{m+1} =: v^\top \psi(x) \\ v := (1-\alpha, w_{m+1})^\top, \quad \psi := (\hat{R}(x), x^{m+1})^\top$$

Minimizing $\sum_{t=1}^n (\hat{R}(x_t^1 \dots x_t^{m+1}) - r_t)^2$ w.r.t. v analogous to (4) just requires a trivial 2×2 matrix inversion. The minimum \tilde{v} results in an initial new estimate $\tilde{w} = ((1-\tilde{\alpha})\hat{w}_0, \dots, (1-\tilde{\alpha})\hat{w}_m, \tilde{w}_{m+1})^\top$, which can be improved by some first order gradient decent algorithm in time $O(m)$, compared to the exact $O(m^3)$ algorithm. When removing a feature, we simply redistribute its local reward to the other features, e.g. uniformly, followed by improvement steps that cost $O(m)$ time.

Value. All iteration algorithms described in the previous section for computing (Q) Values need an initial value for V or Q . We can take the estimate \hat{V} from a previous Φ as an initial value for the new Φ . Similarly as for the rewards, we can redistribute a fraction of the values by solving relatively small systems of equations. The result is then used as an initial value for the iteration algorithms in the previous section. A further speedup can be obtained by using prioritized iteration algorithms that concentrate their time on badly estimated parameters, which are in our case the new values [SB98].

Similarly, results from time t can be (re)used as initial estimates for the next cycle $t+1$, followed by a fast improvement step.

Outlook

Φ DBN leaves much more questions open and room for modifications and improvements than Φ MDP. Here are a few.

¹An *Ansatz* is an initial mathematical or physical model with some free parameters to be determined subsequently. [<http://en.wikipedia.org/wiki/Ansatz>]

- The cost function can be improved by integrating out the states analogous to the Φ MDP case [Hut09]: The likelihood $P(r_{1:n}|a_{1:n}, \hat{U})$ is unchanged, except that $\hat{U} \equiv \hat{T}\hat{R}$ is now estimated locally, and the complexity penalty becomes $\frac{1}{2}(M+m+2)\log n$, where M is (essentially) the number of non-zero counts $n_{u^i x^i}^{i a}$, but an efficient algorithm has yet to be found.
- It may be necessary to impose and exploit structure on the conditional probability tables $P^a(x^i|u^i)$ themselves [BDH99].
- Real-valued observations and beliefs suggest to extend the binary feature model to $[0,1]$ interval valued features rather than coding them binary. Since any continuous semantics that preserves the role of 0 and 1 is acceptable, there should be an efficient way to generalize Cost and Value estimation procedures.
- I assumed that the reward/value is linear in local rewards/values. Is this sufficient for all practical purposes? I also assumed a least squares and Gaussian model for the local rewards. There are efficient algorithms for much more flexible models. The least we could do is to code w.r.t. the proper covariance A .
- I also barely discussed synchronous (within time-slice) dependencies.
- I guess Φ DBN will often be able to work around too restrictive DBN models, by finding features Φ that are more compatible with the DBN and reward structure.
- Extra edges in the DBN can improve the linear value function approximation. To give Φ DBN incentives to do so, the Value would have to be included in the Cost criterion.
- Implicitly I assumed that the action space \mathcal{A} is small. It is possible to extend Φ DBN to large structured action spaces.
- Apart from the Φ -search, all parts of Φ DBN seem to be poly-time approximable, which is satisfactory in theory. In practice, this needs to be improved to essentially linear time in n and m .
- Developing smart Φ generation and smart stochastic search algorithms for Φ are the major open challenges.
- A more Bayesian Cost criterion would be desirable: a likelihood of h given Φ and a prior over Φ leading to a posterior of Φ given h , or so. Monte Carlo (search) algorithms like Metropolis-Hastings could sample from such a posterior. Currently probabilities ($\cong 2^{-CL}$) are assigned only to rewards and states, but not to observations and feature maps.

Summary. In this work I introduced a powerful framework (Φ DBN) for general-purpose intelligent learning agents, and presented algorithms for all required building blocks. The introduced cost criterion reduced the informal reinforcement learning problem to an internal well-defined search for “relevant” features.

References

- [BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- [Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [BT96] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
- [CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.
- [DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
- [FGG97] N. Friedman, D. Geiger, and M. Goldszmid. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.
- [Gag07] M. Gaglio. Universal search. *Scholarpedia*, 2(11):2575, 2007.
- [GKPV03] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003.
- [GP07] B. Goertzel and C. Pennachin, editors. *Artificial General Intelligence*. Springer, 2007.
- [Grü07] P. D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, Cambridge, 2007.
- [Hut03] M. Hutter. Optimality of universal Bayesian prediction for general loss and alphabet. *Journal of Machine Learning Research*, 4:971–1000, 2003.
- [Hut05] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005. 300 pages, <http://www.hutter1.net/ai/uaibook.htm>.
- [Hut09] M. Hutter. Feature Markov decision processes. In *Artificial General Intelligence (AGI’09)*. Atlantis Press, 2009.
- [KK99] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 740–747, San Francisco, 1999. Morgan Kaufmann.
- [KLC98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- [KP99] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *Proc. 16th International Joint Conf. on Artificial Intelligence (IJCAI’99)*, pages 1332–1339, Edinburgh, 1999.
- [KP00] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proc. 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 326–334, San Francisco, CA, 2000. Morgan Kaufmann.
- [Lew98] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proc. 10th European Conference on Machine Learning (ECML’98)*, pages 4–15, Chemnitz, DE, 1998. Springer.
- [LH07] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds & Machines*, 17(4):391–444, 2007.
- [McC96] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1996.
- [RN03] S. J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.
- [RPPCd08] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008(32):663–704, 2008.
- [SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [SDL07] A. L. Strehl, C. Diuk, and M. L. Littman. Efficient structure learning in factored-state MDPs. In *Proc. 27th AAAI Conference on Artificial Intelligence*, pages 645–650, Vancouver, BC, 2007. AAAI Press.
- [SL08] I. Szita and A. Lörincz. The many faces of optimism: a unifying approach. In *Proc. 12th International Conference (ICML 2008)*, volume 307, Helsinki, Finland, June 2008.

Economic Attention Networks: Associative Memory and Resource Allocation for General Intelligence

Matthew Ikle', Joel Pitt, Ben Goertzel, George Sellman

Adams State College (ASC), Singularity Institute for AI (SIAI), Novamente LLC and SIAI, ASC
1405 Bernerd Place, Rockville MD 20851, USA
ben@goertzel.org, stephan@bugaj.com

Abstract

A novel method for simultaneously storing memories and allocating resources in AI systems is presented. The method, Economic Attention Networks (ECANs), bears some resemblance to the spread of activation in attractor neural networks, but differs via explicitly differentiating two kinds of “activation” (Short Term Importance, related to processor allocation; and Long Term Importance, related to memory allocation), and in using equations that are based on ideas from economics rather than approximative neural modeling. Here we explain the basic ideas of ECANs, and then investigate the functionality of ECANs as associative memories, via mathematical analysis and the reportage of experimental results obtained from the implementation of ECANs in the OpenCog integrative AGI system.

Economic Attention Networks or ECANs. ECANs have been designed and implemented within an integrative AGI framework called OpenCog (which overlaps with the related Novamente Cognition Engine system; see Goertzel, 2006). However, ECANs also have meaning outside the OpenCog context; they may be considered nonlinear dynamical systems in roughly the same family as attractor neural networks such as Hopfield nets (Amit, 1992). The main focus of this paper is the study of ECANs as associative memories, which involves mathematical and experimental analyses that are independent of the embedding of ECANs in OpenCog or other AGI systems. But we will also discuss the implications of these results for specific interactions between ECANs and other OpenCog components

Introduction

One of the critical challenges confronting any system aimed at advanced general intelligence is the allocation of computational resources. The central nature of this issue is highlighted by Hutter's (2004) mathematical results showing that if one formalizes intelligence as the achievement of complex computable goals, then there are very simple software programs that can achieve arbitrarily high degrees of intelligence, so long as they are allotted huge amounts of computational resources. In this sense, coping with space and time limitations is the crux of the AGI problem.

Not surprisingly, given its central nature, the management of computational resources ties in with a variety of other concrete issues that AGI systems confront, in ways depending on the specific system in question. In the approach we will describe here, resource allocation is carried out by the same structures and dynamics as associative memory, whereas the relationship between resource allocation and other system processes like reasoning and procedure learning involves feedback between distinct software components.

We will describe here a specific approach to resource allocation and associative memory, which we call

Economic Attention Networks

First we summarize the essential ideas of ECANs; in later sections two specific variants of ECAN equational formalizations are presented.

An ECAN is a graph, consisting of un-typed nodes and links, and also links that may be typed either HebbianLink or InverseHebbianLink. It is also useful sometimes to consider ECANs that extend the traditional graph formalism and involve links that point to links as well as to nodes. The term Atom will be used to refer to either nodes or links. Each Atom in an ECAN is weighted with two numbers, called STI (short-term importance) and LTI (long-term importance). Each Hebbian or InverseHebbian link is weighted with a probability value.

The equations of an ECAN explain how the STI, LTI and Hebbian probability values get updated over time. The metaphor underlying these equations is the interpretation of STI and LTI values as (separate) artificial currencies. The motivation for this metaphor has been elaborated somewhat in (Goertzel, 2007) and will not be recapitulated here. The fact that STI (for instance) is a currency means that the total amount of STI in the system is conserved (except in unusual instances where the ECAN controller

decides to introduce inflation or deflation and explicitly manipulate the amount of currency in circulation), a fact that makes the dynamics of an ECAN dramatically different than that of, say, an attractor neural network (in which there is no law of conservation of activation).

Conceptually, the STI value of an Atom is interpreted to indicate the immediate urgency of the Atom to the ECAN at a certain point in time; whereas the LTI value of an Atom indicates the amount of value the ECAN perceives in the retention of the Atom in memory (RAM). An ECAN will often be coupled with a “Forgetting” process that removes low-LTI Atoms from memory according to certain heuristics.

STI and LTI values will generally vary continuously, but the ECAN equations we introduce below contain the notion of an AttentionalFocus (AF), consisting of those Atoms in the ECAN with the highest STI values. The AF is given its meaning by the existence of equations that treat Atoms with STI above a certain threshold differently.

Conceptually, the probability value of a HebbianLink from A to B is the odds that if A is in the AF, so is B; and correspondingly, the InverseHebbianLink from A to B is weighted with the odds that if A is in the AF, then B is not. A critical aspect of the ECAN equations is that Atoms periodically spread their STI and LTI to other Atoms that connect to them via Hebbian and InverseHebbianLinks; this is the ECAN analogue of activation spreading in neural networks.

Based on the strong correspondences, one could plausibly label ECANs as “Economic Neural Networks”; however we have chosen not to go that path, as ECANs are not intended as plausible neural models, but rather as nonlinear dynamical systems engineered to fulfill certain functions within non-brain-emulative AGI systems.

Integration into OpenCog and the NCE

The OpenCog AGI framework, within which the current ECAN implementation exists, is a complex framework with a complex underlying theory, and here we will only hint at some of its key aspects. OpenCog is an open-source software framework designed to support the construction of multiple AI systems; and the current main thrust of work within OpenCog is the implementation of a specific AGI design called OpenCogPrime (OCP), which is presented in the online wikibook (Goertzel, 2008). Much of the OpenCog software code, and many of the ideas in the OCP design, have derived from the open-sourcing of aspects of the proprietary Novamente Cognition Engine, which has been described extensively in previous publications.

The first key entity in the OpenCog software architecture is the AtomTable, which is a repository for weighted, labeled hypergraph nodes and hyperedges. In the OpenCog implementation of ECANs, the nodes and links involved in the ECAN are stored here. OpenCog also contains an object called the CogServer, which wraps up an AtomTable as well as (among other objects) a Scheduler that schedules a set of MindAgent objects that each (when allocated processor time by the Scheduler)

carry out cognitive operations involving the AtomTable. The essence of the OCP design consists of a specific set of MindAgents designed to work together in a collaborative way in order to create a system that carries out actions oriented toward achieving goals (where goals are represented as specific nodes in the AtomTable, and actions are represented as Procedure objects indexed by Atoms in the AtomTable, and the utility of a procedure for achieving a goal is represented by a certain set of probabilistic logical links in the AtomTable, etc.). OpenCog is still at an experimental stage but has been used for such projects as statistical language analysis, probabilistic inference, and the control of virtual agents in online virtual worlds (see opencog.org).

So, in an OpenCog context, ECAN consists of a set of Atom types, and then a set of MindAgents carrying out ECAN operations such as HebbianLinkUpdating and ImportanceUpdating. OCP also requires many other MindAgents carrying out other cognitive processes such as probabilistic logical inference according to the PLN system (Goertzel et al, 2008) and evolutionary procedure learning according to the MOSES system (Looks, 2006). The interoperation of the ECAN MindAgents with these other MindAgents is a subtle issue that will be briefly discussed in the final section of the paper, but the crux is simple to understand.

The CogServer is understood to maintain a kind of central bank of STI and LTI funds. When a non-EAN MindAgent finds an Atom valuable, it sends that Atom a certain amount of Stimulus, which results in that Atom’s STI and LTI values being increased (via equations to be presented below, that transfer STI and LTI funds from the CogServer to the Atoms in question). Then, the ECAN ImportanceUpdating MindAgent carries out multiple operations, including some that transfer STI and LTI funds from some Atoms back to the CogServer.

Definition and Analysis of Variant 1

We now define a specific set of equations in accordance with the ECAN conceptual framework described above.

We define $\mathbf{H}_{STI} = [s_1, \dots, s_n]$ to be the vector of STI

values, and $\mathbf{C} = \begin{bmatrix} c_{11}, \dots, c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1}, \dots, c_{nn} \end{bmatrix}$ to be the connection matrix of

Hebbian probability values, where it is assumed that the existence of a HebbianLink or InverseHebbianLink between A and B are mutually exclusive possibilities. We

also define $\mathbf{C}_{LTI} = \begin{bmatrix} g_{11}, \dots, g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1}, \dots, g_{nn} \end{bmatrix}$ to be the matrix of LTI

values for each of the corresponding links.

We assume an updating scheme in which, periodically, a number of Atoms are allocated Stimulus amounts, which

causes the corresponding STI values to change according to the equations

$$\forall i: s_i = s_i - \text{rent} + \text{wages},$$

where rent and wages are given by

$$\text{rent} = \begin{cases} \langle \text{Rent} \rangle \cdot \max \left(0, \frac{\log \left(\frac{20s_i}{\text{recentMaxSTI}} \right)}{2} \right), & \text{if } s_i > 0 \\ 0, & \text{if } s_i \leq 0 \end{cases}$$

and

$$\text{wages} = \begin{cases} \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{\sum_{i=1}^n p_i}, & \text{if } p_i = 1 \\ \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{n - \sum_{i=1}^n p_i}, & \text{if } p_i = 0 \end{cases},$$

where $\mathbf{P} = [p_1, \dots, p_n]$, with $p_i \in \{0,1\}$ is the cue pattern for the pattern that is to be retrieved.

All quantities enclosed in angled brackets are system parameters, and LTI updating is accomplished using a completely analogous set of equations.

The changing STI values then cause updating of the connection matrix, according to the ‘‘conjunction’’ equations. First define

$$\text{norm}_i = \begin{cases} \frac{s_i}{\text{recentMaxSTI}}, & \text{if } s_i \geq 0. \\ \frac{s_i}{\text{recentMinSTI}}, & \text{if } s_i < 0 \end{cases}$$

Next define

$$\text{conj} = \text{Conjunction}(s_i, s_j) = \text{norm}_i \times \text{norm}_j$$

and

$$c'_{ij} = \langle \text{ConjDecay} \rangle \text{conj} + (1 - \langle \text{ConjDecay} \rangle) c_{ij}.$$

Finally update the matrix elements by setting

$$c_{ij} = \begin{cases} c_{ij} = c'_{ij}, & \text{if } c'_{ij} \geq 0 \\ c'_{ij}, & \text{if } c'_{ij} < 0 \end{cases}.$$

We are currently also experimenting with updating the connection matrix in accordance with the equations suggested by Storkey (1997, 1998, 1999.)

A key property of these equations is that both wages paid to, and rent paid by, each node are positively correlated to their STI values. That is, the more important nodes are paid more for their services, but they also pay more in rent.

A fixed percentage of the links with the lowest LTI values is then forgotten (which corresponds equationally to setting the LTI to 0).

Separately from the above, the process of Hebbian probability updating is carried out via a diffusion process in which some nodes ‘‘trade’’ STI utilizing a diffusion matrix \mathbf{D} , a version of the connection matrix \mathbf{C} normalized so that \mathbf{D} is a left stochastic matrix. \mathbf{D} acts on a similarly scaled vector \mathbf{v} , normalized so that \mathbf{v} is equivalent to a probability vector of STI values.

The decision about which nodes diffuse in each diffusion cycle is carried out via a decision function. We currently are working with two types of decision functions: a standard threshold function, by which nodes diffuse if and only if the nodes are in the AF; and a stochastic decision function in which nodes diffuse with probability $\frac{\tanh(\text{shape}(s_i - \text{FocusBoundary})) + 1}{2}$, where shape and

FocusBoundary are parameters.

The details of the diffusion process are as follows. First, construct the diffusion matrix from the entries in the connection matrix as follows:

$$\text{If } c_{ij} \geq 0, \text{ then } d_{ij} = c_{ij},$$

$$\text{else, set } d_{ji} = -c_{ij}.$$

Next, we normalize the columns of \mathbf{D} to make \mathbf{D} a left stochastic matrix. In so doing, we ensure that each node spreads no more than a $\langle \text{MaxSpread} \rangle$ proportion of its STI, by setting

$$\text{if } \sum_{i=1}^n d_{ij} > \langle \text{MaxSpread} \rangle:$$

$$d_{ij} = \begin{cases} d_{ij} \times \frac{\langle \text{MaxSpread} \rangle}{\sum_{i=1}^n d_{ij}}, & \text{for } i \neq j \\ d_{jj} = 1 - \langle \text{MaxSpread} \rangle \end{cases}$$

else:

$$d_{jj} = 1 - \sum_{\substack{i=1 \\ i \neq j}}^n d_{ij}$$

Now we obtain a scaled STI vector \mathbf{v} by setting

$$\text{minSTI} = \min_{i \in \{1,2,\dots,n\}} s_i \text{ and } \text{maxSTI} = \max_{i \in \{1,2,\dots,n\}} s_i$$

$$v_i = \frac{s_i - \text{minSTI}}{\text{maxSTI} - \text{minSTI}}$$

The diffusion matrix is then used to update the node STIs

$$\mathbf{v}' = \mathbf{D}\mathbf{v}$$

and the STI values are rescaled to the interval $[\text{minSTI}, \text{maxSTI}]$.

In both the rent and wage stage and in the diffusion stage, the total STI and LTI funds of the system each separately form a conserved quantity: in the case of diffusion, the vector \mathbf{v} is simply the total STI times a probability vector. To maintain overall system funds within homeostatic bounds, a mid-cycle tax and rent-adjustment can be triggered if necessary; the equations currently used for this are

- $\langle \text{Rent} \rangle = \frac{\text{recent stimulus awarded before update} \times \langle \text{Wage} \rangle}{\text{recent size of AF}}$;
- $\text{tax} = \frac{x}{n}$, where x is the distance from the current AtomSpace bounds to the center of the homeostatic range for AtomSpace funds;
- $\forall i: s_i = s_i - \text{tax}$

Investigation of Convergence Properties

Now we investigate some of the properties that the above ECAN equations display when we use an ECAN defined by them as an associative memory network in the manner of a Hopfield network.

We consider a situation where the ECAN is supplied with memories via a “training” phase in which one imprints it with a series of binary patterns of the form $\mathbf{P} = [p_1, \dots, p_n]$, with $p_i \in \{0,1\}$. Noisy versions of these patterns are then used as cue patterns during the retrieval process.

We obviously desire that the ECAN retrieve the stored pattern corresponding to a given cue pattern. In order to achieve this goal, the ECAN must converge to the correct fixed point.

Theorem: For a given value of e in the STI rent calculation, there is a subset of hyperbolic decision functions for which the ECAN dynamics converge to an attracting fixed point.

Proof: Rent is zero whenever $s_i \leq \frac{\text{recentMaxSTI}}{20}$, so we consider this case first. The updating process for the rent and wage stage can then be written as $f(s) = s + \text{constant}$. The next stage is governed by the hyperbolic decision function

$$g(s) = \frac{\tanh(\text{shape}(s - \text{FocusBoundary})) + 1}{2}$$

The entire updating sequence is obtained by the composition $(g \circ f)(s)$, whose derivative is then

$$(g \circ f)' = \frac{\text{sech}^2(f(s)) \cdot \text{shape}}{2} \cdot (1),$$

which has magnitude less than 1 whenever $-2 < \text{shape} < 2$. We next consider the case $s_i > \frac{\text{recentMaxSTI}}{20}$. The function f

now takes the form

$$f(s) = s - \frac{\log(20s/\text{recentMaxSTI})}{2} + \text{constant},$$

and we have

$$(g \circ f)' = \frac{\text{sech}^2(f(s)) \cdot \text{shape}}{2} \cdot \left(1 - \frac{1}{s}\right),$$

which has magnitude less than 1 whenever $|\text{shape}| < \frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI} - 20}$. Choosing the shape parameter to

satisfy $0 < \text{shape} < \min\left(2, \frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI} - 20}\right)$ then

guarantees that $|(g \circ f)'| < 1$. Finally, $g \circ f$ maps the closed interval $[\text{recentMinSti}, \text{recentMaxSTI}]$ into itself, so applying the Contraction Mapping Theorem completes the proof.

Definition and Analysis of Variant 2

The ECAN variant described above has performed completely acceptably in our experiments so far; however we have also experimented with an alternate variant, with different convergence properties. In Variant 2, the dynamics of the ECAN are specifically designed so that a certain conceptually intuitive function serves as a Liapunov function of the dynamics.

At a given time t , for a given Atom indexed i , we define two quantities: $OUT_i(t)$ = the total amount that Atom i pays in rent and tax and diffusion during the time- t iteration of ECAN; $IN_i(t)$ = the total amount that Atom i receives in diffusion, stimulus and welfare during the time- t iteration of ECAN. Note that welfare is a new concept to be introduced below. We then define $\text{DIFF}_i(t) = |IN_i(t) - OUT_i(t)|$; and define $\text{AVDIFF}(t)$ as the average of $\text{DIFF}_i(t)$ over all i in the ECAN.

The design goal of Variant 2 of the ECAN equations is to ensure that, if the parameters are tweaked appropriately, AVDIFF can serve as a (deterministic or stochastic, depending on the details) Liapunov function for ECAN dynamics. This implies that with appropriate parameters the ECAN dynamics will converge toward a state where $\text{AVDIFF}=0$, meaning that no Atom is making any profit or incurring any loss. It must be noted that this kind of convergence is not always desirable, and sometimes one might want the parameters set otherwise. But if one wants the STI components of an ECAN to converge to some

specific values, as for instance in a classic associative memory application, Variant 2 can guarantee this easily.

In Variant 2, each ECAN cycle begins with rent collection and welfare distribution, which occurs via collecting rent via the Variant 1 equation, and then performing the following two steps. Step A: calculate X , defined as the positive part of the total amount by which AVDIFF has been increased via the overall rent collection process. Step B: redistribute X to needy Atoms as follows: *For each Atom z , calculate the positive part of $(OUT - IN)$, defined as $deficit(z)$. Distribute $(X + e)$ wealth among all Atoms z , giving each Atom a percentage of X that is proportional to $deficit(z)$, but never so much as to cause $OUT < IN$ for any Atom (the welfare being given counts toward IN).* Here $e > 0$ ensures AVDIFF decrease; $e = 0$ may be appropriate if convergence is not required in a certain situation.

Step B is the welfare step, which guarantees that rent collection will decrease AVDIFF. Step A calculates the amount by which the rich have been made poorer, and uses this to make the poor richer. In the case that the sum of $deficit(z)$ over all nodes z is less than X , a mid-cycle rent adjustment may be triggered, calculated so that step B will decrease AVDIFF. (I.e. we cut rent on the rich, if the poor don't need their money to stay out of deficit.)

Similarly, in each Variant 2 ECAN cycle, there is a wage-paying process, which involves the wage-paying equation from Variant 1 followed by two steps. Step A: calculate Y , defined as the positive part of the total amount by which AVDIFF has been increased via the overall wage payment process. Step B: exert taxation based on the surplus Y as follows: *For each Atom z , calculate the positive part of $(IN - OUT)$, defined as $surplus(z)$. Collect $(Y + e1)$ wealth from all Atom z , collecting from each node a percentage of Y that is proportional to $surplus(z)$, but never so much as to cause $IN < OUT$ for any node (the new STI being collected counts toward OUT).*

In case the total of $surplus(z)$ over all nodes z is less than Y , one may trigger a mid-cycle wage adjustment, calculated so that step B will decrease AVDIFF. I.e. we cut wages since there is not enough surplus to support it.

Finally, in the Variant 2 ECAN cycle, diffusion is done a little differently, via iterating the following process: If AVDIFF has increased during the diffusion round so far, then choose a random node whose diffusion would decrease AVDIFF, and let it diffuse; if AVDIFF has decreased during the diffusion round so far, then choose a random node whose diffusion would increase AVDIFF, and let it diffuse. In carrying out these steps, we avoid letting the same node diffuse twice in the same round. This algorithm does not let all Atoms diffuse in each cycle, but it stochastically lets a lot of diffusion happen in a way that maintains AVDIFF constant. The iteration may be modified to bias toward an average decrease in AVDIFF.

The random element in the diffusion step, together with the logic of the rent/welfare and wage/tax steps, combines to yield the result that for Variant 2 of ECAN dynamics, AVDIFF is a stochastic Lyapunov function. The details of

the proof of this will be given elsewhere due to space considerations but the outline of the argument should be clear from the construction of Variant 2. And note that by setting the e and $e1$ parameter to 0, the convergence requirement can be eliminated, allowing the network to evolve more spontaneously as may be appropriate in some contexts; these parameters allow one to explicitly adjust the convergence rate.

One may also derive results pertaining to the meaningfulness of the attractors, in various special cases. For instance, if we have a memory consisting of a set M of m nodes, and we imprint the memory on the ECAN by stimulating m nodes during an interval of time, then we want to be able to show that the condition where precisely those m nodes are in the AF is a fixed-point attractor. However, this is not difficult, because one must only show that if these m nodes and none others are in the AF, this condition will persist. Rigorous proof of this and related theorems will appear in a follow-up paper.

Associative Memory

We have carried out experiments gauging the performance of Variant 1 of ECAN as an associative memory, using the implementation of ECAN within OpenCog, and using both the conventional and Storkey Hebbian updating formulas. Extensive discussion of these results (along with Variation 2 results) will be deferred to a later publication due to space limitations, but we will make a few relevant comments here.

As with a Hopfield net memory, the memory capacity (defined as the number of memories that can be retrieved from the network with high accuracy) depends on the sparsity of the network, with denser networks leading to greater capacity. In the ECAN case the capacity also depends on a variety of parameters of the ECAN equations, and the precise unraveling of these dependencies is a subject of current research. However, one interesting dependency has already been uncovered in our preliminary experimentation, which has to do with the size of the AF versus the size of the memories being stored.

Define the size of a memory (a pattern being imprinted) as the number of nodes that are stimulated during imprinting of that memory. In a classical Hopfield net experiment, the mean size of a memory is usually around, say, .2-.5 of the number of neurons. In typical OpenCog associative memory situations, we believe the mean size of a memory will be one or two orders of magnitude smaller than that, so that each memory occupies only a relatively small portion of the overall network.

What we have found is that the memory capacity of an ECAN is generally comparable to that of a Hopfield net with the same number of nodes and links, if and only if the ECAN parameters are tuned so that the memories being imprinted can fit into the AF. That is, the AF threshold or (in the hyperbolic case) shape parameter must be tuned so that the size of the memories is not so large that the active nodes in a memory cannot stably fit into the AF. This

tuning may be done adaptively by testing the impact of different threshold/shape values on various memories of the appropriate size; or potentially a theoretical relationship between these quantities could be derived, but this has not been done yet. This is a reasonably satisfying result given the cognitive foundation of ECAN: in loose terms what it means is that ECAN works best for remembering things that fit into its focus of attention during the imprinting process.

Interaction between ECANs and other OpenCog Components

Our analysis above has focused on the associative-memory properties of the networks, however, from the perspective of their utility within OpenCog or other integrative AI systems, this is just one among many critical aspects of ECANs. In this final section we will discuss the broader intended utilization of ECANs in OpenCog in more depth.

First of all, associative-memory functionality is directly important in OpenCogPrime because it is used to drive concept creation. The OCP heuristic called “map formation” creates new Nodes corresponding to prominent attractors in the ECAN, a step that (according to our preliminary results) not only increases the memory capacity of the network beyond what can be achieved with a pure ECAN but also enables attractors to be explicitly manipulated by PLN inference.

Equally important to associative memory is the capability of ECANs to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items (Atoms). For example, one key role of ECANs in OCP is to guide the forward and backward chaining processes of PLN (Probabilistic Logic Network) inference. At each step, the PLN inference chainer is faced with a great number of inference steps from which to choose; and a choice is made using a statistical “bandit problem” mechanism that selects each possible inference step with a probability proportional to its expected “desirability.” In this context, there is considerable appeal in the heuristic of weighting inference steps using probabilities proportional to the STI values of the Atoms they contain. One thus arrives at a combined PLN/EAN dynamic as follows:

1. An inference step is carried out, involving a choice among multiple possible inference steps, which is made using STI-based weightings (and made among Atoms that LTI weightings have deemed valuable enough to remain in RAM)
2. The Atoms involved in the inference step are rewarded with STI and LTI proportionally to the utility of the inference step (how much it increases the confidence of Atoms in the system’s memory)
3. The ECAN operates, and multiple Atom’s importance values are updated
4. Return to Step 1 if the inference isn’t finished

An analogous interplay may occur between ECANs and the MOSES procedure learning algorithm that also plays a key role in OCP.

It seems intuitively clear that the same attractor-convergence properties highlighted in the present analysis of associative-memory behavior, will also be highly valuable for the application of ECANs to attention allocation. If a collection of Atoms is often collectively useful for some cognitive process (such as PLN), then the associative-memory-type behavior of ECANs means that once a handful of the Atoms in the collection are found useful in a certain inference process, the other Atoms in the collection will get their STI significantly boosted, and will be likely to get chosen in subsequent portions of that same inference process. This is exactly the sort of dynamics one would like to see occur. Systematic experimentation with these interactions between ECAN and other OpenCog processes is one of our research priorities going forwards.

References

- Amit, Daniel (1992). *Modeling Brain Function*. Cambridge University Press.
- Goertzel, Ben (2006). *The Hidden Pattern*. Brown Walker.
- Goertzel, Ben (2007). *Virtual Easter Egg Hunting*. In *Advances in Artificial General Intelligence*, IOS Press.
- Goertzel, Ben (2008). *OpenCogPrime: Design for a Thinking Machine*, online at <http://www.opencog.org/wiki/OpenCogPrime:WikiBook>
- Goertzel, Ben, Matthew Ikle’, Izabela Goertzel and Ari Heljakka. *Probabilistic Logic Networks*. Springer.
- Hutter, Marcus (2004). *Universal AI*. Springer.
- Looks, Moshe (2006). *Competent Program Evolution*. PhD thesis in CS department, Washington University at St. Louis
- Storkey A.J. (1997) *Increasing the capacity of the Hopfield network without sacrificing functionality*, ICANN97 p451-456.
- Storkey, Amos (1998). *Palimpsest Memories: A New High Capacity Forgetful Learning Rule for Hopfield Networks*.
- Storkey A.J. and R. Valabregue (1999) *The basins of attraction of a new Hopfield learning rule*, Neural Networks 12 869-876.

A formal framework for the symbol grounding problem

Benjamin Johnston and Mary-Anne Williams

University of Technology, Sydney

Broadway, Ultimo 2007, Australia

johnston@it.uts.edu.au

Abstract

A great deal of contention can be found within the published literature on grounding and the symbol grounding problem, much of it motivated by appeals to intuition and unfalsifiable claims. We seek to define a formal framework of representation grounding that is independent of any particular opinion, but that promotes classification and comparison. To this end, we identify a set of fundamental concepts and then formalize a hierarchy of six *representational system classes* that correspond to different perspectives on the representational requirements for intelligence, describing a spectrum of systems built on representations that range from symbolic through iconic to distributed and unconstrained. This framework offers utility not only in enriching our understanding of symbol grounding and the literature, but also in exposing crucial assumptions to be explored by the research community.

Introduction

The symbol grounding problem [1] represents a long standing (and often misunderstood) point of contention within the Artificial Intelligence community (e.g., [2,3,4]) and continues to concern researchers exploring Artificial General Intelligence¹ (AGI). The problem, as it is classically conceived, concerns the nature of the abstract symbols used in computer systems, how they may be seen as having a real-world meaning, and how that meaning can be made intrinsic to a system.

Consider the problem of a knowledge base designed to reason about the possible security threats posed by terrorist organizations. The system may have an internal symbol *nuclear_weapon* that we as humans understand as representing the real-world concept of nuclear weapons. However, to a purely symbolic computer system, the symbol *nuclear_weapon* is nothing more than an arbitrary token that has no more *intrinsic* meaning than any other symbol in a computer, say *waffle_blah*. The symbol grounding problem concerns the question of how the meaning of symbols can be embedded into a system, and grounding is said to be the process of ensuring that these abstract symbols have meaning.

While there is the philosophical question of whether a machine really can have intrinsically ground symbols (indeed, this is the motivation for Searle's Chinese Room argument), the symbol grounding problem poses the more practical question of whether a purely symbolic system *could* solve problems that apparently demand deep intelligence and understanding. Is it possible, for example, for a purely symbolic system to understand the nuanced relationship between a nuclear weapon and a dirty bomb, or to explain that a zebra is like a horse with stripes, or even to determine what other letter of the alphabet an upside-down 'M' resembles; without requiring the specific answers to these questions to be explicitly given to the system in advance?

Our objective is not to argue a specific position on the symbol grounding problem, but rather, to provide the first formal

framework for the symbol grounding problem. Our approach offers standardized terminology for discussing assumptions and ideas and also raises important new research questions. In particular, we aim to allow an AI researcher to express their assumptions regarding the symbol grounding problem as elegantly as a computer scientist might use computational complexity classes to motivate the need for heuristics in preference to brute force search. Furthermore, we hope that our framework will direct future arguments about grounding away from appeals to intuition and toward a greater emphasis on formalizable and falsifiable claims.

In this paper, we will first define symbol systems and representations, and review the problem of grounding such symbols and representations. We then introduce our formal notation and explore 'semantic interpretability'. These serve as preliminaries for the primary contribution of this paper: the definition of our representational system classes and their correspondences with the published literature. We then conclude with some observations about the representational classes and their relevance, including a brief overview of future research directions.

Symbol Systems and Symbol Grounding

Harnad [1] first proposed the symbol grounding problem as a question concerning semantics: "How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads?" While Harnad's original formulation of the problem is largely philosophical, his motivation is clearly of a pragmatic nature: he implicitly assumes that the property of 'intrinsic interpretability' is crucial for intelligence. We therefore prefer to reformulate the symbol grounding problem in more straightforward terms: "Is it possible to use formal symbolic reasoning to create a system that is intelligent?" Of course, this reformulation presupposes a definition of what it means to be or to appear intelligent—but the reformulation is an improvement in the sense that it brings us closer to something objectively measurable.

Harnad saw the mechanisms of an isolated formal symbol system as analogous to attempting to learn Chinese as a second language from a Chinese-Chinese dictionary. Even though characters and words are defined in terms of other characters, reading the dictionary would amount to nothing more than a 'merry-go-round' passing endlessly from one symbol string (a term) to another (its definition); never coming to a 'halt on what anything meant' [1]. He therefore argued that since symbols only refer to other symbols in a symbol system, there is no place where the symbols themselves are given meaning. The consequence of this is that it is impossible for a formal symbol system to distinguish between any two symbols except using knowledge that has been explicitly provided in symbolic form. This, in the view of Harnad, limits the comprehension and capabilities of a symbolic system in the same way that a non-speaker armed with a Chinese-Chinese dictionary may manage to utter random syntactically correct sentences, but would exhibit extremely poor performance in understanding real conversation.

¹ For example, consider recent debate on an AGI email list: www.mail-archive.com/agi@v2.listbox.com/msg07857.html

Of course, Harnad’s argument is not universally accepted by computer scientists. One objective of this paper is to explore the problem, so we will use our representational system classes to outline and classify the diversity of opinions later in this paper.

The symbol grounding problem concerns symbolic systems, but what *is* a formal symbol system? Harnad [1] provides eight criteria:

A symbol system is: (1) a set of arbitrary “physical tokens” (scratches on paper, holes on a tape, events in a digital computer, *etc.*) that are (2) manipulated on the basis of “explicit rules” that are (3) likewise physical tokens and strings of tokens. The rule-governed symbol-token manipulation is based (4) purely on the shape of the symbol tokens (not their ‘meaning’), *i.e.*, it is purely syntactic, and consists of (5) ‘rulefully combining’ and recombining symbol tokens. There are (6) primitive atomic symbol tokens and (7) composite symbol-token strings. The entire system and all its parts—the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules—are all (8) ‘semantically interpretable’: the syntax can be systematically assigned a meaning (*e.g.*, as standing for objects, as describing states of affairs).

It is interesting to note that criteria 1–7 can be used to describe any universal or Turing-complete language. However, Harnad is not claiming that meaning or intelligence is incomputable—he proposes his own computational framework for solving the symbol grounding problem. It is the 8th criterion of a formal symbol system that defines the essential point of difference between his conception of symbolic systems and representations used in arbitrary computation. The requirement of interpretability in criterion 8 is intended to capture the essence of familiar symbolic systems such as logical theorem proving and rule based systems, and to exclude highly distributed, ‘holographic’ or connectionist representations that do not behave in a symbolic manner (even though they operate on a digital computer). For example, while connectionist approaches to building intelligent systems can be framed so as to meet criteria 1–7, connectionist methods do not typically allow for a systematic assignment of real-world interpretation to hidden layer neurons (*i.e.*, hidden neurons learn with a bias towards performance rather than any particular ‘meaning’), they therefore do not satisfy criterion 8, and are therefore not (directly) subject to Harnad’s criticism.

Harnad’s 8th criteria for a symbol system is essential to understanding the symbol grounding problem. We will consider how changes to this criterion (whether in its phrasing or in comprehension) influence an understanding of the symbol grounding problem. Specifically, we further generalize the symbol grounding problem as follows: “What kinds of reasoning can be performed by systems constrained by different representational criteria?” In this formulation, we can regard different research groups as working from both different assumptions of what constitutes intelligence (*i.e.*, the kind of reasoning) and different representational constraints.

Notational Preliminaries

Problems We begin by assuming the existence of a set, \mathcal{P} , that contains all problems that may be posed to an intelligent system. Each problem is a declarative sentence (in some formal language) about the world and an agent is said to be able to solve a problem if its determination of the truth of that statement matches the ‘real world’ truth. A problem might be a query

posed by a person to a theorem prover or a question-answering system, or it might represent an encoding of the inputs and outputs of a robotic system (*i.e.*, “given certain sensory inputs x , the appropriate behavior at time t , is to perform action a ”). While a real life agent may encounter complex situations and exhibit nuanced performance that is neither success nor failure, we assume that these situations can be analyzed as a large set of binary sub-problems, and the agent’s performance is a measure of how many of the sub-problems can be successfully solved. If an agent, f , believes statement p , we denote² this as $f \vdash p$. If an agent, f , can correctly solve a problem, $p : \mathcal{P}$, then we denote this as $f \sim p$.

Problem Sets We define a *problem-set* as a set of problems: an object of type $\mathbb{P}(\mathcal{P})$. An agent, f , can solve a problem-set, $ps : \mathbb{P}(\mathcal{P})$, if it can solve all problems within that set. This is denoted $f \sim ps$, and we have $f \sim ps \Leftrightarrow \forall p : ps \bullet f \sim p$.

Intelligence We use problem-sets to define intelligence. In this paper, we do not choose any particular definition of intelligence: we assume a range of definitions so that we can not only denote the largely subjective and unformalizable ‘I’ll know it when I see it’ attitude of many AI researchers towards intelligence, but also offer scope for formal definitions of intelligence. As such, a given definition, I , of intelligence is a set of problem-sets; *i.e.*, $I : \mathbb{P}(\mathbb{P}(\mathcal{P}))$. An agent, f , is considered intelligent with respect to a definition of intelligence I , if it can solve *all* problems in *some* problem-set. This is denoted $f \sim I$, and we therefore have $f \sim I \Leftrightarrow \exists ps : I \mid \forall p : ps \bullet f \sim p$.

This approach to representing definitions of intelligence, admits many definitions beyond that of simple IQ tests or fixed checklists of skills. Consider that how one may regard Albert Einstein and Elvis Presley as both possessing exceptional intelligence, even though their genius is expressed in different ways. Their distinct skills correspond to different problem-sets within our common interpretation of ‘genius’.

We allow many definitions of intelligence; for example:

- A set $I_{\text{Harnad}} : \mathbb{P}(\mathbb{P}(\mathcal{P}))$ for those systems that Harnad would regard as exhibiting intelligence,
- A set $I_{\text{IQ}=100} : \mathbb{P}(\mathbb{P}(\mathcal{P}))$ to denote sets of problems that a person of average Human intelligence would be able to solve,
- A set $I_{\text{Market}} : \mathbb{P}(\mathbb{P}(\mathcal{P}))$ of buying decisions that a trading agent would need to solve successfully in order to exceed break-even on a market over a particular time interval,
- Given a formal definition of intelligence with a precise threshold, we may have a set $I_{\text{Formal}} : \mathbb{P}(\mathbb{P}(\mathcal{P}))$ denoting those problem-sets that a formally intelligent system could solve.

Formal Systems We define \mathcal{F} as the set of all finite formal systems that satisfy criteria 1–7 of Harnad and are finitely realizable³. We define \mathcal{T} as the universal set of symbols, and assume

- 2 Throughout this paper, we use the Z notation per international standard ISO/IEC 13568:2002. We treat typing as equivalent to membership in a set and denote this with a colon. The power-set operator is denoted as \mathbb{P} , the set of natural numbers as \mathbb{N} , the set of partial functions from A to B as $A \twoheadrightarrow B$, and the domain and range of a function, f , as $\text{dom}(f)$ and $\text{ran}(f)$ respectively.
- 3 By finitely realizable, we mean that the systems’s representations and computational processes that can be described in finite space on a Turing machine by a finite agent within the universe, and that the corresponding computations of the system in solv-

that each formal system comprises a set of fixed symbolic transition rules and a dynamic execution state. We assume (without loss of generality) that an execution trace of a formal system, f , on a problem, p , is comprised of a two-dimensional grid of symbols. We denote this as $\mathbf{t}(f, p) : \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T}$. The two axes of the grid correspond to the state of the system (analogous to a CPU clock) and the position or address of each symbol. The value of each grid-cell is the single symbol stored in the ‘address’ in that state (be it a byte value stored in RAM, a mark on a tape, a neural network weight, or an ‘atom’ in a logical programming language).

Representational Units In most non-trivial systems, the individual symbols do not convey meaning alone; the intelligent behavior stems from the manipulation of entire subsequences or subsets of the system’s symbolic state. Furthermore, such intelligent behavior stems from the manipulation of only certain possible subsets: those subsets of the system state that correspond to legal guards and arguments of the system’s transition rules. For example, if the numbers 1, 25 and 334 are denoted as the fixed-length sequence of digits <001025334> at a given step of the system trace, then a system’s transition rules might only accept sequences aligned to three-digit boundaries (*i.e.*, 001, 025 and 334, but neither 00102 nor 253). For a given formal symbolic system $f : \mathcal{F}$, and problem $p : \mathcal{P}$, we define the set of all representational units, $\mathbf{a}(f, p) : \mathbb{P}(\mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T})$ as the set of all subsets of the system trace, $\mathbf{t}(f, p)$, that can match part or all of a guard or parameter of a transition rule in f .

Semantic Interpretability

‘Semantic interpretability,’ the cornerstone of Harnad’s 8th criteria for a symbol system, presents a challenge to the formalization of the symbol grounding problem. Indeed, we believe that the philosophical difficulties of the symbol grounding problem lie in the elusiveness of ‘semantic interpretability’.

The model-theoretic approach to defining semantic interpretability would be to assume some valuation function, m , that maps from symbols to ‘real world’ counterparts so that the problems that a formal system believes to be true correspond to truths that follow from their real world valuations. That is, if we assume the existence of a universal set, \mathcal{U} , containing the complete universe of actual, possible, conjectured and imaginary objects, actions, categories, relations and concepts in the ‘real world,’ then given a formal system, f , we may attempt to formalize semantic interpretability in a manner such as the following⁴:

$$\exists m : \mathcal{P} \rightarrow \mathcal{U} \mid \forall p : \mathcal{P} \bullet f \vdash p \Rightarrow m(p) \vDash p$$

However, such a definition is clearly not what was intended by Harnad; it merely states that the agent has true beliefs for every problem it can solve. Model theoretic methods do not directly apply because problems of intelligence are already assumed to be statements about the ‘real world’. Semantic interpretability of a formal system demands inspection of not only its internal symbols, but also the *use* of the symbols. For example, a system that uses both constructive proof and proof-by-contradiction may use the same symbol to denote a concept and its negation: it the use of the symbol in reasoning that reveals the true meaning of the symbol.

ing a problem occur in finite time.

4 We introduce the semantic entailment operator, $u \vDash p$, to denote that proposition, p , is (or would be) true in every universe consistent with the set u .

Unfortunately, it is impossible to analyze use without defining a particular computational model (and our goal is to retain a level of abstraction from such particulars). In future works, we intend to explore such philosophical challenges of defining semantic interpretability, especially given symbol use. We propose here a working definition.

Let SI denote the type of semantic interpretations of representational units $SI = \mathbb{P}(\mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T}) \rightarrow \mathcal{U}$. Then, given a (model-theoretic) semantic interpretation $m : SI$, that maps from a set of representational units, $r : \mathbb{P}(\mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T})$, to elements of \mathcal{U} ; we say that a formal system, f , in solving a problem, p , is semantically interpretable if *syntactic entailment* (*i.e.*, computation) corresponds to *semantic entailment* from the model implied by the conjunction of the semantic mapping of the system’s entire execution trace. *i.e.*;

$$\begin{aligned} & \mathbf{si}(m, f, p) \\ \Leftrightarrow & (f \vdash p \Leftrightarrow \mathbf{t}(f, p) \subseteq (\cup \text{dom}(m)) \wedge \\ & \{u \mapsto e : m \mid u \subseteq \mathbf{t}(f, p) \bullet e\} \vDash p) \end{aligned}$$

While this working definition ignores the internal use of symbols and may be overly stringent for any particular system, we do not believe it limits the generality of our work. Representational units with different purposes may be expanded to include the neighboring section markers, delimiters, type definitions, annotations or positional information that indicates their purpose: thereby embedding the *use* of a representational unit in the formal system into its surface structure.

The nature and existence of ‘real world’ concepts, and consequently, the membership of the set \mathcal{U} remains an open question that bears upon the symbol grounding problem and the work we describe here. We have assumed that the ‘real world’ universe includes concepts such as historical, hypothetical and imaginary entities, as well as attributes, verbs and abstract nouns like *up*, *walk*, *happy* and *beauty*. However, one could trivially ‘solve’ the symbol grounding problem on a technicality by excessively generalizing \mathcal{U} , so that the ‘real world’ concept of any symbol can be taken as “those situations, entities and environments that stimulate the generation of the symbol”. Such contrived entities would seem absurd to a human-observer and are also highly context dependent, so therefore do not correspond to our intuitions of meaningful ‘real world’ entities that belong in \mathcal{U} . The relationship between the nature of \mathcal{U} and symbol grounding is an interesting problem, that we plan to explore in future work.

Representational System Classes

We can now use our notions of and notation for semantic interpretability to formalize the differences between attitudes toward the symbol grounding problem. Our goal is to analyze the space of finite formal systems, \mathcal{F} , and categorize these into a hierarchy based on the semantic interpretability of their representations. That is, we assume Harnad’s criteria 1–7, and build a hierarchy from specializations and generalizations of Harnad’s 8th criteria.

For example, $SIR_{\mathcal{U}}$ is one such class intended to denote the set of systems with fully Semantically Interpretable Representations. We can use this class to restate Harnad’s thesis that a symbolic system cannot be intelligible as follows:

$$\forall f : SIR_{\mathcal{U}} \bullet \neg (f \sim I_{\text{Harnad}})$$

Or, if we extend the ‘solves’ operator to representational classes so that $c \sim i \Leftrightarrow \exists f : c \bullet f \sim i$, then we have Harnad’s thesis as:

$$\neg SIR_{\mathcal{U}} \sim I_{\text{Harnad}}$$

By exploring variations of Harnad’s definition of symbolic systems, we have identified a range of representational system classes beyond $SI\mathcal{R}$. In particular, we have identified six representational system classes that appear to capture the philosophical position of many AI researchers, and that form (by their definition) a hierarchy of representational expressiveness. Each class represents a set of formal systems and is therefore of the type $\mathbb{P}(\mathcal{F})$.

The following subsections describe the classes ordered from most restrictive to most general. Each class has been defined so that it subsumes (*i.e.*, is a super-set of) those classes that have been presented before it. The subsumption property can be proven syntactically from the formal definitions in this work, and holds irrespective of the foundational assumptions of this work.

3.1 Context-Free Semantically Interpretable Representation

$CF\mathcal{SIR}$: Every symbol must have a unique semantic interpretation.

$$CF\mathcal{SIR} = \{f: \mathcal{F} \mid \exists m: SI \mid \forall p: \mathcal{P} \bullet si(m, f, p) \wedge \forall r: \text{dom}(m) \bullet \#r = 1\}$$

Systems in $CF\mathcal{SIR}$ are those in which every single symbol has some meaning (given by valuation function, m): symbols do not acquire meaning from their context in some larger representational unit such as a sentence or structure.

A system that, for example, uses the symbol *Mouse* to represent the common house mouse, but also uses that same symbol in the context of a Disney movie to state that Mickey Mouse is a mouse *could not* be regarded as making use of symbols with universal and unambiguous meanings (consider, for example, posing the system the question of whether a mouse can speak English). In a symbolic system with context-free semantic interpretations, such distinctions would first need to be translated into separate symbols: *e.g.*, *Natural_Mouse* and *Cartoon_Mouse*. Whether complete translations are possible for all symbolic systems remains an open question, and is, in fact, the question of whether $SI\mathcal{R} = CF\mathcal{SIR}$.

Systems in $CF\mathcal{SIR}$ include:

1. Semantic web systems based on RDF: every resource is denoted by a globally unique URL that is intended to capture some unique context-free interpretation. RDF provides no facility to contextualize the truth of an RDF triple without complex reification [5].
2. Traditional database systems: in typical database designs, each record is intended to have a unique and context-free interpretation.
3. Internal representations of industrial robotic systems: every variable in the control system of an industrial robot can be assigned a unique meaning (*e.g.*, joint position, current distance sensor reading, x-coordinate of a recognized widget).

3.2 Semantically Interpretable Representation

$SI\mathcal{R}$: Every representational unit must have a semantic interpretation.

$$SI\mathcal{R} = \{f: \mathcal{F} \mid \exists m: SI \mid \forall p: \mathcal{P} \bullet si(m, f, p) \wedge \text{dom}(m) \subseteq a(f, p)\}$$

The set $SI\mathcal{R}$ corresponds to those systems that match Harnad’s original definition of formal symbolic systems. Every represen-

tational unit in the system must have a semantic interpretation, and every symbol used by the system belongs to a representational unit.

Systems in $SI\mathcal{R}$ (but not $CF\mathcal{SIR}$) include:

1. John McCarthy’s early proposal to incorporate context into formal symbolic systems [6], and related efforts that have arisen from this, such as PLC and MCS [7].
2. The CYC project’s symbolic engineering wherein symbols have meaning, and that meaning is given within context spaces [8].

3.3 Iconic and Symbolic Representation

$IS\mathcal{R}$: Representational units may have semantic interpretation. Non-interpretable representational units must be composable as sets that in aggregate have semantic interpretation and resemble their meaning.

$$IS\mathcal{R} = \{f: \mathcal{F} \mid \exists m: SI \mid \forall p: \mathcal{P} \bullet si(m, f, p) \wedge \text{iconic}(\text{dom}(m) - a(f, p))\}$$

In $IS\mathcal{R}$, individual representational units need not have a semantic interpretation, but may be part of an aggregate that is semantically interpretable as a whole. Such aggregations in $IS\mathcal{R}$ must have a structure that somehow resembles the meaning of their referent (*e.g.*, by projection or analogy)—they must be iconic.

For example, the individual pixels of a high-resolution image could not typically be regarded as having a particular meaning when considered individually, but in aggregate may be understood as denoting the object that they depict. A system with hybrid visual/symbolic representations could refer to its symbolic knowledge to answer factual queries, but use high resolution images to compute answers to queries about nuanced physical traits or to compare the appearances of different people. Iconic representations in some way resemble their meaning: be they low-level resemblances such as images, 3D models and perspective invariant features, or more abstract forms such as graphs representing the social networks in an organization or the functional connections of components.

Precisely what, then, does it mean for a symbol to resemble its meaning? If a system resembles its meaning, then a small representational change should correspond to a small semantic change. That is, for a set of iconic representations, i , there should exist a computable representational distance function, \mathbf{rdist} , and a semantic distance function (with some real world meaning, and therefore a member of \mathcal{U}), \mathbf{sdist} , and error limit, ϵ , such that:

$$\text{iconic}(i)$$

\Leftrightarrow

$$\forall i_1, i_2: i \bullet |\mathbf{rdist}(i_1, i_2) - \mathbf{sdist}(m(i_1), m(i_2))| \leq \epsilon$$

Systems in $IS\mathcal{R}$ include:

1. Harnad’s [1] proposed ‘solution’ to the symbol grounding problem via the use of visual icons.
2. The Comirit project that combines ‘imaginative’ graph-based iconic representation and reasoning with the deductive reasoning of a logical theorem prover [9].
3. Reasoning performed within Gärdenfors’ conceptual spaces framework, especially as a mechanism for embedding greater ‘semantics’ into symbolic systems such as the Semantic Web [10]. The cases or prototypes of a case-based reasoner may also be regarded as a similar form of iconic representation.
4. Setchi, Lagos and Froud’s [11] proposed agenda for com-

putational imagination.

3.4 Distributed Representation

\mathcal{DR} : *Representational units may have semantic interpretation. Non-interpretable representational units must be composable as sets that in aggregate have semantic interpretation.*

$$\mathcal{DR} = \{f : F \mid \exists m : SI \mid \forall p : \mathcal{P} \bullet si(m, f, p)\}$$

Every element of the set \mathcal{DR} is a finite system that makes use of two kinds of representations: those that can be systematically assigned meaning, and those that only have meaning in aggregate (and may be of arbitrary form). That is, \mathcal{DR} requires semantic interpretability, but does not require that the units of semantic interpretation correspond to the same representational units that are manipulated by the rules of the formal system.

Consider, for example, a neural network that has been trained to identify the gender of a human face. Some of the network's output nodes may be specifically trained to activate in the presence of masculine features: these output nodes, in addition to the hidden layer neurons that feed into the output nodes, may in aggregate be seen as meaning 'facial masculinity'. Even though it may be impossible to assign a coherent semantic interpretation to the representations and values of the hidden layer neurons that the formal system manipulates, the aggregated network can be seen as capturing specific real-world meaning.

Examples of systems that make representational assumptions or restrictions consistent with \mathcal{DR} include:

1. Hybrid systems wherein neural networks have been trained under supervised learning to recognize symbols of a higher level symbolic reasoning processes. Indeed, all forms of supervised machine learning where the internal structure of the induced representations are regarded as a black box would be consistent with the restrictions of \mathcal{DR} .
2. Neural-symbolic systems that, for example, perform symbolic reasoning within connectionist mechanisms (e.g., [12]).

3.5 Unconstrained Representation

\mathcal{UR} : *Representational units may or may not have any particular semantic interpretation.*

$$\mathcal{UR} = F$$

Every element of the set \mathcal{UR} corresponds to a problem-set that may be solved by a finite formal system (i.e., a Turing-complete machine). The set \mathcal{UR} therefore corresponds to the capabilities of computational systems, irrespective of whether their internal representations can be assigned particular semantic interpretations.

Examples of systems that make representational assumptions or restrictions consistent with \mathcal{UR} include:

1. Neural-network-based systems, in which output or activity is triggered entirely by arbitrary connectionist processes (e.g., [13,14]). In such systems, input nodes correspond to raw sensory data, output nodes are motor commands corresponding to actions, and internal hidden nodes are trained without regard to the development of meaningful cognitive symbols (i.e., black-box intelligence): none of these nodes can be seen as capturing meaningful semantic interpretations.
2. Universal computable models of intelligence such as AI_{ξ}^{it} [15]. Such approaches emphasise computation or modelling that maximizes a reward function without regard

for the semantic interpretability of the computational processes (though there is an implicit assumption that the most successful representations are likely to be those that best capture the environment and therefore are likely to acquire semantic interpretability in the limit).

3. Reactive systems, such as those concrete implementations of Brooks [16]. Such systems do not attempt to explicitly model the world (or may only partially model the world), and so lack semantic interpretability.

3.6 Non-Formal

\mathcal{NF} : *Representation units may or may not have any particular semantic interpretation, and may be manipulated by rules (such as interaction with the environment or hyper-computational systems) that are beyond formal definition.*

$$\mathcal{NF} = F \cup F^*$$

The class \mathcal{NF} extends \mathcal{UR} with a set of 'enhanced' formal symbolic systems, F^* —systems with distinguished symbols that are connected to the environment⁵. While problems associated with action in the physical environment may already be found in the set \mathcal{P} , and these may already be solved by systems of other representational system classes (such as \mathcal{UR}), the set \mathcal{NF} includes those systems that use embodiment directly as part of its deductive processes: systems where the environment is 'part' of the reasoning, rather than merely the 'object' of a solution. \mathcal{NF} encompasses systems that, for example, need the environment to generate truly random sequences, to perform computations that aren't finitely computable on a Turing machine but may be solved by physical systems, to exploit some as-yet-unknown quantum effects, to build physical prototypes, or more simply, to solve problems about objects and complex systems that simply cannot be described or modelled in sufficient detail on a realizable computer system.

Examples of systems and research that make representational assumptions or restrictions consistent with \mathcal{NF} include:

1. Embodied robotics in the true spirit of Brooks' vision, that treat 'the world [as] its own best model' and that refute the possibility of a disembodied mind [16]. Such work regards direct sensory experience and manipulation of the physical environment throughout problem solving as an essential part of the intelligent thought: that intelligence has co-evolved with the environment and sensory abilities; that it is not sufficient merely to have a reactive system; but that higher order intelligence arises from the complex interactions between reactivity and the environment. Note however, *actual* reactive robots/systems to date would, in fact, be better classified in \mathcal{UR} (as we have done) because they do not yet operate at a level of interaction beyond primitive reactivity.
2. Models of intelligence and consciousness that are not Turing-computable or constrained by Gödel's incompleteness theorem. Examples of these may be found in work such as that of Penrose [17] postulating significant (but currently unspecified) quantum effects on intelligent thought and consciousness.

⁵ For example, we can allow a Turing machine to interact with the environment by reserving a segment of its tape as 'memory mapped' I/O. Symbols written to this segment of the tape will manipulate actuators and sensory feedback is itself achieved by a direct mapping back onto symbols of the I/O segment of the tape.

Discussion

The representational system classes not only serve to clarify many of the loose and ambiguous concepts that appear in debate on symbol grounding, but offer many other benefits: a language for rapidly communicating assumptions; a tool for analyzing the symbol grounding problem and generating new research assumptions; and a framework for better understanding underlying assumptions and the inter-relationships between assumptions. For example, Harnad's claims may be succinctly summarized as $\neg SIR \sim I_{\text{Harnad}} \wedge ISR \sim I_{\text{Harnad}}$.

Simple proof techniques can show that the representational classes form a hierarchy (i.e., $CFSIR \subseteq SIR \subseteq ISR \subseteq DR \subseteq UR \subseteq NF$), and it follows that the combined sets of problems that each class may solve also forms a hierarchy (i.e., we have a hierarchy of intelligence). However, it remains an interesting question whether this hierarchy is strict: are there classes of representational systems C_1 and C_2 such that $C_1 \subseteq C_2$ but there exists some definition of intelligence I where $\neg C_1 \sim I$, and $C_2 \sim I$ (we denote this, $C_1 < C_2$). i.e., is C_2 strictly more intelligent than C_1 ? Our intuitions are that this is indeed the case for our hierarchy, and we plan to show this in future work. Here, we briefly outline our reasons for believing so:

- $CFSIR < SIR$, because even though the context-sensitive symbols of SIR could be systematically mapped into sets of context-free symbols in $CFSIR$ (e.g., **Mouse** \rightarrow **Cartoon_Mouse**), the potentially unbounded regress of contexts may make it impossible to ensure that this mapping remains finite when problem-sets are unbounded (i.e., it can be done for any *particular* problem, but not *in general*, in advance of knowledge of the problem).
- $SIR < ISR$, following the arguments of Harnad.
- $ISR < DR$, because we believe that there are pathological concepts that *emerge* from complex chaotic systems so that iconic representations of structure or appearance hinder rather than enhance performance (i.e., systems in which emergence is crucial to understanding the global system behavior, but for which properties of emergence cannot be predicted from local or structural analysis).
- $DR < UR$, because we believe that there are pathological situations where an attempt to analyze the situation into concepts diminishes the ability to learn appropriate behaviors (compare this to the manner in which human beings 'discover' false patterns in randomized data, hindering their ability to make optimal decisions using that data).
- $UR < NF$, because even though the universe may be formally computable, it may not be possible for any agent *situated within* the universe to describe the universe in sufficient detail such that a Turing machine could compute the solution to all 'intelligent' problems.

Finally, we are careful to emphasise again that we do not claim to have *solved* the problem. Instead, our framework reduces the symbol grounding to two long-standing philosophical challenges: the selection and definition of intelligence, I , and the problem of the nature of 'meaningful' entities in the universe (i.e., the set \mathcal{U} , and consequently how to define $si(m, f, p)$). While our framework does not yet offer precise guidance towards solving these sub-problems, it provides straightforward machinery by which the symbol grounding problem can be understood in such terms. Our contribution lies in formalizing the

connections between sub-problems, and thereby narrowing the ambiguity in the problem and closing opportunities for circular reasoning.

Conclusion

By defining a formal framework of representation grounding, we help clarify the contention in important work on symbol grounding as stemming from arguments about different kinds of representational system classes. We have proposed six classes to this end: $CFSIR \subseteq SIR \subseteq ISR \subseteq DR \subseteq UR \subseteq NF$. These capture many perspectives on the symbol grounding problem: the classes have significant power both for explanation and investigation. Not only can future research use these classes to quickly express assumptions, but the abstractions assist in the exploration of the problem, the classification and comparison of existing work, and provide machinery for the development of novel conjectures and research questions.

References

1. Harnad, S. 1990. 'The symbol grounding problem', *Physica D*, 42:335-346.
2. Coradeschi, S. and Saffiotti A. (eds) 2003. *Robotics and Autonomous Systems*, 43(2-3).
3. Williams, M-A., Gärdenfors, P., Karol, A., McCarthy, J. and Stanton, C. 2005. 'A framework for evaluating the groundedness of representations in systems: from brains in vats to mobile robots', *IJCAI 2005 Workshop on Agents in Real Time and Dynamic Environments*.
4. Ziemke, T. 1997. 'Rethinking grounding', *Proceedings of New Trends in Cognitive Science, 1997*, Austrian Society for Computer Science.
5. Bouquet, P., Serafini, L. and Stoermer, H. 2005. 'Introducing context into RDF knowledge bases', *SWAP 2005*.
6. McCarthy, J. 1993. 'Notes on formalizing context', *IJCAI 1993*.
7. Serafini, L. and Bouquet, P. 2004. 'Comparing formal theories of context in AI', *Artificial Intelligence*, 155(1): 41-67.
8. Lenat, D. 1998. 'The dimensions of context space', *Cycorp Technical Report*.
9. Johnston, B. and Williams, M-A. 2008. 'Comirit: Commonsense reasoning by integrating simulation and logic', *AGI 2008*, 200-211.
10. Gärdenfors, P. 2004. 'How to make the semantic web more semantic', In Varzi, A. and Lieu, L. (eds) *Formal Ontology in Information Systems*, 17-34, IOS Press.
11. Setchi, R., Lagos, N. and Froud, D. 2007. 'Computational imagination: research agenda', *Australian AI 2007*, 387-393.
12. Kilkerry Neto, A., Zaverucha, G. and Carvalho, L. 1999. 'An implementation of a theorem prover in symmetric neural networks', *IJCNN 1999*, 6: 4139-4144.
13. Markram, H. 2006. 'The Blue Brain project', *Nature Reviews: Neuroscience*, 7: 153-160.
14. de Garis, H., Tang, J-Y., Huang, Z-Y., Bai, L., Chen, C., Chen, S., Guo, J-F., Tan, X-J., Tian, H., Tian, X-H., Wu, X-J., Xiong, Y., Yu, X-Q. and Huang, D. 2008. 'The China-Brain project: building China's artificial brain using an evolved neural net module approach', *AGI 2008*, 107-121.
15. Hutter, M. 2000. 'Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions', *EMCL 2001*, 226-238.
16. Brooks, R. 1991. 'Intelligence without reason', *IJCAI 1991*, 569-595.
17. Penrose, R. 1999. *The Emperor's New Mind*, Oxford University Press.

A Cognitive Map for an Artificial Agent

Unmesh Kurup

Rensselaer Polytechnic Institute
Carnegie 013, 110 8th Street
Troy, NY 12180
kurupu@rpi.edu

B. Chandrasekaran

The Ohio State University
591 Dreesse, 2015 Neil Ave
Columbus, OH 43210
chandra@cse.ohio-state.edu

Abstract

We show how a general-purpose cognitive architecture augmented with a general diagrammatic component can represent and reason about Large-scale Space. The diagrammatic component allows an agent built in this architecture to represent information both symbolically and diagrammatically as appropriate. Using examples we show (a) how the agent's bimodal representation captures its knowledge about large-scale space as well as how it learns this information while problem solving and (b) the agent's flexibility when it comes to using learned information and incorporating new information in solving problems involving large-scale space.

Introduction

An agent based on a general purpose cognitive architecture has the ability to work on a range of problems. Agents based on task-specific structures are usually restricted to particular task and problem domains. The advantages of task-specific knowledge such as faster solution times can however be realized in the general architecture case through the use of general-purpose learning mechanisms that account for the formation of task-specific knowledge from the general underlying representations. Newell has made a comprehensive case for the development of unified general architectures for solving the problem of general human cognition (Newell 1990). Soar (Laird, Newell et al. 1987) and ACT-R (Anderson and Lebiere 1998) have been two of the architectures that have tried to rise to the challenge of generality. For our purposes it is not their differences that are important but what they share. Their central representational framework is symbolic, or more precisely, predicate-symbolic. In the predicate-symbolic view, the agent's knowledge, goals etc are represented in terms of symbol structures that describe the world of interest in terms of properties of and relations between individuals in the world.

Cognitive models built in Soar and ACT-R have been very effective in showing how a general purpose cognitive

architecture can produce task-optimal behavior, and how learning provides efficiencies over time and across tasks. However, work using these architectures has tended to focus more on certain kinds of tasks over others. Among the tasks that have received less attention are those that deal with the representation of and reasoning about large-scale space. As it happens, for any given spatial reasoning task, a model builder can represent the task-specific spatial information symbolically, treating the extraction of such symbolic information to perceptual processes outside the architecture. On the other hand, it has been proposed that there are certain forms of perceptual representation that belong inside the cognitive architecture itself (Chandrasekaran 2002). In this paper, we demonstrate how an agent based on a general cognitive architecture, albeit one with an additional diagrammatic representation, can represent and reason about space. We further show the agent can learn during problem solving and show transfer of learning within and between tasks.

Representing and Reasoning about Large-Scale Space

In 1948, (Tolman 1948) proposed that animals have an internal representation of large-scale space which he called the cognitive map. In 1960, (Lynch 1960) produced his seminal study of the environment in which he identified *landmarks, routes, nodes, districts* and *edges* as the features that are important in building a cognitive map. Since then there have been a number of models, both descriptive models without commitment to mechanisms, and computational models that propose mechanisms, which have been proposed to account for various phenomena associated with the representation of space (add refs). A variety of behavioral/psychological studies have also aided the development of these models by providing a set of characteristics or behaviors that a model should possess. Chief among them is the understanding that

the cognitive map is less of a map and more of a collage (Tversky 1993). That is, representations of large-scale spaces are not holistic but stored in pieces and that these pieces are brought together as needed during problem solving. Such a representation allows the agent to be flexible with respect to representing inconsistent and incomplete information.

Knowledge of large-scale space can come from multiple sources. The most common, of course, being personal experience of navigation in space. We automatically build representations of our environment as we traverse them. A second, and important, source is maps. Our knowledge of large environments, such as the spatial extent and geographical locations of the fifty states of the USA, originated from our use of maps. Representations, originating from either source, are combined and modified in various ways and for various purposes during problem solving. In this paper, we focus on spatial reasoning tasks that involve an external map rather than the agent moving about in space. This is because biSoar, being built on Soar, is a theory of high-level cognition, and navigating in the world requires perception of the external world (as well as motor systems to act on it), capabilities which biSoar, and cognitive architectures in general, are not intended to capture. However, we believe our proposal can be suitably extended in the case when such abilities become available.

Generality comes at a price. Currently, biSoar agents cannot outperform more task-specific proposals for representing and reasoning about space. Instead, our focus in this paper is in showing how agents built in the biSoar architecture are flexible and versatile. Using examples we show how information about large-scale space can be represented in a piece-meal fashion in biSoar's underlying bimodal representation. We then show how an agent, during the course of problem solving, learns these pieces of information. We use two sets of examples to show biSoar's flexibility and versatility. In the first one, we show how information learned by an agent in one task (route-finding) can be used to solve problems in a different but similar task (geographic recall). In the second example, we show how the agent can incorporate information from multiple sources during an episode of route-finding.

biSoar

To create biSoar (Kurup and Chandrasekaran 2006), a general-purpose cognitive architecture, Soar was augmented with the Diagrammatic Reasoning System (DRS), a domain-independent system for representing diagrams (Chandrasekaran, Kurup et al. 2005). The diagrammatic component of the state, encoded in DRS, is part of the agent's *internal* representation, just as the predicate symbolic component is in an agent's state representation in current theories of cognitive architecture. The content can come from various sources, recall from memory, imagination by composing elements from memory, or from an external image of a diagrammatic representation. DRS of an external diagram is an

abstraction of the external diagram: regions in the image intended to be points are abstracted in DRS as points but with the same location of the intended point, regions intended to be curves are abstracted into the intended curves, symbolic annotations abstracted to symbols associated with the DRS element, and so on. The perceptual routines operate on the DRS elements, whether they were generated internally or from external representations. Of course, creating a DRS corresponding to an external diagram requires image processing routines, such as those that do background subtraction or edge detection, to go from an image to a collection of objects with their spatial extents. Such external processing, however, is not part of the theory of the bimodal internal state, nor of the operation of the cognitive architecture. In our examples that involve an agent interacting with an external map, we assume that such image processing routines are available to the agent and focus on how the result of these routines, the diagrammatic representation, is represented and manipulated in spatial reasoning tasks

While a physical diagram (on a screen or on paper) is an image that contains diagrammatic objects, each to be interpreted as a *point*, *curve* or a *region*, the diagram is viewed as a configuration of diagrammatic objects. Note too that while in the physical diagram all the objects are regions, so that they can be perceived, DRS captures the intended diagram. If an object in the physical diagram appears as a circle, in DRS it would be treated as a Euclidean point object with just location to characterize it. DRS is domain-independent – the only objects are points, curves or regions. Interpreting them in domain terms is the job of the user of the representation. The objects in DRS have associated with them information about their spatiality -- locations for point objects, and representations that are functionally equivalent to the sets of points that constitute the objects for curves and regions. Associated with the DRS are a set of perception and diagram construction/modification capabilities; following (Ullman 1984), these are called *routines*. All these routines are visual, but we use the more general term so that it will apply to the more general multi-modal view.

Perception Routines take diagrammatic elements as arguments and return information about specified spatial properties or spatial relations. There are two types of perception routines: the ones in the first type re-perform the figure-ground separation on the image – rather than on the DRS – perceiving emergent objects (e.g., the two sub-regions that emerge when a curve intersects a region.) Routines of the second type return specified spatial properties of objects, e.g., the length of a curve; and evaluate specified spatial relations between objects, e.g., Inside(Region1, Region2). These routines work from descriptions in DRS. DRS thus is an intermediate representation that supports reconstituting the image, a capability needed for emergent object identification, and also the perceptual routines that perceive properties of and relations between objects.

Routines that help in constructing or modifying the diagram are *action routines*. They create diagrammatic objects that satisfy specific perceptual criteria, such as “a curve object that intersects a given region object,” and “a point object inside the region object.” The sets of routines are open-ended, but routines that are useful across a number of domains are described in (Chandrasekaran, Kurup et al. 2004), which also contain more information on DRS.

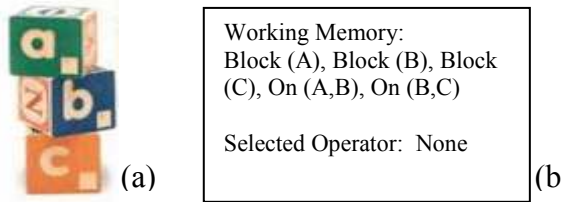


Figure 1: (a) Blocks World and (b) Soar's representation of the world in (a).

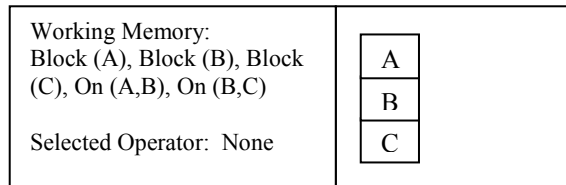


Figure 2: biSoar representation of the world shown in 1(a)

Cognitive State in Soar

Soar's representations are predicate-symbolic. The cognitive state in Soar is represented by the contents of Soar's WM and operator, if any, that has been selected. Fig 1(b) shows Soar's cognitive state representation of the blocks world example in 1(a).

Cognitive State in biSoar

The cognitive state in biSoar is bimodal – it has both symbolic and diagrammatic parts. Fig 2 shows the bimodal representation of the world depicted in Fig 1(a). Working memory in biSoar is represented as a quadruplet, with each Identifier, Attribute, Value triplet augmented with a diagrammatic component in DRS that represents the visualization (metrical aspect) of the triplet. Since not all triplets need to be (or can be) visualized, the diagrammatic components are present only as needed. States represent the current or potential future state of interest in the world and the symbolic and the diagrammatic part may represent related or distinct aspects of the world. However, the diagrammatic representation is “complete” in a way that the symbolic representation is not. For example, from the symbolic representation alone it is not possible to say without further inference whether A is above C. But the same information is available for pick up in the diagram with no extra inference required. This has advantages (for instance in dealing with certain aspects of the Frame Problem) and disadvantages (over-specificity).

Bimodal LTM and Chunking

There are two questions that have to be answered in an implementation of Long Term Memory (LTM) – how are elements put into LTM (i.e., learned) and how are elements retrieved from LTM. In the case of Soar the answers to these two questions are chunking for learning and a matching process that matches the LHS of a LTM rule to WM for retrieval.

Chunking - Chunking simply transfers the relevant contents of WM to LTM. In the case of biSoar, chunking transfers to LTM both symbolic and diagrammatic elements present in WM.

Matching - In the case of Soar the retrieval process is straightforward because matching (or even partial matching when variables are present) symbols and symbol structures to each other is an exact process; either they match or they don't. When the cognitive state is bimodal, WM has metrical elements in addition to symbols. Matching metrical elements to each other (say a curve to another curve) is not an exact process since two metrical elements are unlikely to be exactly the same. Matching metrical elements would require a different approach like a non-exact process that can match roughly similar elements in a domain-independent manner (since the matching should be architectural). It may also turn out that only calls to perceptual routines are present in LTM while matching metrical elements is a more low-level cognitive process present only in stimulus-response behavior. For now we take the latter approach where the LHS of biSoar rules contain only symbol structures while the RHS contains calls to the diagram that execute perceptual routines. The results of executing these routines appear as symbol structures in the symbolic side at the end of a decision cycle. We think that this approach can account for many of the diagrammatic learning capabilities that are required in

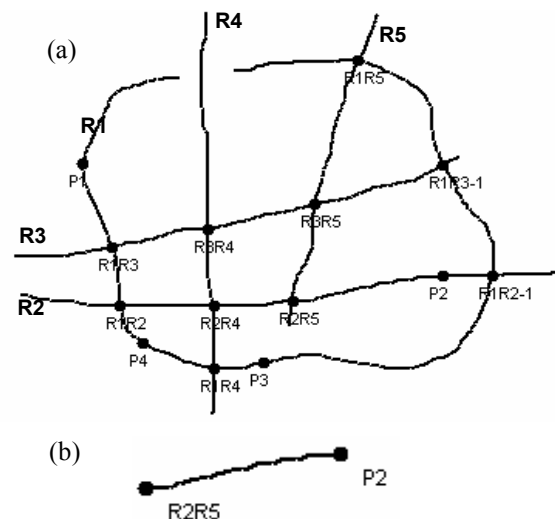


Figure 3: (a) A map of main highways in Columbus, OH showing routes R1...R5 and locations P1...P4. Intersections of routes also form additional locations. (b) The DRS representation of the route from R2R5 to P2

models of cognition except in cases where goal specifications contain irreducible spatial components, such as might be the case in the problem solving of a sculptor.

1. locate the starting & destination locations in the map
2. make the starting location the current location
3. Find the routes on which the current location lies
4. For each route, find the directions of travel
5. for each route and direction of travel, find the next location
6. calculate the Euclidean distance between these new locations and the destinations
7. pick the location that is closest to the destination and make that the current point
8. repeat 3-8 until destination is reached

Figure 4: The wayfinding strategy used by the biSoar agent

Representing Large-Scale Space in BiSoar

Soar’s LTM is in the form of rules. Each rule can be thought of as an if-then statement where the condition (if) part of the rule matches against the existing conditions in WM and the action (then) part of the rule describes the changes to be made to WM. Thus, Soar’s LTM is arranged to respond to situations (goals) that arise as part of problem solving. This makes sense because, ideally, a majority of an agent’s rules are learned as part of problem solving and hence in response to a particular situation. If the situation (or a similar one) arises in the future, Soar can now use this learned rule in response. For the route-finding scenario, the agent has knowledge about various locations on the map and about routes between these locations, presumably as a result of learning from previous problem solving episodes. The agent’s knowledge of the area consists of bimodal rules in the following form:

If goal is find_destination and at location A and traveling in direction Dx on route Rx, then destination is location B, diagram is DRSx

Where DRSx represents the spatial extent of the section of the route that runs from Lx to Ly along route Rx and is represented using DRS. So, for example, for the map in Fig 3(a), the spatial relationship between locations R2R5 and P2 would be expressed as

If goal is find_destination and at R2R5 and traveling Right on Route R2, then destination is P2, diagram is DRS1

Where DRS1 is shown in Fig 3(b). The directions available are Right, Left, Up and Down though this choice of directions is arbitrary. For convenience, we represent the above information in a more concise form as follows:

$$Gx, Lx, Dx, Rx \rightarrow Ly, DRSx$$

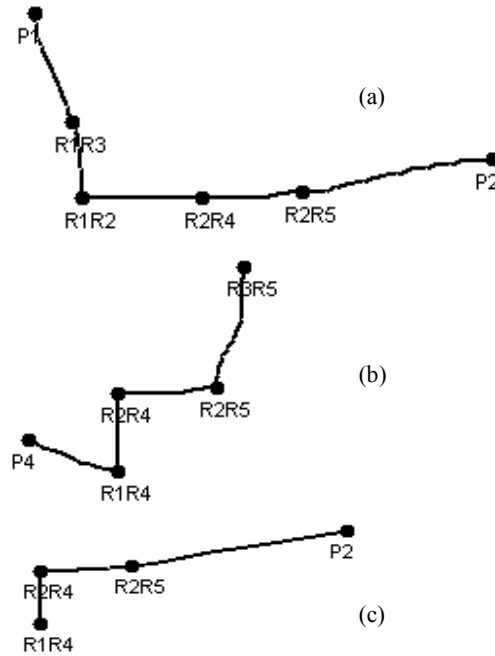


Figure 5: Routes found by the agent from (a) P1 to P2 b) P4 to R3R5 and (c) R1R4 to P2

Route-finding & Learning Using BiSoar

We created an agent to perform route-finding tasks given a map using the simple strategy shown in Fig 4. The agent finds a route by locating the starting and destination points and finds a path by moving to the next point on the route that is the closest to the destination using a simple Euclidean distance measure. Some of the steps of the algorithm require information from the diagrammatic component. This information is of two types - in certain cases it is symbolic, such as the answer to the query “On what route(s) is the point located?” while in other cases, the information is diagrammatic, like for the question “What’s the route between the point P1 and R1R3?”

Each step of the algorithm is implemented such that it becomes a sub-goal for the biSoar agent. As a result, when the agent solves a sub-goal, the architecture automatically learns a chunk (rule) that captures the solution to the task. For example, corresponding to step 5 of the algorithm, the agent finds that if you move down from P1 along route R1, you reach R1R3. The next time the agent is faced with this sub-goal, the chunk that was learned is used to answer it. Fig 5(a) shows the path found by the agent from P1 to P2. Table 1 shows the information learned by the agent as a result of that task. Fig 5(b) similarly shows the result of route-finding by the agent between the locations P4 and R3R5 and Table 2, the information learned by the agent. Within-task Transfer – To show within-task transfer of learning, we ran the agent on route-finding tasks with an external map. As a result of learning, the agent acquired a number of chunks. The external map was then removed and the agent was giving a new route-finding task, one in

which the starting and destination points were present in the previous tasks but never as a pair between which a route had to be found. For example, with the external map available, the agent was asked to find routes from P1 to P2 and from P4 to R3R5. The external map was then removed and the agent was asked to find a route from R1R4 to P2. The agent using information learned during the previous two tasks (Tables 1 & 2), found the route in Fig 5(c).


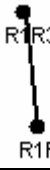
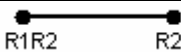
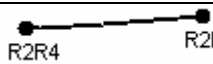
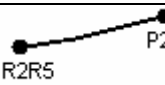
Gon,P1 → R1	
Gon,R1R3 → R1,R3	
Gon,R1R2 → R1,R2	
Gon,R2R4 → R2,R4	
Gon,R2R5 → R2,R5	
Gdir,P1,R1 → up,down	
Gdir,R1R3,R1 → up,down	
Gdir,R1R2,R2 → right,left	
Gdir,R2R4,R2 → right, left	
Gdir,R2R5,R2 → right,left	
Gdest,P1,R1,down → R1R3	
Gdest,R1R3,R1,down → R1R2	
Gdest,R1R2,R2,right → R2R4	
Gdest,R2R4,R2,right → R2R5	
Gdest,R2R5,R2,right → P2	

Table 1: rules learned by the agent in finding the route from P1 to P2

Between-task Transfer – To show between-task transfer, we had an agent trained on one type of task (route-finding) perform a different spatial task (geographic recall). As an example, the agent in the within-task transfer example, with the knowledge in tables 1 and 2, was asked to find the geographic relationship between two points that it had encountered during the tasks. The agent’s strategy was to recreate (not just recall) the map using the learned information, and extract the relationship between the locations from the re-created map. Fig 6 shows the map created by the agent.

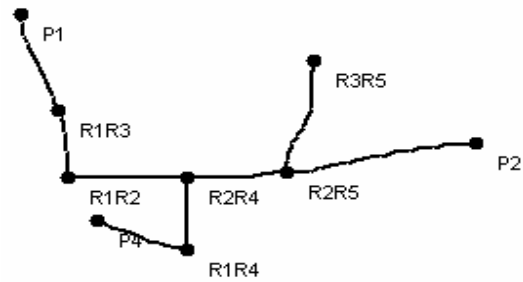


Figure 6: Map created by the agent for geographic recall

Predicting Paths – To show how the agent can incorporate information from multiple sources, we gave the biSoar agent an incomplete version of the earlier map (as shown in Fig 7) with route R5 missing. The agent was also given the symbolic information that there is a path from R2R5 in the up direction, but because the diagram was incomplete it is not clear what the destination of traveling along that route is. The goal of the agent was to find a route from P2 to P3 taking into account possible paths. Fig 8 shows the route found by the agent. The strategy shown in Fig 4 is slightly modified so that the agent makes predictions about destinations. In step 5 of the strategy, when the agent checks the external representation to find the destination from R2R5 on R5 in the “up” direction, it finds that the information is not available in the map. In response, the agent creates a straight line path from R2R5 in that

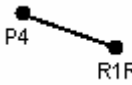


Gon,P4 → R1	
Gon,R1R4 → R1,R4	
Gdir,P4,R1 → left,right	
Gdir,R1R4,R4 → up	
Gdir,R2R5,R5 → up	
Gdest,P4,R1,right → R1R4	
Gdest,R1R4,R1,up → R2R4	
Gdest,R2R5,R2,up → R3R5	

Table 2: rules learned by the agent in finding the route from P4 to R3R5

direction and sees that it intersects R3. It names this point INT1 and proposes a route that goes through INT1. Comparing the map in Fig 3(a) and the route found in Fig

7(b), we can see that the straight line assumption by the agent results in a slightly different route R5 than what would have been found using the complete map.

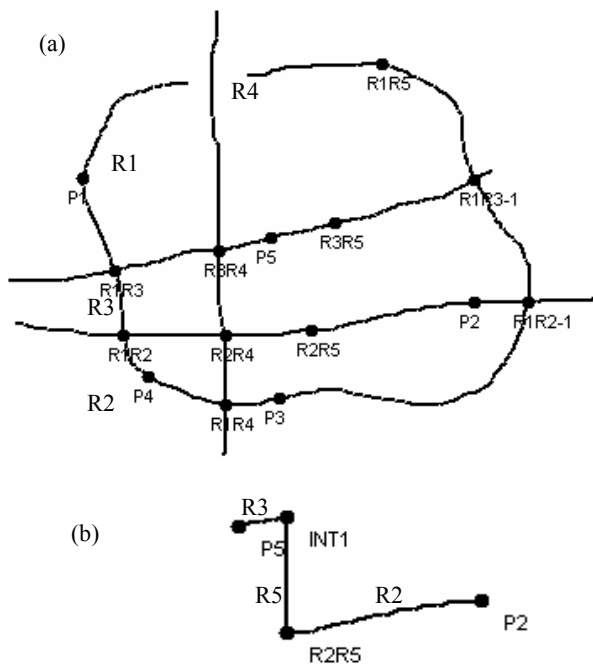


Figure 7: (a) Map without route R5 for the predicting paths task. (b) Route found between P2 and P5

Conclusion

Representations of large-scale space in general purpose architectures are usually limited to topological graph-like representations due to constraints imposed by the underlying predicate-symbolic representation. Reasoning, most commonly route-finding, then proceeds via the application of a graph-traversal algorithm. In our bimodal architecture, biSoar, large-scale space is represented using both symbolic and diagrammatic representations. This bimodal representation provides a richer representational format that can be used to solve a wider range of spatial reasoning tasks. At the same time, the diagrammatic representations are not specific to large-scale space but part of a more general approach to understanding cognition as multi-modal. In the case of large-scale space reasoning, such an approach has the following benefits. First, it captures not only the topological but also the metrical aspects of space. Depending on the task, either or both of the representations can be used to solve the problem. Second, an agent in this architecture can learn both symbolic and diagrammatic elements via chunking. This information can then be used to solve similar and related tasks. Third, the agent's representation of the space is not holistic in nature. Instead it is spread over a number of rules and smaller diagrammatic pieces. This allows the agent to function under the presence of inconsistencies as well as include information from multiple sources during

problem solving. The lack of a consistent single representation also makes it easier for the agent since it does not have to maintain consistency as new information comes in. In these respects, the agent's representation is similar to human spatial representation. Lastly, the presence of a metrical representation allows the agent to reason about space in a way that topological representations cannot, namely, in predicting destinations of paths or finding shortcuts or novel paths.

Acknowledgements

Advanced Decision Architectures Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAD19-01-2-0009.

References

- Anderson, J. R., and Lebiere, C. 1998. *The Atomic Components of Thought*, Lawrence Erlbaum Associates.
- Chandrasekaran, B. 2002. Multimodal Representations as Basis for Cognitive Architecture: Making Perception More Central to Intelligent Behavior. *Intelligent Information Processing*, Kluwer Academic Publishers.
- Chandrasekaran, B.; Kurup, U.; Banerjee, B. 2005. A Diagrammatic Reasoning Architecture: Design, Implementation and Experiments. AAAI Spring Symposium *Reasoning with Mental and External Diagrams: Computational Modeling and Spatial Assistance*.
- Chandrasekaran, B.; Kurup, U.; Banerjee B.; Josephson, J.; Winkler, Robert. 2004. An Architecture for Problem Solving with Diagrams. *Diagrammatic Representation and Inference conference*, Springer-Verlag.
- Kurup, U., and Chandrasekaran, B. 2006. Multi-modal Cognitive Architectures: A Partial Solution to the Frame Problem. *28th Annual Conference of the Cognitive Science Society*, Vancouver.
- Laird, J. E.; Newell, A.; Rosenbloom, P. 1987. Soar: an architecture for general intelligence. *Artificial Intelligence* 33(1): 1-64.
- Lynch, K. 1960. *The Image of the City*. Cambridge, MIT Press.
- Newell, A. 1990. *Unified theories of cognition*. Harvard University Press.
- Tolman, E. C. 1948. *Cognitive Maps in Rats and Man*. *Psychological Review* 55: 189-208.
- Tversky, B. 1993. Cognitive maps, cognitive collages, and spatial mental model. *Spatial information theory: Theoretical basis for GIS*. U. Frank and I. Campari. Heidelberg-Berlin, Springer-Verlag: 14-24.
- Ullman, S. 1984. *Visual Routines*. *Cognition* 18: 97-159.

Claims and Challenges in Evaluating Human-Level Intelligent Systems

*John E. Laird**, *Robert E. Wray III***, *Robert P. Marinier III**, *Pat Langley****

*Division of Computer Science and Engineering, University of Michigan, Ann Arbor, MI 48109-2121

**Soar Technology, Inc., 3600 Green Court, Suite 600, Ann Arbor, MI 48105

***School of Computing and Information, Arizona State University, Tempe, AZ 85287-8809

Abstract

This paper represents a first step in attempting to engage the research community in discussions about evaluation of human-level intelligent systems. First, we discuss the challenges of evaluating human-level intelligent systems. Second, we explore the different types of claims that are made about HLI systems, which are the basis for confirmatory evaluations. Finally, we briefly discuss a range of experimental designs that support the evaluation of claims.

Introduction

One of the original goals of Artificial Intelligence (AI) was to create systems that had general intelligence, able to approach the breadth and depth of human-level intelligence (HLI). In the last five years, there has been a renewed interest in this pursuit with a significant increase in research in cognitive architectures and general intelligence as indicated by the first conference on Artificial General Intelligence. Although there is significant enthusiasm and activity, to date, evaluation of HLI systems has been weak, with few comparisons or evaluations of specific claims, making it difficult to determine when progress has been made. Moreover, shared evaluation procedures, testbeds, and infrastructure are missing. Establishing these elements could bring together the existing community and attract additional researchers interested in HLI who are currently inhibited by the difficulty of breaking into the field.

To confront the issue of evaluation, the first in a series of workshops was held in October 2008 at the University of Michigan, to discuss issues related to evaluation and comparison of human-level intelligent systems. This paper is a summarization of some of the discussions and conclusions of that workshop. The emphasis of the workshop was to explore issues related to the evaluation of HLI, but to stop short of making proposals for specific evaluation methodologies or testbeds. That is our ultimate goal and it will be pursued at future workshops. In this first workshop, we explored the challenges in HLI evaluation, the claims that are typically made about HLI, and how those claims can be evaluated.¹

¹For an in depth and more complete discussion of evaluation of AI systems in general, see Cohen (1995).

Challenges in Evaluating HLI Systems

Defining the goal for HLI

One of the first steps in determining how to evaluate research in a field is to develop a crisp definition its goals, and if possible, what the requirements are for achieving those goals. Legg and Hutter (2007) review a wide variety of informal and formal definitions and tests of intelligence. Unfortunately, none of these definitions provide practical guidance in how to evaluate and compare the current state of the art in HLI systems.

Over fifty years ago, Turing (1950) tried to finesse the issue of defining HLI by creating a test that involved comparison to human behavior, the Turing Test. In this test, no analysis of the components of intelligence was necessary; the only question was whether or not a system behaved in a way that was indistinguishable from humans. Although widely known and popular with the press, the Turing Test has failed as a scientific tool because of its many flaws: it is informal, imprecise, and is not designed for easy replication. Moreover, it tests only a subset of characteristics normally associated with intelligence, and it does not have a set of incremental challenges that can pull science forward (Cohen, 2005). As a result, none of the major research projects pursuing HLI use the Turing Test as an evaluation tool, and none of the major competitors in the Loebner Prize (an annual competition based on the Turing Test) appear to be pursuing HLI.

One alternative to the Turing Test is the approach taken in cognitive modeling, where researchers attempt to develop computational models that think and learn similar to humans. In cognitive modeling, the goal is not only to build intelligent systems, but also to better understand human intelligence from a computational perspective. For this goal, matching the details of human performance in terms of reaction times, error rates, and similar metrics is an appropriate approach to evaluation. In contrast, the goal of HLI research is to create systems, possibly inspired by humans, but using that as a tactic instead of a necessity. Thus, HLI is not defined in terms of matching human

reaction times, error rates, or exact responses, but instead, the goal is to build computer systems that exhibit the full range of the cognitive capabilities we find in humans.

Primacy of Generality

One of the defining characteristics of HLI is that there is no single domain or task that defines it. Instead, it involves the ability to pursue tasks across a broad range of domains, in complex physical and social environments. An HLI system needs broad competence. It needs to successfully work on a wide variety of problems, using different types of knowledge and learning in different situations, but it does not need to generate optimal behavior; in fact, the expectation is it rarely will. This will have a significant impact on evaluation, as defining and evaluating broad competency is more difficult than evaluating narrow optimality.

Another aspect of generality is that, within the context of a domain, an HLI system can perform a variety of related tasks. For example, a system that has a degree of competence in chess should be able to play chess, teach chess, provide commentary for a chess game, or even develop and play variants of chess (such as Kriegsspiel chess). Thus, evaluation should not be limited to a single task within a domain.

Integrated Structure of HLI Systems

Much of the success of AI has been not only in single tasks, but also in specific cognitive capabilities, such as planning, language understanding, specific types of reasoning, or learning. To achieve HLI, it is widely accepted that a system must integrate many capabilities to create coherent end-to-end behavior, with non-trivial interactions between the capabilities. Not only is this challenging from the standpoint of research and development, but it complicates evaluation because it is often difficult to identify which aspects of a system are responsible for specific aspects of behavior.

A further complication is that many HLI systems are developed not by integrating separate implementations of the cognitive capabilities listed earlier, but instead by further decomposing functionality into more primitive structures and process, such as short-term and long-term memories, primitive decision making and learning, representations of knowledge, and interfaces between components, such as shown in Figure 1. In this approach, higher-level cognitive capabilities, such as language processing or planning are implemented in a fixed substrate, differing in knowledge, but not in primitive structures and processes. This is the cognitive architecture approach to HLI development (Langley, Laird, & Rogers, in press), exemplified by Soar (Laird, 2008) and ICARUS (Langley & Choi, 2006).

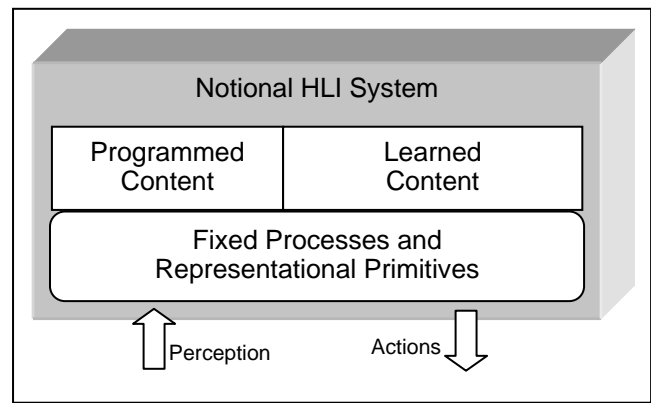


Figure 1: Structure of a Notional HLI System.

This issue is clear in research on cognitive architectures because they make the following distinctions:

- The architecture: the fixed structure that is shared across all higher-level cognitive capabilities and tasks.
- The initial knowledge/content that is encoded in the architecture to achieve capabilities and support the pursuit of a range of tasks.
- The knowledge/content that is learned through experience.

For any HLI system, it is often difficult to disentangle the contributions of the fixed processes and primitives of the systems to the system's behavior from any initial, domain-specific content and the learned knowledge, further complicating evaluation. There is a concern that when evaluating an agent's performance, the quality of behavior can be more a reflection of clever engineering of the content than properties of the HLI systems itself. Thus, an important aspect of evaluation for HLI is to recognize the role of a prior content in task performance and attempt to control for such differences in represented knowledge.

Long-term Existence

Although not a major problem in today's implementations, which typically focus on tasks of relatively short duration, HLI inherently involves long-term learning and long-term existence. It is one thing to evaluate behavior that is produced over seconds and minutes – it is possible to run many different scenarios, testing for the effects of variation. When a trial involves cumulative behavior over days, weeks, or months, such evaluation becomes extremely challenging due to the temporal extents of the experiments, and the fact that behavior becomes more and more a function of experience.

Claims about HLI

We are primarily interested in constructive approaches to HLI; thus, our claims are related to the functionality of our systems, and not their psychological or neurological realism. Achieving such realism is an important scientific goal, but one of the primary claims made by many

practitioners in HLI is that it can be achieved without an exact reimplementation of the human mind and/or brain.

A major step in empirical evaluation is to consider the claims we want or expect to make about the systems we develop. Only by knowing these claims can we define appropriate experiments that test those claims and let us determine what we need to measure in those experiments. An explicit claim (hypothesis) is usually about the relationship between some characteristic of a HLI system and its behavior. To test the hypothesis, a manipulation experiment can be performed in which the characteristic (the independent factor) is varied and changes in behavior along some dimensions (dependent variables) are measured. Many of the difficulties described earlier arise because of the types of claims that we as researchers are attempting to make in the pursuit of HLI.

HLI System Claims

There are varieties of claims that can be made about HLI at the systems level. We highlight four types of claims below:

1. A computer system (HLI1) can achieve some type of behavior or cognitive capability related to HLI. There are many examples of this type of claim from the early history of cognitive architectures. For example, cognitive architectures were used to illustrate capabilities such as associative retrieval and learning, improvements in performance with experience, and the “emergence” of so-called high-level capabilities, such as planning or natural-language comprehension, from the primitives of the architecture. Such a claim is invariably a sufficiency claim, where the structure of the agent is not claimed to be the only way of achieving the desired behavior (a necessity claim). These claims are generally made within the context of some class of environments, tasks, and an agent’s ability to interact with its environment. A few cases where necessity claims have been made about the general properties of architectures such as Newell and Simon’s (1976) symbol system hypothesis.
2. A modification of a system (HLI1’) leads to expanding the set of problems the system can solve or improving behavior along some dimension related to HLI across a range of tasks (see section on dependent variables for a discussion of metrics related to behavior). For example, the progression from version 7 of Soar to version 8 led to improvements in system robustness and learning capability in Soar (Wray & Laird, 2003). This is probably the most common claim, as it is part of the standard practice of systematically improving and extending the capabilities of an HLI system.
3. One system (HLI1) differs from another system (HLI2) in the set of problems that can be solved or in its performance. This claim usually involves comparing two systems and is currently less common, as it involves creating two separate systems and applying them to similar tasks. One notable example of systematic

comparison was the Agent Modeling and Behavior Representation (AMBR) program, sponsored by the Air Force Research Laboratory (Gluck & Pew, 2005). AMBR compared four different architectures on a few tasks in a single task domain. One lesson of AMBR is the importance and difficulty of controlling for a priori content, as suggested previously. The HLI community is capable of descriptive and analytical comparisons of architectures (e.g., see Jones & Wray, 2006) but empirical comparison of architectures and HLI systems (as opposed to example instances in single task domains) is currently infeasible.

4. One system (HLI1) has behavior similar along some relevant dimension to human behavior (H1). This is a special case of 3 above, where human behavior provides the target metric and the emphasis is usually on similarity. Even though we are concentrating on the functionality of HLI systems, humans often provide the best yardstick for comparison. However, even in the cognitive modeling community, evaluation is typically focused on model evaluation rather than evaluation of the underlying system. Anderson and Lebiere (2003) offer suggestions for more systematic evaluation of the paradigm supporting task models, which may also provide a framework for a near-term, descriptive approach to HLI evaluation.

There will often be a hierarchy of claims. Stronger claims are usually reserved for general properties of the architecture, such as that symbol systems are necessary in order to achieve general competence. Claims about the general properties of a specific component in relation to achieving competency will usually be weaker sufficiency claims. One can also make claims about specific algorithms and data structures, such as that the RETE algorithm achieves nearly constant match time even as the number of rules grows (Doorenbos, 1994).

Independent Variables

Central to claims are that there is some relationship among different variables, in particular that varying the *independent* variables leads to changes in the *dependent* variables. In HLI systems, the independent variables often fall in three classes:

- Components of the overall system: As components or modules are added, removed, or modified, it is claimed that there will be changes in behavior. With these types of independent variables, there often is not an ordering – these are categorical and not numeric, so that results are not summarized on a graph in which the values of the dependent variables can be connected with lines.
- Amount of knowledge: Knowledge is varied to determine how effectively the system can use or process knowledge to guide its behavior. One challenge is to compare knowledge across systems, given their different representations. However, within a given architecture, it is usually easy to measure the impact of knowledge by

comparing behavior with and without specific knowledge elements.

- **System parameters:** Many systems have parameters that affect their behavior, such as the learning rate in a reinforcement learning agent. This leads to parametric studies that involve systematic variation of system parameters. The current state-of-art in computational cognitive modeling provides examples of how much parametric exploration is possible and offers glimpses into how those explorations can inform one's evaluation of contributions to overall behavior.

In addition to changes in the HLI systems, many claims concern how changes in the environment or task influence the behavior of an HLI system. For such claims, the independent variables are properties of the environment and task. These are often more difficult to vary systematically.

Examples of environmental independent variables include:

- **Experience in an environment:** For claims related to efficacy of learning, independent variables can be the amount of experience, the time existing in the environment, and related properties.
- **Complexity of the environment:** Analysis of how a system responds as one changes the number of objects, their relations and properties, and the types of interactions between objects.
- **Accessibility of the environment:** The kinds of information can be known/perceived at any time.
- **Indeterminacy in environmental interaction:** The ease of unambiguously sensing and acting in the environment.
- **Dynamics of the environment:** How different aspects of the environment change independently of the HLI system and how fast the environment changes relative to the basic processing rate of the system.

Examples of task-related independent variables include:

- **Whether the task requires satisficing vs. optimizing.** Given the competing constraints on HLI systems, often the goal is to satisfice.
- **Complexity of the task:** How many goals/subgoals must be achieved? What interdependences are there between goals?
- **Length of existence:** How long does the system behave in the environment in ways that put significant stress on its ability to respond quickly to environmental dynamics?

Dependent Variables: Metrics

Dependent variables allow measurement of properties of behavior relevant to evaluating claims. These metrics can be either quantitative or qualitative, and we evaluations will often involve multiple metrics.

We do not consider properties of theories of HLI, such as parsimony, because they relate to claims about theories as opposed to properties of the HLI system.

Our analysis of metrics is split into two parts. The first addresses concrete metrics that directly measure some aspect of behavior, such as solution quality, while the second will address metrics that cover abstract properties of HLI systems that cannot be measured directly, such as robustness and flexibility.

Concrete metrics:

- **Performance** includes measures such as solution time, quality of solution, and whether or not a solution is found. These are the standard metrics used in evaluating AI systems. One must careful when using CPU time because of variation in the underlying hardware. Usually solution time will be in some hardware independent measure (such as nodes expanded in a search) that can then be mapped to specific hardware.
- **Scalability** involves change in some performance variable as problem complexity changes. Scalability is an important metric for HLI systems because of the need for large bodies of knowledge acquired through long-term learning. Other scalability issues can arise from interacting with complex environments where the number of relevant objects varies.

Evaluating only concrete metrics of behavior poses the danger of driving research toward engineering optimizations. Behavioral evaluations should include both a notion of behavior (e.g., learning optimization) and what goes in (level of programming, research, etc.). Current practice is usually just to measure behavior. However, a general claim is that an HLI approach should decrease the amount of re-engineering (what goes in) required for a task. Thus, there are other metrics that are typically independent variables (varied during testing to determine their effect on performance and scalability) but that can become dependent variables if the experiment is set up to determine when a certain level of performance is achieved. For example, one could measure how much knowledge or training is required to achieve a certain level of performance or how much additional knowledge and training (or re-engineering of the architecture) is required to perform on a new task, a property termed *incrementality* (Wray & Lebiere, 2007).

Abstract metrics

Concrete metrics have the advantage that they are usually easy to measure; however, many of the claims about HLI systems are not directly grounded in concrete metrics such as performance measures or scalability. Usually claims concern more abstract properties, such as generality, expressiveness, and robustness. Abstract metrics are often properties that involve integration of multiple properties across multiple trials and even across multiple tasks and domains. One challenge is to determine how to ground these abstract metrics in measurable properties of HLI systems' behavior.

- **Task and Domain Generality:** How well does a system (or architecture) support behavior across a wide range of tasks and domains? Concerns about task and domain generality are one of the primary factors that distinguish research in HLI from much of the other research in AI. This requires measures of diversity of tasks and domains, which are currently lacking. Given the primacy of generality, it is not surprising that many other abstract metrics address aspects of behavior and system construction that are related to generality.
- **Expressivity:** What kinds or range of knowledge can an HLI system accept and use to influence behavior? This relates to generality because restrictions on expressiveness can, in turn, restrict whether a system can successfully pursue a task in a domain. For example, systems that only support propositional representations will have difficulty reasoning about problems that are inherently relational.
- **Robustness:** How does speed or quality of solutions change as a task is perturbed or some knowledge is removed or added? One can also measure robustness of an architecture – how behavior changes as an aspect of the architecture is degraded – but this is rarely considered an important feature of HLI systems. Instead, the interest lies in how well the system can respond to partial or incomplete knowledge, incorrect knowledge, and changes in a task that require some mapping of existing knowledge to a novel situation.
- **Instructability:** How well can a system accept knowledge from another agent? Instructability emphasizes acquiring new skills and knowledge, as well as acquiring new tasks. Finer-grain measures of instructability include the language needed for instruction, the breadth of behavior that can be taught, and the types of interactions supported, such as whether the instructor is in control, whether the agent is in control, or whether dynamic passing of control occurs during instruction.
- **Taskability:** To what extent can a system accept and/or generate, understand, and start on a new task? Taskability is related to instructability, but focuses working on new tasks. Humans are inherently taskable and retaskable, being able to attempt new tasks without requiring an external programmer that understands its internal representations. Humans also generate new tasks on their own. In contrast, most current systems only pursue the tasks and subtasks with which they were originally programmed and cannot dynamically extend the tasks they pursue.
- **Explainability:** Can the system explain what it has learned or experienced, or why it is carrying out some behavior? Humans do not have “complete” explainability – the ability to provide justifications for all decisions leading up to external behavior – so this capability is a matter of degree.

Conclusion

It is obvious that developing human-level intelligence is a huge challenge. However, important parts of that scientific and engineering enterprise are the methods and practices for evaluating the systems as they are developed. In this paper, we present some of the primary challenges that arise in evaluation that distinguish it from research on more specialized aspects of artificial intelligence. We also attempt to characterize the types of scientific claims that arise in research on HLI, distinguishing different classes of claims that can be made at the system level, and then further analyzing the independent and dependent variables of those claims.

Having clear, explicit claims has always been a critical part of scientific progress, and we encourage researchers to be more explicit in the claims of the theories and systems they develop. This not only helps ourselves in designing appropriate experiments, it also makes it much easier for other researchers to evaluate the contribution of a piece of work. In addition to being specific about claims, the field also needs shared notions of methodologies and metrics associated with evaluating those claims. The abstract metrics enumerated here suggest some ways in which HLI researchers can begin to better distinguish these systems from more traditional AI systems. However, much work remains to identify methods for measuring and evaluating these capabilities.

The next step for this effort is to explore tasks and environments that can be shared across the community. Given the broad goals of HLI research, we need multiple testbeds that support environments in which many tasks can be pursued, and which include tools for performing experimentation and evaluation. Working on common problems will simplify cross-system evaluation and collaboration, both important steps toward developing human-level intelligence.

Acknowledgments

This paper is based on discussions at a workshop on evaluation of human-level intelligence, held at the University of Michigan in October, 2008. Participants included Joscha Bach, Paul Bello, Nick Cassimatis, Ken Forbus, Ben Goertzel, John Laird, Christian Lebiere, Pat Langley, Robert Mariner, Stacey Marsella, Matthias Scheultz, Satinder Singh, and Robert Wray. The workshop was funded under grant N00014-08-1-1214 from ONR.

References

- Anderson, J. R. & Lebiere, C. L. 2003. The Newell test for a theory of cognition. *Behavioral & Brain Science* 26, 587-637.
- Cohen, P. R., 1995. *Empirical methods for artificial intelligence*, Cambridge, MA: MIT Press.
- Cohen, P.R. 2005. If not Turing’s test, then what? *AI Magazine*, Winter, 26; 61-68.

- Doorenbos, R. B. 1994. Combining left and right unlinking for matching a large number of learned rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.
- Gluck, K. A. & Pew, R.W. 2005. Modeling human behavior with integrated cognitive architectures: comparison, evaluation, and validation LEA/Routledge.
- Jones, R. M., & Wray, R. E. 2006. Comparative analysis of frameworks for knowledge-intensive intelligent agents. *AI Magazine*, 27, 57-70.
- Laird, J. E., 2008. Extending the Soar cognitive architecture. In *Proceedings of the First Conference on Artificial General Intelligence*.
- Langley, P., & Choi, D., 2006. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston: AAAI Press.
- Langley, P., Laird, J. E., & Rogers, S., in press. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*.
- Langley, P., & Messina, E. 2004. Experimental studies of integrated cognitive systems. Proceedings of the Performance Metrics for Intelligent Systems Workshop. Gaithersburg, MD.
- Legg, S. & Hutter, M. 2007. Universal Intelligence: A Definition of Machine Intelligence, *Minds and Machines*, 17:4, 391-444.
- Newell, A., Simon, H. A., 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, **19**
- Turing, A., 1950. Computing Machinery and Intelligence. *Mind*, 59; 433-460.
- Wray, R. E., & Laird, J. E. 2003. An architectural approach to consistency in hierarchical execution. *Journal of Artificial Intelligence Research*. 19; 355-398.
- Wray, R. E., & Lebiere, C. 2007. Metrics for Cognitive Architecture Evaluation. In Proceedings of the AAAI-07 Workshop on Evaluating Architectures for Intelligence, Vancouver, B. C.

Extending Cognitive Architectures with Mental Imagery

Scott D. Lathrop

United States Military Academy
D/EECS
West Point, NY 10996
scott.lathrop@usma.edu

John E. Laird

University of Michigan
2260 Hayward
Ann Arbor, MI 48109-2121
laird@umich.edu

Abstract

Inspired by mental imagery, we present results of extending a symbolic cognitive architecture (Soar) with general computational mechanisms to support reasoning with symbolic, quantitative spatial, and visual depictive representations. Our primary goal is to achieve new capabilities by combining and manipulating these representations using specialized processing units specific to a modality but independent of task knowledge. This paper describes the architecture supporting behavior in an environment where perceptual-based thought is inherent to problem solving. Our results show that imagery provides the agent with additional functional capabilities improving its ability to solve rich spatial and visual problems.

Introduction

The generality and compositional power of sentential, symbolic processing has made it central to reasoning in general AI systems. However, these general symbolic systems have failed to address and account for inherently perceptual, modality-specific processing that some argue should participate directly in thinking rather than serve exclusively as a source of information (Barsalou 1999; Chandrasekaran 2006). Mental imagery is an example of such thought processing.

In this paper, we argue that general, intelligent systems require mechanisms to compose and manipulate amodal, symbolic and modality-specific representations. We defend our argument by presenting a synthesis of cognition and mental imagery constrained by a cognitive architecture, Soar (Laird 2008). Empirical results strengthen our claim by demonstrating how specialized, architectural components processing these representations can provide an agent with additional reasoning capability in spatial and visual tasks.

Related Work

One of the key findings in mental imagery experiments is that humans imagine objects using multiple representations and mechanisms associated with perception (Kosslyn, et al., 2006). For spatial and visual imagery, we assume there are at least three distinct representations: (1) *amodal symbolic*, (2) *quantitative spatial*, and (3) *visual depictive*. General reasoning with each of these representations is a key

distinction between this work and others. The history of using these representations in AI systems begins perhaps with Gelernter's (1959) geometry theorem prover and Funt's (1976) WHISPER system that reasoned with quantitative and depictive representations respectively. Some researchers, to include Glasgow and Papadias (1992) and Barkowsky (2002), incorporated mental imagery constraints in the design of their specific applications.

The CaMeRa model of Tabachneck-Schijf's et al. (1997) is perhaps the closest system related to this work. CaMeRa uses symbolic, quantitative, and depictive representations and includes visual short-term and long-term memories. Whereas their shape representation is limited to algebraic shapes (i.e. points and lines), we leave the type of object open-ended. CaMeRa's spatial memory is limited to an object's location while ignoring orientation, size, and hierarchical composition (e.g. a car is composed of a frame, four wheels, etc.). Our system uses these spatial properties, providing significantly more reasoning capability.

Cognitive architectures have traditionally ignored modality specific representations. ACT-R's (Anderson, 2007) perceptual and motor systems focus on timing predictions and resource constraints rather than their reuse for reasoning. Some researchers have extended the perception and motor capabilities of cognitive architectures (e.g. see Best et al., 2002; Wray et al., 2005). Each contribution effectively pushes the system closer to the environment but requires ad-hoc, bolted-on components tailored for specific domains. These approaches assume that cognition abandons perceptual mechanisms after input rather than using these mechanisms for problem solving.

Kurup and Chandrasekaran (2007) argue for general, multi-modal architectures and augment Soar with diagrammatic reasoning. They are non-committal as to whether diagrams are quantitative or depictive. Their current implementation uses strictly quantitative structures. Key differences include their proposal for a single, working memory containing both symbolic and diagrammatic representations. We propose separate symbolic and representation-specific short-term memories where perceptual representations are not directly accessible to the symbolic system. Whereas their diagrammatic system constrains the specific types of objects to a point, curve, or

region, we leave the type of object open-ended to any shape the agent experiences in the world or imagines by composing known objects.

Wintermute and Laird (2008) extend Soar with a spatial reasoning system that focuses on translating qualitative predicates into quantitative representations and simulating continuous motion—extending the framework described here as it relates to spatial imagery. Gunzelmann and Lyon (2007) propose extending ACT-R with specialized, spatial processing that includes quantitative information. They do not plan to incorporate depictive representations without compelling evidence for their use. We hope to provide some evidence and argue that all three representations are necessary to achieve general functionality.

Experimental Environment

In previous work, Lathrop and Laird (2007) demonstrated how extending Soar with quantitative spatial and visual depictive representations provided an agent with capabilities for recognizing implicit spatial and visual properties. However, the results were limited to solving internally represented problems. This paper extends those results to a dynamic environment where the agent must interpret and act on information from multiple internal and external sources.

The U.S. Army’s work in developing robotic scouts for reconnaissance missions (Jaczkowski, 2002) motivates the evaluation environment. In support of this effort, we built a simulation modeling a section of two robotic scout vehicles that must cooperate to maintain visual observation with an approaching enemy (Figure 1a). One scout, the section lead, is a Soar agent, modeled with and without mental imagery for evaluation purposes. The other, teammate, scout is scripted. The section’s primary goal is to keep their commander informed of the enemy’s movement by periodically sending observation reports (through the lead) of the enemy’s location and orientation. The agent cannot observe its teammate because of terrain occlusions. However, the teammate periodically sends messages regarding its position. The teammate continuously scans the area to its front (Figure 1b) and sends reports to the agent when it observes the enemy. The teammate can reorient its view in response to orders from the agent. The agent can look at the environment (Figure 1c) or its map (Figure 1d).

To motivate the reasoning capabilities when using multiple representations, consider how the agent makes decisions in this domain. Typically, a scout follows these steps after initial visual contact: (1) Deploy and report, (2) analyze the situation, and (3) choose and execute a course of action (U.S. Army 2002). Analysis involves reasoning about friendly and enemy locations and orientations, terrain, and obstacles. If the scout leader does not know the locations of all expected enemy, then he might hypothesize where other enemy entities are (Figure 1d). Note that the hypothesized enemy in Figure 1d is not the same as the actual situation in Figure 1a but rather an estimate based on the agent’s knowledge of how the enemy typically fights.

Analysis involves visualizing the situation and mentally simulating alternatives. Using spatial imagery, an agent can imagine each observed entity’s map icon on its external map. If the agent is confident in the information, it can “write” it on the external map, in effect making it persist. As information changes the agent updates the map, keeping its perceived image of the situation up to date. Using the external map as perceptual background, the agent can then imagine key terrain (enemy goals), possible enemy paths, its viewpoint, and its teammate’s viewpoint. Using visual imagery to take advantage of explicit space representation in a depiction, the agent can imagine what portion of those viewpoints cover the possible enemy paths and then imagine alternative courses of action by simulating different viewpoints. Based on the analysis the agent decides if it should reorient itself, its teammate, or both.

In summary, decision-making proceeds by combining perceptual representations with task specific knowledge to construct an imagined scene. Analysis emerges through the manipulation of symbolic, quantitative, and depictive representations. Retrieval of the resulting representations provides new information to the agent that it uses to reason and produce action in the environment.

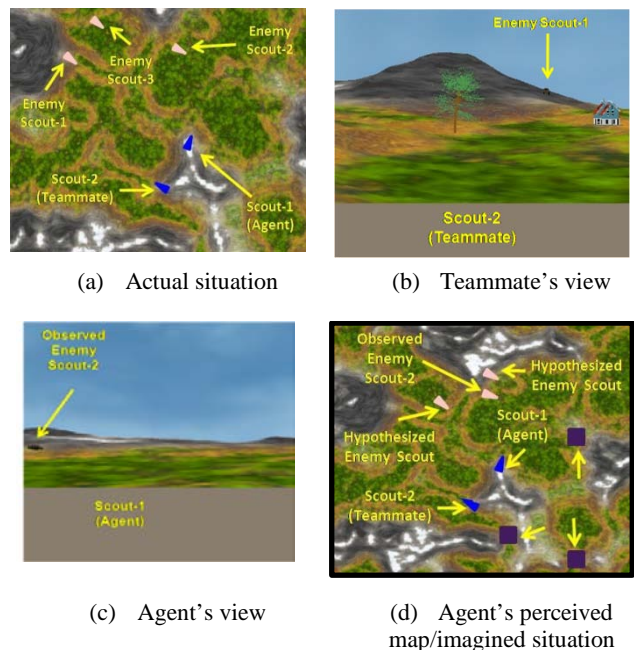


Figure 1: Experimental Environment

Architecture

Soar and its Spatial-Visual Imagery (Soar+SVI) module are the two major components in our system (Figure 2). Soar encompasses the symbolic representation. SVI includes the quantitative and depictive representations. It encapsulates high-level visual perception and mental imagery processing.

Soar’s symbolic memories include a *declarative*, short-term memory (STM) and a *procedural*, long-term memory (LTM). The symbolic STM is a graph structure (Figure 2)

representing the agent’s current state. Some symbols may represent an object (filled gray circle in Figure 2). These “visual-object” symbols emerge from the current perception or activation of a previously stored memory. They may be associated with non-visual symbols that augment the object with additional information (e.g., the object is an enemy scout). The visual-object symbol may have properties defining its explicit visual features and qualitative spatial relationships with other objects. Procedural LTM is a set of productions, some of which propose *operators* that a decision procedure selects for application. The application of an operator makes persistent changes to short-term memory and may send commands to a motor system, or, in SVI’s case, imagery processes. Processing occurs by iteratively proposing, selecting, and applying operators.

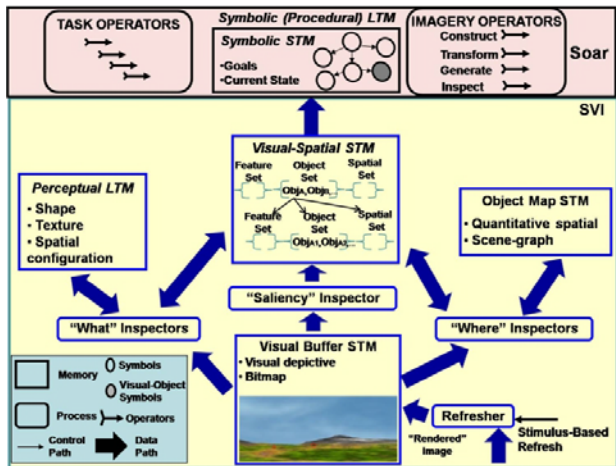


Figure 2: Architectural overview with visual perceptual processing

Within SVI, the *Visual Buffer* (bottom of Figure 2) is a depictive memory activated from bottom-up visual-perception or top-down imagery processing. In contrast to sentential symbols, space is inherent in the representation and the encoding is strictly visual information. The depiction as a whole represents shape, size, orientation, location, and texture from a specific perspective. Computationally, it is a set of 2D bitmaps with at least one bitmap representing either the egocentrically perceived scene or an imagined scene from a specific viewpoint. The system creates additional bitmaps to support the processing.

The *Object Map* (right side of Figure 2) maintains the quantitative spatial representation of objects in the currently perceived or imagined scene by fixing an object’s location, orientation, and size in space. The Object Map uses a scene-graph data structure (Figure 3). The root node represents the perceived or imagined scene and children nodes are salient, visual objects. Figure 3 shows the number of visual objects to be N where N is hypothesized to be four to five based on working-memory capacity (Jonides et al., 2008). Intermediate nodes represent an object’s composition and contain translation, scaling, and rotation metrics to capture spatial relationships between objects. Leaf nodes represent

shape (i.e. a three-dimensional mesh of vertices and indices) and texture to support rendering a bitmap. The structure is a graph because multiple leaf nodes may share shape and texture (e.g. a shared wheel). A viewpoint facilitates the generation of a depiction from a particular perspective.

Sentential, geometric algorithms are the basis for the computational processing that infers knowledge from this representation. The structure is sufficient for spatial reasoning between convex objects and simulating dynamical systems (Wintermute 2008). However, if reasoning requires specific shape or visual properties, a depictive representation is more appropriate.

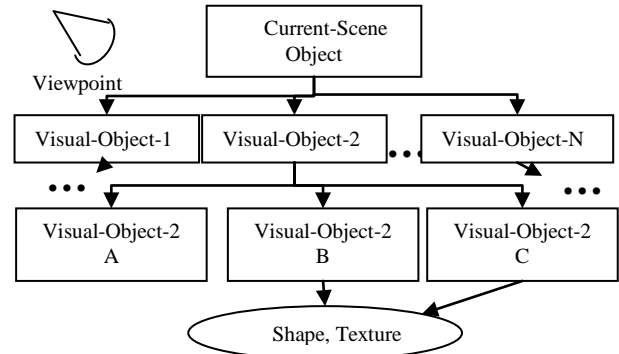


Figure 3: Object Map’s scene-graph and viewpoint data structures

The remaining two memories in SVI are not associated with a particular representation but support reasoning indirectly. The *Visual-Spatial STM* (middle of Figure 2) is a shared memory between Soar and SVI. It is hierarchical with the root representing sets of extracted salient objects, spatial relationships, and visual features applying to the current scene. Each salient object may have subsequent levels in the hierarchy with its own feature, object, and spatial sets. *Perceptual long-term memory* (PLTM) is a container of prototypical objects where each object is a scene graph. A scene-graph in PLTM is distinct from the Object Map as the graph is not an instance in the current scene but rather a memory of an object’s shape, texture, and spatial configuration without a fixed frame of reference.

Visual Perception

Our modeling of visual perception, to include the separation between “what” and “where” pathways is theoretical. We include it since psychological evidence indicates that mental imagery and vision share similar mechanisms thereby constraining the architectural design. Our ultimate goal is to incorporate realistic perception in the architecture.

A *Refresher* process activates the Visual Buffer from sensory stimulus (bottom right of Figure 2). Upon activation, a “*Saliency*” *Inspector* marks relevant objects in the current scene and creates a symbolic structure for each salient object in VS-STM. Two parallel processes then initiate a more detailed inspection of the Visual Buffer, focusing on the marked objects. The “*What*” *inspectors*

extract features in support of recognition by matching features with shape and color in PLTM. Simultaneously, the “Where” inspectors extract the location, orientation, and size of the objects from the Visual Buffer and build the quantitative spatial representation in the Object Map. Both inspectors update the structures in VS-STM and symbolic results are sent to Soar where operators associate the input with existing knowledge (e.g. the object is an enemy).

Spatial Imagery

An agent uses spatial imagery by invoking an imagery operator (top right of Figure 2). To *construct* a spatial image, the agent can compose two visual-objects from PLTM or add a visual-object from PLTM to the scene. Specialized processing units within SVI respond to the specific imagery command (Figure 4). The *Constructor* receives the operator’s symbolic information and builds the quantitative representation in the Object Map by combining each object’s general shape from PLTM with qualitative spatial knowledge from Soar. In the scout domain, the agent continuously analyzes the situation by imagining the friendly, enemy, and obstacle locations and orientations.

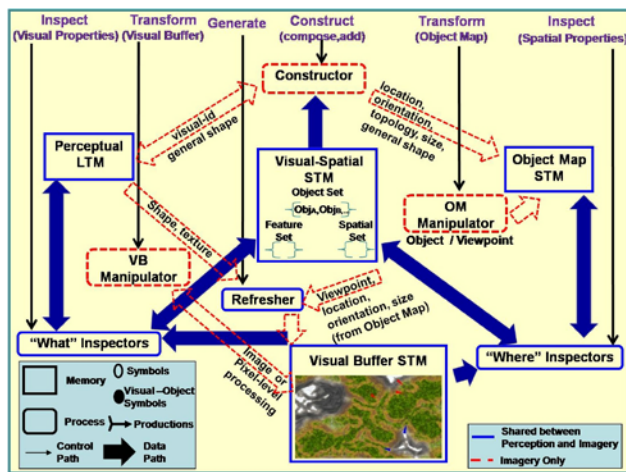


Figure 4: Mental imagery processes

The *transform* operator manipulates the Object Map’s quantitative representation through its *Manipulator* (Figure 4). The manipulation may change the viewpoint or transform (i.e. translation, rotation, scaling) a specific object. In the scout domain, the agent dynamically updates specific objects when observing (either visually or via a teammate’s report) changes to spatial relationships. The agent may also imagine different situations, effectively simulating hypothesized scenarios, and infer the changed spatial relationships. For example, the agent modifies the orientation of imagined views to determine if its team can improve coverage of enemy routes. When the agent or its teammate loses visual contact with the enemy, the agent can simulate movement with knowledge of a vehicle’s velocity. From SVI’s perspective, the objects it is manipulating are general—task knowledge remains encoded in Soar.

Visual Imagery

If a depictive representation is required (e.g. to determine if the scout section has adequate visual coverage), the *generate* operator (Figure 4) initiates processing. The *Refresher* interprets the command and combines each object’s specific shape and texture from PLTM with the Object Map’s quantitative information to generate the bitmap in the Visual Buffer. Generation may render some or all of the visual objects in the Object Map and create one or more bitmaps to support visual reasoning.

The *VBManipulator* transforms the images in the VisualBuffer using either standard image processing (e.g. edge detectors) or algorithms that take advantage of the topological structure and color using pixel-level rewrites (Furnas et al., 2000). Unlike sentential processing (e.g. Gaussian filters), pixel-level rewrites take advantage of the explicit topological structure and color of a bitmap. Similar to a production system, there are a set of rules with a left-hand side (LHS) and a right-hand side (RHS). Rather than predicate symbols, however, the LHS conditions and RHS actions are depictive representations that operate on the shared image. The color and shape of each LHS depiction, determines a match rather than the sentential structure.

Figure 5 illustrates an example of two depictive rules. The top rule is a 1x2 rule stating, “If there is a black pixel adjacent to a gray pixel then change the gray pixel to a white pixel.” Similarly, the bottom rule is a 2x2 rule that states, “If there is a black pixel diagonally adjacent to a gray pixel then change the gray pixel to a white pixel.” The asterisks represent wildcard values and a rule may specify alternative rotations (90, 180, 270 degrees) for matching. Each rule can have arbitrary shape and color and a set of these rules can represent a high-level task in Soar (e.g. find-enemy-path). Priorities enforce sequencing, and the processing iterates over the image while there are matches.

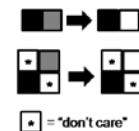
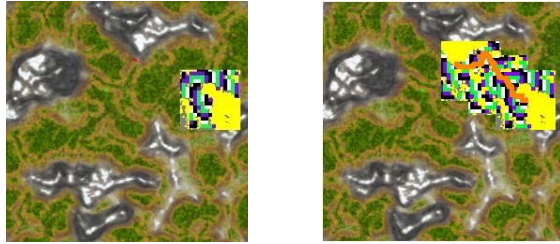


Figure 5: Example pixel rewrite rules

A way for the agent to analyze its team’s position is to imagine a hypothesized path from each enemy’s location to key terrain (Figure 6). The analysis should take into account the agent’s knowledge about the surrounding terrain and known obstacles. An algorithmic solution translated into a set of pixel rewrites is the following:

- (1) Mark all known obstacles and impassable terrain (known threshold values) with a color (yellow). Mark all other pixels gray.
- (2) Grow an iso-distance contour field of four colors avoiding any previously marked, barriers (Figure 6a).
- (3) Walk the contour field from source to sink, marking the path along the way (Figure 6b).

After the imagined path(s) are marked, the agent can generate each scout’s view to determine if there is adequate coverage (Figure 7).



(a) Distance field flood (b) Path finding
Figure 6: Demonstration of pixel-level rewrites



Figure 7: Agent imagining coverage of an imagined enemy path

After constructing and manipulating the representations, the agent can infer spatial and visual properties. The *inspect* operator (Figure 4) provides the symbolic query. For example, “what is the direction and distance between enemy scout-1 and the key terrain in the east” or “how much of the teammate’s view covers enemy-1’s hypothesized path (Figure 7)?” The appropriate “What” or “Where” process interprets the query and returns the symbolic results to Soar as described for visual perception.

The reasoning uses abstract mechanisms rather than problem specific annotations. For example, “how much of the teammate’s view covers enemy-1’s hypothesized path?” proceeds as follows:

- (1) What is the topology between object-1 (the teammate’s view) and object-2 (the hypothesized path)? The inspector provides a symbolic “overlaps” result and stores a shape feature (shape-1) representing the overlap in VS-STM (Figure 4).
- (2) What is the scalar size (i.e. length) of shape-1? SVI calculates and returns the size of shape-1.

Functional Evaluation

Extending a symbolic architecture with mental imagery mechanisms provides an agent with functional capability that the system cannot achieve without it. To evaluate this claim, we created three agents modeling the lead scout. The first agent (Soar+SVI) observes, analyzes, and decides on a course of action by using symbolic, quantitative spatial, and visual depictive representations. The second agent (Soar-SVI) uses the same task knowledge as the first agent but reasons using strictly symbolic representations and

processing in Soar. As a baseline, a third agent (Observer) and its teammate simply observe to their front and send reports without any attempt at re-positioning.

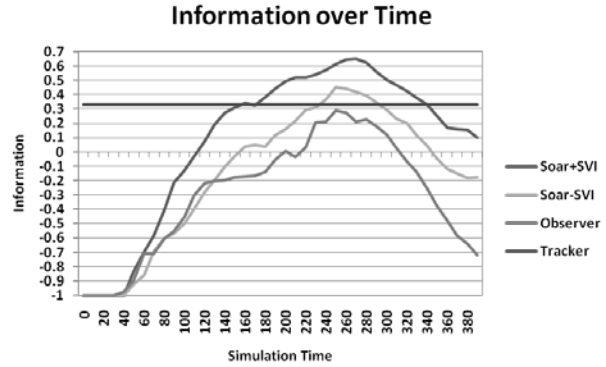


Figure 8: Measure of information over time

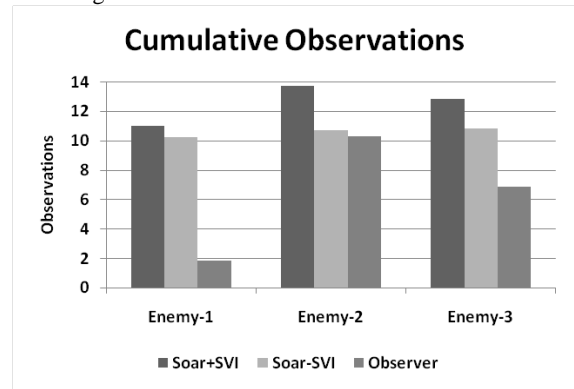


Figure 9: Number of reported observations

There are two evaluation metrics. The first is the amount of information the commander receives on the enemy’s location over time (Figure 8). The second metric is the number of reported enemy observations (Figure 9). Each reflects an average of 30 trials. In Figure 8, the x-axis is the current time and the y-axis measures the amount of information per unit time with 1.0 signaling perfect information and -1.0 indicating no information. The measure of information is an average of all three enemy entities at simulation time, t , calculated as follows:

$$I_t = \begin{cases} -1 & \text{if no observation} \\ 1 - \delta & \text{otherwise} \end{cases} \quad \text{where:}$$

$$\delta = \sqrt{(obs_x - act_x)^2 + (obs_y - act_y)^2} / d_{acceptable}$$

(obs_x, obs_y) is the reported location of an entity at time, t and (act_x, act_y) is the actual location of an entity at time, t

$d_{acceptable} = \text{the acceptable square distance}$

$$= \sqrt{d_x^2 + d_y^2} \text{ where } d_x = d_y = 500 \text{ meters}$$

The agent receives a positive score for a given enemy if at simulation time, t , a reported enemy’s location is within a 500 x 500 meter square area of the enemy’s actual location at that time. Otherwise, the information score is negative for

with a minimum score of -1.0. The “Tracker” in Figure 8 illustrates the amount of information a scout team provides if each scout observes one enemy at the beginning of the simulation and then “tracks” that entity to the simulation’s conclusion. Assuming no terrain occlusions, instantaneous message passing, and the third enemy not in vicinity of the tracked entities, the “Tracker” would receive an information score of $(1.0 + 1.0 - 1.0) / 3 = 0.33$ for each time unit.

The results demonstrate that the Soar+SVI agent provides more information upon initial contact (the slope of its line in Figure 8 is steeper) and for a longer, sustained period. The reason is that the Soar+SVI agent is able to reposition its team more effectively as its analysis is more accurate. The Soar-SVI agent often under or overestimates adjustments resulting in the team missing critical observations.

On average, the Soar+SVI agent sends more observation reports to the commander (Figure 9) indicating that the team has detected the enemy more frequently. The number of observation reports also shows that the agent is able to perform other cognitive functions (observe, send and receive reports) indicating that imagery is working in conjunction with the entire cognitive system.

Conclusion

In this paper, we demonstrate that augmenting a cognitive architecture with mental imagery mechanisms provides an agent with additional, task-independent capability. By combining symbolic, quantitative, and depictive representations, an agent improves its ability to reason in spatially and visually demanding environments. Our future work includes expanding individual architectural components—specifically by pushing the architecture closer to sensory input. To investigate this in more depth, we are exploring robotics to determine how cognitive architectures augmented with mental imagery can provide a robot with higher-level reasoning capabilities. Paramount in this exploration is an understanding of how our perceptual theory incorporates typical robotic sensors (e.g. laser, stereoscopic video, global positioning system, etc.) and how imagery may prime robotic effectors (motor imagery).

References

- Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* New York, NY: Oxford University Press.
- Barkowsky, T. (2002). Mental representation and processing of geographic knowledge - A computational approach. Berlin: Springer-Verlag.
- Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, 22, 577-660.
- Best, B.J., Lebiere, C., and Scarpinato, C.K. (2002). A Model of Synthetic Opponents in MOUT Training Simulations using the ACT-R cognitive architecture. In *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavior Representation*. Orlando, FL.
- Chandrasekaran, B. (2006). Multimodal Cognitive Architecture: Making Perception More Central to Intelligent Behavior. AAAI National Conference on Artificial Intelligence, Boston, MA.
- Funt, B.V. (1976). “WHISPER: A computer implementation using analogues in reasoning,” PhD Thesis, The University of British Columbia, Vancouver, BC Canada.
- Furnas, G., Qu, Y., Shrivastava, S., and Peters, G. (2000). The Use of Intermediate Graphical Constructions in Problem Solving with Dynamic, Pixel-Level Diagrams. In *Proceedings of the First International Conference on the Theory and Application of Diagrams: Diagrams 2000*, Edinburgh, Scotland, U.K.
- Glasgow, J., and Papadias, D. (1992). Computational imagery. *Cognitive Science*, 16, 355-394.
- Gelernter, H. (1959). Realization of a geometry theorem-proving machine. Paper presented at the International Conference on Information Processing, Unesco, Paris.
- Gunzelmann, G., and Lyon, D. R. (2007). Mechanisms of human spatial competence. In T. Barkowsky, M. Knauff, G. Ligozat, & D. Montello (Eds.), *Spatial Cognition V: Reasoning, Action, Interaction. Lecture Notes in Artificial Intelligence #4387* (pp. 288-307). Berlin, Germany: Springer-Verlag.
- Jaczkowski, J. J. (2002). Robotic technology integration for army ground vehicles. *Aerospace and Electronic Systems Magazine*, 17, 20-25.
- Jonides, J., Lewis, R.L., Nee, D.E., Lustig, C.A., Berman, M.G., and Moore, K.S. (2008). The Mind and Brain of Short-Term Memory. *Annual Review of Psychology*, 59, 193-224.
- Kosslyn, S. M., Thompson, W. L., and Ganis, G. (2006). *The Case for Mental Imagery*. New York, New York: Oxford University Press.
- Kurup, U., and Chandrasekaran, B. (2007). Modeling Memories of Large-scale Space Using a Bimodal Cognitive Architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, Ann Arbor, MI.
- Laird, J.E. (2008). Extending the Soar Cognitive Architecture, Artificial General Intelligence Conference, 2008.
- Lathrop, S. D., and Laird, J. E. (2007). Towards Incorporating Visual Imagery into a Cognitive Architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, Ann Arbor, MI.
- Tabachneck-Schijf, H.J.M., Leonardo, A.M., and Simon, H.A. (1997). CaMeRa: A Computational Model of Multiple Representations. *Cognitive Science*, 21(3), 305-350.
- U.S. Army. (2002). Field Manual 3-20.98, Reconnaissance Platoon. Department of the Army, Washington D.C.
- Wintermute, S. and Laird, J. E. (2008). Bimodal Spatial Reasoning with Continuous Motion. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, Chicago, Illinois
- Wray, R. E., Laird, J.E., Nuxoll, A., Stokes, D., and Kerfoot, A. (2005). Synthetic Adversaries for Urban Combat Training. *AI Magazine*, 26(3), 82-92.

A Comparative Approach to Understanding General Intelligence: Predicting Cognitive Performance in an Open-ended Dynamic Task

Christian Lebiere¹, Cleotilde Gonzalez² and Walter Warwick³

¹ Psychology Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213

² Department of Social and Decision Sciences, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213

³ Alion Science and Technology, 4949 Pearl East Circle, Boulder, CO 80401

cl@cmu.edu, coty@cmu.edu, wwarwick@alionscience.com

Abstract

The evaluation of an AGI system can take many forms. There is a long tradition in Artificial Intelligence (AI) of competitions focused on key challenges. A similar, but less celebrated trend has emerged in computational cognitive modeling, that of model comparison. As with AI competitions, model comparisons invite the development of different computational cognitive models on a well-defined task. However, unlike AI where the goal is to provide the maximum level of functionality up to and exceeding human capabilities, the goal of model comparisons is to simulate human performance. Usually, goodness-of-fit measures are calculated for the various models. Also unlike AI competitions where the best performer is declared the winner, model comparisons center on understanding in some detail how the different modeling “architectures” have been applied to the common task. In this paper we announce a new model comparison effort that will illuminate the general features of cognitive architectures as they are applied to control problems in dynamic environments. We begin by briefly describing the task to be modeled, our motivation for selecting that task and what we expect the comparison to reveal. Next, we describe the programmatic details of the comparison, including a quick survey of the requirements for accessing, downloading and connecting different models to the simulated task environment. We conclude with remarks on the general value in this and other model comparisons for advancing the science of AGI development.

Introduction

The evaluation of an AGI system can take many forms. Starting with Turing (e.g., Turing, 1950), the idea that artificial intelligence might be “tested” has led quite naturally to a tradition of competition in AI in which various systems are pitted against each other in the performance of a well-specified task. Among the most famous include the Friedkin prize for a machine chess player that could beat the human chess champion (Hsu, 2002), the robocup soccer competition for autonomous robots (Asada et al., 1999) and the DARPA Grand Challenge race across the desert (Thrun et al., 2006). A similar, but less celebrated trend has emerged in computational cognitive modeling, that of model comparison. As with AI competitions, model comparisons

invite the development of different computational cognitive models on a well-defined task. However, unlike AI where the goal is to provide the maximum level of functionality up to and exceeding human capabilities, the goal of model comparisons is to most closely simulate human performance. Thus, usually, goodness-of-fit measures are calculated for the various models. Also, unlike AI competitions where the best performer is declared the winner, model comparisons center on understanding in some detail how the different modeling “architectures” have been applied to the common task. In this regard model comparisons seek to illuminate general features of computational approaches to cognition rather than identify a single system that meets a standard of excellence on a narrowly defined task (Newell, 1990).

In this paper we announce a new model comparison effort that will illuminate the general features of cognitive architectures as they are applied to control problems in dynamic environments. We begin by briefly describing the general requirements of a model comparison. Next, we describe the task to be modeled, our motivation for selecting that task and what we expect the comparison to reveal. We then describe the programmatic details of the comparison, including a quick survey of the requirements for accessing, downloading and connecting different models to the simulated task environment. We conclude with remarks on the general value we see in this and other model comparison for advancing the science of AGI development. Although everyone loves a winner, the real value of a model comparison is found in its methodological orientation. Indeed, given the inherent flexibility of computational abstractions, understanding the workings of a particular cognitive system, much less judging its usefulness or “correctness,” is not easily done in isolation.

General Requirements of a Model Comparison

We have gained direct experience from a number of modeling comparisons projects, including the AFOSR AMBR modeling comparison (Gluck & Pew, 2005) and

the NASA Human Error Modeling comparison (Foyle & Hooley, 2008). We have also entered cognitive models into multi-agent competitions (Billings, 2000; Erev et al, submitted) and organized symposia featuring competition between cognitive models as well as mixed human-model competitions (Lebiere & Bothell, 2004; Warwick, Allender, Strater and Yen, 2008). From these endeavors, we have gained an understanding of the required (and undesirable) characteristics of a task for such projects.

While previous modeling comparison projects did illustrate the capabilities of some modeling frameworks, we found that the tasks were often ill-suited for modeling comparison for a number of reasons:

- The task demands a considerable effort just to model the details of task domain itself (and sometimes, more practically, to connect the model to the task simulation itself). This often results in a model whose match to the data primarily reflects the structure and idiosyncrasies of the task domain itself rather than the underlying cognitive mechanisms. While this task analysis and knowledge engineering process is not without merit, it does not serve the primary purpose of a model comparison effort, which is to shed light upon the merits of the respective modeling frameworks rather than the cleverness and diligence of their users.
- The task is defined too narrowly, especially with regard to the data available for model fitting. If the task does not require model functionality well beyond the conditions for which human data is available, then the comparison effort can be gamed by simply expanding effort to parameterize and optimize the model to the data available. This kind of task puts frameworks that emphasize constrained, principled functionality at a disadvantage over those that permit arbitrary customization and again serves poorly the goals of a modeling comparison.
- The task is too specialized, emphasizing a single aspect, characteristic or mechanism of cognition. While this type of task might be quite suitable for traditional experimentation, it does not quite the kind of broad, general and integrated cognitive capabilities required of a general intelligence framework.
- No common simulation or evaluation framework is provided. While this allows each team to focus on the aspects of the task that are most amenable to their framework, it also makes direct comparison between models and results all but impossible.
- No suitably comparable human data is available. While a purely functional evaluation of the

models is still possible, this biases the effort toward a pure competition, which emphasizes raw functionality at the expense of cognitive fidelity.

This experience has taught us that the ideal task for a model comparison is:

- lightweight, to limit the overhead of integration and the task analysis and knowledge engineering requirements
- fast, to allow the efficient collection of large numbers of Monte Carlo runs
- open-ended, to discourage over-parameterization and over-engineering of the model and test its generalization over a broad range of situations
- dynamic, to explore emergent behavior that is not predictable from the task specification
- simple, to engage basic cognitive mechanisms in a direct and fundamental way
- tractable, to encourage a direct connect between model and behavioral data

Like other enduring competitive benchmarks of human cognition that have kept on driving the state of the art in some fields (e.g. Robocup), the key is to find the right combination of simplicity and emergent complexity. We believe the task we have selected, described in detail below, meets these requirements and strikes the right combination between simplicity and complexity. In fact, in our own pilot studies, we have encountered significant challenges in developing models of the task that could account for even the basic results of the data and our models have consistently surprised us by their emergent behavior, and even minor changes in task representation have had deep consequences for model behavior (Lebiere, Gonzalez, & Warwick, under review). We expect the same will be true for other participants in this effort.

The Dynamic Stocks and Flows Task

In dynamic systems, complexity has often been equated with the number of elements to process at a given time: goals, alternatives, effects and processes (Brehmer & Allard, 1991; Dörner, 1987). Researchers have investigated the problems and errors that people make while dealing with this type of complexity in dynamic systems to foster our understanding of decision making. However, dynamic systems manifest another type of complexity that is less well known, that is *dynamic complexity* (Diehl & Serman, 1995). This type of complexity does not depend on the number of elements to process in a task. In fact, the underlying task could be superficially simple depending on a single goal and one element to manage and make decisions. Dynamic complexity follows from the combinatorial relationships that arise from the interactions of even a few variables over time.

Gonzalez & Dutt (2007) and Dutt & Gonzalez (2007) have investigated human performance in dynamically complex environments using a simple simulation called the dynamic stocks and flows (DSF). DSF (see figure 1) is an interactive learning tool that represents a simple dynamic system consisting of a single *stock* in which the rate of accumulation is a function of time; *inflows* increase the level of stock, and *outflows* decrease the level of stock. The goal in DSF is to maintain the stock within an acceptable range over time. The stock is influenced by external flows (External Inflow and Outflow) that are out of the user's control, and by user flows (User Inflow and Outflow) that the player of DSF decides on in every time period.

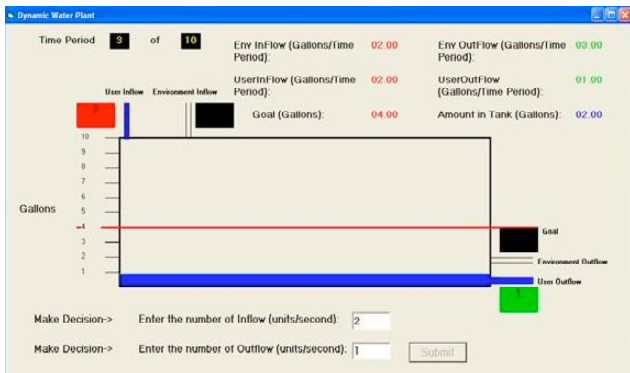


Figure 1: DSF Interface

A stock, inflows and outflows are the basic elements of every dynamic task, at the individual, organizational, and global levels (for a discussion of the generality of this structure in complex dynamic systems, see Cronin, Gonzalez, & Sterman, 2008). For example, the structure of the task discussed here, is currently being used to investigate the problems of control of the atmospheric CO₂, believed to lead to Global Warming (Dutt & Gonzalez, 2008).

Despite its seeming simplicity, controlling the DSF is very difficult for most subjects (Gonzalez & Dutt, 2007; Dutt & Gonzalez, 2007). For example, Cronin, Gonzalez & Sterman (2008) found that in a sample of highly educated graduate students with extensive technical training nearly half were unable to predict the qualitative path of a stock given very simple patterns for its inflow and outflow. Subject learning was slow and ultimately sub-optimal even in the simple conditions of the task, for example, that of controlling the system due to an increasing inflow and zero outflow (Gonzalez & Dutt, 2007; Dutt & Gonzalez, 2007). Moreover, Cronin and Gonzalez (2007) presented subjects with a series of manipulations related to the form of information display, context of the task, incentives and others factors intended to help the subject understand the

task, demonstrating that the difficulty in understanding the DSF is not due to lack of information or the form of information presentation.

For all the difficulty subjects have controlling DSF, the task environment itself is easily modeled and extended. The state of the task environment is completely determined by the functional relationship among inflow, outflow, user action and stock, while the functions themselves can be modified in direct ways. For example, stochastic “noise” can be added to the functions that control environmental inflow and outflow to explore the effects of uncertainty; the addition of different or variable delays between user actions and outcomes changes the nature of the dynamic complexity; finally, the task lends itself to the exploration of team or adversarial performance simply by allowing another agent to control the environmental inputs and outputs.

Participating in the DSF Model Comparison

Participation in this model comparison begins with a visit to the DSF Model Comparison website: <http://www.cmu.edu/ddmlab/ModelDSF>. There, potential participants will be asked to register for the competition. Registration is free, but is required so that we can plan to allocate adequate resources to the evaluation of the participants' models (as described below).

At the website, participants will find a more detailed description of the DSF task and a free downloadable version of the task environment. The DSF task environment requires a Windows platform and can be run in two modes. First, the DSF can be run as a live experiment so that participants can interact with exactly the same task environment the subjects used in the experiments. In this way, modelers can gain hands-on experience with the task and use this experience to inform the development of their own models. Second, the DSF environment can be run as a constructive simulation, without the user interface, in a faster-than-real time mode with the participants' computational models interacting directly with the task environment.

The DSF uses a TCP/IP socket protocol to communicate with external models. Details about the “client” requirements and the communication syntax will be available on the website, along with example software code for connecting to the DSF.

Once participants have established a connection to the DSF environment, we invite them to calibrate their models running against the “training” protocols and comparing model performance against human performance data. Both the training protocols and data will be available from the website. In this way, participants will be able to gauge

whether their models are capable of simulating the basic effects seen in human control of the DSF task. Our past experience suggests that this will lead to an iterative development process where models are continually refined as they are run under different experimental protocols and against different data sets.

Model comparison begins only after participants are satisfied with the performance they have achieved on the training data. At that point, participants will submit executable version of their model through the website to be run against “transfer” protocols. As we indicated above, the DSF task supports several interesting variants. We are currently running pilot studies with human subjects to identify robust effects under these various conditions. The choice of specific transfer conditions will be entirely at our discretion and submitted models will be run under these conditions as-is.

Our goal for this blind evaluation under the transfer condition is not to hamstring participants, but to see how well their models generalize without the benefit of continual tweaking or tuning. Assessing robustness under the transfer condition is an important factor to consider when we investigate the invariance of architectural approaches. That said, goodness-of-fit under the training and transfer conditions is not the only factor will use in our comparison effort. In addition to submitting executable versions of their models, we will require participants to submit written accounts of their development efforts and detailed explanations of the mechanisms their models implement. As we discuss below, this is where model comparisons bear the most fruit. Again, based on our past experience, we recognize that it is difficult to explain the workings of a cognitive model to the uninitiated, but it is exactly that level of detail that is required to understand what has been accomplished.

On the basis of both model performance and written explanation, we will select three participants to present their work at the 2009 International Conference on Cognitive Modeling (http://web.mac.com/howesa/Site/ICCM_09.html). We will also cover basic travel expense to that conference. Finally, participants will be invited to prepare manuscripts for publication in a Special Issue of the Journal for Cognitive Systems Research (<http://www.sts.rpi.edu/~rsun/journal.html>) devoted to the topic of model comparison.

Model Comparison as Science

The call for the development of an artificial general intelligence is meant to mark a turn away from the development of “narrow AI.” From that perspective, a model comparison might seem to be an unwelcome return to the development of one-off systems engineered to excel

only on well-specified highly-constrained tasks. It would be a mistake, however, to view the outcome of a model comparison as merely a matter of identifying the approach that produces the best fit to the human performance data on a specific task. Rather, a goodness-of-fit measure is only a minimum standard for the more detailed consideration of the computational mechanisms that lead to that fit. Insofar as these mechanisms implement invariant structures, they shed light on the general nature of cognition. But the devil is in the details; understanding whether an architecture actually constrains the modeling approach and thereby shed some insight into the general features of cognition, or whether it merely disguises the skill of the clever modeler is never easy.

This difficulty is compounded by several other factors. First, as Roberts and Pashler (2000) have pointed out, good human performance data are hard to come by and it is harder still to see these data, by themselves, can undergird an *experimentum crucis* among different modeling approaches. Simply put, even good fits to good data will underdetermine the choices of architecture. This is not merely a problem of loose data, but is also due to one of the defining insights of computation, that of Turing equivalence and the related notion in the philosophy of mind of multiple realizability. The fact that any algorithm can be implemented by any number of Turing-equivalent mechanisms all but guarantees some degree of underdetermination when we consider the relationship between model and theory. Unless one is willing to engage in a question-begging argument about the computational nature of mind, Turing equivalence does not guarantee theoretical equivalence when it comes to cognitive modeling and different computational mechanisms will come with different theoretical implications.

While some might argue that this latter problem can be addressed by fixing the appropriate level of abstraction this otherwise sound advice has had the practical effect of allowing the modeler to decide what the appropriate relationship is between model and theory. Moreover, proposing an abstraction hierarchy, no matter how elegant or appealing, is not the same thing as discovering a natural kind, and it remains an empirical endeavor to establish whether the abstractions we impose really carve the nature of cognition at the joints. Thus, correspondence is too often asserted by fiat, and notions like “working memory,” “situation awareness” “problem detection” and such are reduced to simple computational mechanisms without any serious theoretical consideration. Those concerned about the implementation of a general intelligence are left to wonder whether if-then-else is all there is to it.

A number of tests for a general theory of intelligence have been advanced (e.g. Cohen, 2005; Selman et al, 1996;

Anderson & Lebiere, 2003). A key common aspect is to enforce generality in approach, in order to prevent special-purpose optimization to narrow tasks and force integration of capabilities. One can view that strategy as effectively overwhelming the degrees of freedom in the architecture with converging constraints in the data. However, precise computational specifications of those tests have to tread a tight rope between requiring unreasonable amounts of effort in modeling broad and complex tasks and falling back into narrow task specifications that will again favor engineered, optimized approaches. This model competition is our attempt at testing general cognitive capabilities in an open-ended task while offering low barriers to entry.

We see model comparison as one solution to these problems. Model comparison is not just a practical solution for understanding how different systems work, but a theoretical prescription for identifying invariant structures among different approaches, seeing how they are, in fact, applied to specific problems and a way of seeing past the buzzwords to the mechanism that might finally illuminate what is needed to realize an artificial general intelligence.

References

- Anderson, J. R. & Lebiere, C. L. 2003. The Newell test for a theory of cognition. *Behavioral & Brain Sciences* 26, 587-637.
- Asada, M., Kitano, H., Noda, I., and Veloso, M. 1999. RoboCup: Today and tomorrow – What we have have learned. *Artificial Intelligence*, 110:193–214.
- Brehmer, B., & Allard, R. 1991. Dynamic decision-making: The effects of task complexity and feedback delay. In J. Rasmussen, B. Brehmer & J. Leplat (Eds.), *Distributed decision making: Cognitive models of cooperative work* (pp. 319-334). Chichester: Wiley.
- Billings, D. 2000. The First International RoShamBo Programming Competition. *ICGA Journal*, Vol. 23, No. 1, pp. 42-50.
- Cohen, P. 2005. If Not Turing's Test, Then What? *AI Magazine* 26(4): 61–67.
- Cronin, M., & Gonzalez, C. 2007. Understanding the building blocks of dynamic systems. *System Dynamics Review*. 23(1), 1-17.
- Cronin, M., Gonzalez, C., & Serman, J. D. 2008. Why don't well-educated adults understand accumulation? A challenge to researchers, educators and citizens. In press. *Organizational Behavior and Human Decision Processes*.
- Diehl, E., & Serman, J. D. 1995. Effects of feedback complexity on dynamic decision-making. *Organizational Behavior and Human Decision Processes*, 62(2), 198-215.
- Dutt, V. & Gonzalez, C. 2008. Human Perceptions of Climate Change. The 26th International Conference of the System Dynamics Society. (pp.). Athens, Greece: System Dynamics Society.
- Dutt, V. & Gonzalez, C. 2007. Slope of Inflow Impacts Dynamic Decision Making. The 25th International Conference of the System Dynamics Society. (pp. 79). Boston, MA: System Dynamics Society.
- Erev, I., Ert, E., Roth, A. E., Haruvy, E., Herzog, S., Hau, R., Hertwig, R., Stewart, T., West, R., & Lebiere, C. (submitted). A choice prediction competition, for choices from experience and from description. *Journal of Behavioral Decision Making*.
- Foyle, D. & Hooey, B. 2008. *Human Performance Modeling in Aviation*. Mahwah, NJ: Erlbaum.
- Gluck, K., & Pew, R. 2005. *Modeling Human Behavior with Integrated Cognitive Architectures*. Mahwah, NJ: Erlbaum.
- Gonzalez, C., & Dutt, V. 2007. Learning to control a dynamic task: A system dynamics cognitive model of the slope effect. In *Proceedings of the 8th International Conference on Cognitive Modeling*, Ann Arbor, MI.
- Hsu, F-H. 2002. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.
- Lebiere, C., & Bothell, D. 2004. Competitive Modeling Symposium: PokerBot World Series. In *Proceedings of the Sixth International Conference on Cognitive Modeling*, Pp. 32-32.
- Lebiere, C., Gonzalez, C., & Warwick, W. (submitted). Emergent Complexity in a Dynamic Control Task: Model Comparison.
- Newell, A. 1990. *Unified Theories of Cognition*. Harvard University Press.
- Roberts, S. and Pashler, H. 2000. "How Persuasive Is a Good Fit? A Comment on Theory testing." *Psychological Review* 107(2): pp358-367.
- Schunn, C. D. & Wallach, D. 2001. Evaluating goodness-of-fit in comparisons of models to data. Online manuscript. <http://lrddc.pitt.edu/schunn/gof/index.html>
- Selman, B., Brooks, R., Dean, T., Horvitz, E., Mitchell, T., & Nilsson, N. 1996. Challenge problems for artificial intelligence. In *Proceedings of the 13th Natl. Conf. on Artificial Intelligence (AAAI-96)*, Portland, OR, pp. 193-224.
- Thrun, S. et al. 2006. Stanley, the robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9), 661–692.
- Turing, A. 1950. Computing Machinery and Intelligence, *Mind* LIX (236): 433–460.
- Warwick, W., Allender, L., Strater, L., & Yen, J. 2008. AMBR Redux: Another Take on Model Comparison. Symposium given at the *Seventeenth Conference on Behavior Representation and Simulation*. Providence, RI.

Incorporating Planning and Reasoning into a Self-Motivated, Communicative Agent

Daphne Liu and Lenhart Schubert

Department of Computer Science
University of Rochester
Rochester, NY USA

Abstract

Most work on self-motivated agents has focused on acquiring utility-optimizing mappings from states to actions. But such mappings do not allow for explicit, reasoned anticipation and planned achievement of future states and rewards, based on symbolic knowledge about the environment and about the consequences of the agent's own behavior. In essence, such agents can only behave reflexively, rather than reflectively. Conversely, planning and reasoning have been viewed within AI as geared towards satisfaction of explicitly specified user goals, without consideration of the long-range utility of the planner/reasoner's choices. We take a step here towards endowing a self-motivated, utility-optimizing agent with reasoning and planning abilities, and show that such an agent benefits both from its knowledge about itself and its environment, and from exploiting opportunities as it goes. Our simulated simple agent can cope with unanticipated environmental events and can communicate with the user or with other agents.

Introduction

There is rather broad agreement in AI that general human-like intelligent behavior, apart from lower-level activities like perception and reflexes, is guided by planning and reasoning. However, planning and reasoning have traditionally been understood as aimed at the fulfillment of specified user goals, rather than as internally motivated by consideration of the long-range utility of the planner/reasoner's choices. Conversely, research on self-motivated agents has focused almost exclusively on acquiring utility-optimizing mappings from states to actions, without reasoned anticipation and planned attainment of future states and rewards based on symbolic knowledge about the environment and consequences of the agent's own behavior. These observations have motivated our work on *explicit self-awareness* (Sch05) and more thoughtful self-motivation in agents.

Here we present a self-aware and self-motivated agent that thinks ahead, plans and reasons deliberately, and acts reflectively, both by drawing on knowledge about itself and its environment and by seizing opportunities

to optimize its utility. As an initial step towards a full conversation agent, our simple agent can communicate with the user or with other agents in addition to coping with unforeseen environmental events while acting opportunistically.

In the following sections, we discuss the notions of *explicit self-awareness* (Sch05) and thoughtful self-motivation, as well as how they are realized in our system. Then we outline our system and describe how our agent benefits from self-awareness, thoughtful self-motivation, and opportunistic choices. We conclude with a summary and a discussion of future work.

Explicit Self-Awareness

Explicit self-awareness was characterized by Schubert (Sch05) as being both human-like and explicit. Specifically, it is human-like in that an explicitly self-aware agent must have a well-elaborated human-like model of the world, including a model of itself and its relationships to the world. The self-model encompasses its beliefs, desires, intentions, knowledge, abilities, autobiography, the current situation, etc.; in addition, the agent must be capable of goal- and utility-directed reasoning and planning.

In addition to prescribing the aforementioned human-like capabilities, explicit self-awareness is explicit in three respects. First, an agent's representation of self-knowledge must be amenable to self-observation and use by the agent (and for engineering reasons, browsable and comprehensible to the designer). Second, explicit self-awareness must be conveyable by the agent, through language or other modalities. Third, the agent's self-knowledge must be amenable to inferences in conjunction with world knowledge.

Schubert (Sch05) enumerated reasons motivating the need for explicit self-awareness. First, given its bootstrapping potential with respect to meta-control, error recovery, autoepistemic reasoning, and learning of skills or facts, explicit self-awareness would help expand the frontiers of AI. Moreover, an explicitly self-aware agent can interact with human users in a transparent, natural, and engaging manner by having a shared context. Lastly, operational, explicitly self-aware agents can serve as exemplars of entities whose internal ba-

sis for self-awareness can be analyzed by consciousness theorists wishing to better understand self-awareness.

For additional discussions of explicit self-awareness, see (MS05; MS07; MS08) and (Liu08). The last elaborates on the contrast with other (weaker) notions of self-awareness as exhibited by self-monitoring agents (e.g., the metacognitive loop of Anderson and Perlis (AP05)), self-explaining agents (e.g., SHRDLU by Winograd (Win71)), global workspace systems (e.g., the opportunistic planning model by Hayes-Roth and Hayes-Roth (HRHR79)), and adaptive and robust goal-directed systems (e.g., an antibody system combatting viral intrusions). These conceptions of self-awareness either do not assume a self-model, or do not assume integration the self-model into general reasoning mechanisms.

Our conception of self-awareness has much in common with that of McCarthy, who proposes a formalization in terms of a mental situation calculus (McC95). He postulates that a machine must declaratively represent its mental states in order to introspect – observe and reason about its mental states, including beliefs, desires, intentions, knowledge, abilities, and consciousness. Schubert’s proposal (Sch05) further specifies the knowledge representation and reasoning requirements for explicit self-awareness. In addition to a basic logical framework, these requirements include logical formulations of events, situations, attitudes, autoepistemic inferences, generic knowledge, and various metasyntactic devices such as axiom schemas, knowledge categorization, knowing or deriving a value, and experience summarization.

Our agent, dubbed ME for Motivated Explorer, to some extent meets the requirements for explicit self-awareness. ME knows specific facts about itself and the current situation, expressed as ground predications, as well as possessing general knowledge in the form of Horn-like clauses (but also allowing for reified propositions and questions). In effect, ME has a self-model that relates it to its simulated world and potentially the user. ME’s knowledge of the current situation is initialized with its initial location, its possessions, geographical knowledge about the world, the current state facts about itself, and its propositional attitudes. For example, facts (*book book5*), (*owns ME book5*), (*knows ME (that (likes Grunt book5))*), and (*knows ME (whether (readable book5))*) in ME’s knowledge base specify that ME owns the book *book5*, knows whether *book5* is readable, and knows that entity *Grunt* likes *book5*.

ME operates according to a plan of actions to be carried out, which ME dynamically modifies, evaluates for expected cumulative utility, and partially executes. Specifically, ME thinks ahead into the future and chooses to execute a seemingly best action. Such an action is one that constitutes the first action in some plan (with a limited horizon) that is judged to be executable from the current state, and in addition is anticipated to yield the highest cumulative utility among all such plans. Both the contemplated actions and the states

they are expected to lead to can contribute positively or negatively to the anticipated utility of a plan. As actions are performed, ME’s knowledge base, as well as the world state, will evolve accordingly. For instance, when ME obtains a book, the effect will be that ME has the book; this fact will enter ME’s knowledge base, and as a result it will *know* that it has the book.

ME is introspective in three key respects. First, ME has knowledge of what operators it can invoke and what goals it can readily achieve in a given state. Second, ME can introspect about what it knows and doesn’t know, and can handle propositional attitudes. Third, when ME performs an action, this is recorded in ME’s history list of all actions and exogenous events that have occurred thus far in the world. ME’s history list and knowledge base are both open to introspection, enabling ME to engage in more interesting question-answering with the human user and with other agents.

In a limited way, ME also meets the reasoning and communication requirements for explicitly self-aware agents. Apart from its ability to plan, it can also reason. In any given state, it performs bounded forward inference based on all of its current factual knowledge and all of its general quantified knowledge. One current limitation is that ME is excessively “skeptical”, in the sense that it presumes to be false any ground predication that it cannot establish. Conversely, ME depends on a “commonsense inertia” assumption that whatever was true in the past and has not observably become false remains true.

Thoughtful Self-Motivation

A dynamic world poses three main types of challenges to a planning agent; namely, unexpected changes can arise in the form of unexpected action failures, unexpected threats, and unexpected serendipitous opportunities. In essence, these unexpected eventualities are due to the agent’s incomplete or partial knowledge of the world. Thus the agent is necessarily confronted with the qualification and ramifications problems, that is, it simply does not know all the conditions for an action to succeed or all the possible consequences of an action, and so may experience unexpected outcomes. This is aggravated by the possibility of unpredictable exogenous events such as rain and fire. In the face of such indeterminacy, it is important that the agent act opportunistically in order to recover from (unexpected) failures, avoid (unexpected) threats, and pursue (unexpected) favorable opportunities in an appropriate and timely manner.

ME’s opportunistic behavior is the byproduct of its constant, step-by-step striving towards maximum cumulative utility. For instance, suppose ME chooses to walk from home to school as that seems to be a best action to take. While walking from home to school, it may encounter a fire that makes the road unnavigable, at which point ME, undeterred, will do another look-ahead into the future to select a best next action to take. Or, while walking from home to school, ME may

see an unclaimed ten-dollar bill along the way, at which point it may pocket it if it finds doing so sufficiently pleasing (i.e., if doing so turns out to be the first step of the (cumulatively) most promising course of action).

ME is self-motivated in the sense that it has its own metrics of rewards (and penalties), and is driven by the “desire” to maximize cumulative rewards, rather than by some particular symbolic goal assigned to it, and to be pursued at all costs. Nonetheless, it pursues goals, to the extent that those goals promise high returns. ME’s self-motivation is thoughtful, because of its grounding in reasoned look-ahead and evaluation. (Details concerning the look-ahead scheme are given in the following section.) Such deliberate self-motivation differs importantly from the impulsive self-motivation inherent in behavioral robots and reinforcement-learning agents, as most commonly understood. Broadly, such an agent functions in accord with a (preprogrammed or learned) *policy* that maps states of the world to the actions the agent should take in those states in order to maximize some overall reward criterion. However, neither the policy, nor the search for it, is guided by reasoning about future actions and situations, but instead both depend on the current state alone, and any past experience associated with it. In fact, reasoning typically is not an option, because states are typically not represented by symbolic descriptions, or in any case are not amenable to application of general planning and inference methods. The notions of beliefs, desires, and intentions are realized in only very elementary ways in such agents, if at all.

ME’s self-motivation does not necessarily imply selfishness. While it may find certain self-involved states and actions rewarding (e.g., eating) or displeasing (e.g., being tired), and this will definitely affect its behavior, ME can also experience vicarious satisfaction, for example by answering the human user’s questions, or, say, helping another agent in its world in some way. Notably, the anticipatory satisfaction in the look-ahead can also be used to implement curiosity, by making the acquisition of new knowledge, or going to as yet unvisited places, intrinsically satisfying for ME.

System Implementation

The knowledge, planning and behavior (both physical and dialog) of ME are programmed in LISP in a simulated world consisting of locations, connecting roads, and both animate entities (agents) and inanimate entities (objects) positioned at various locations. Some objects in the world may be consumable or portable and potentially useful to ME, while others might be harmful or mere obstacles. ME navigates the world interacting with the human user (in dialog) and with other entities in the world. All agents except ME are stationary and might be asked questions by ME and may supply things or information that ME wants upon request.

Creation of a world is enabled through commands for creating a road network, defining object types with various properties, and for placing instances of object

types, with additional properties, at various locations in the network. Miscellaneous general knowledge can also be added. The additional properties of an instantiated entity include its associated objects (such as possessions or parts), and initial state facts about it, such as location or edibility, and for agents, propositional attitudes (beliefs, wants). ME’s initial knowledge base contains the geographical knowledge about the world, general quantified conditional facts (with a conjunctive antecedent and a positive predication as consequent), and ME keeps a history list of all actions and exogenous events that have occurred so far in the simulated world. Examples of general knowledge might be properties of certain types of entities (e.g., that a sasquatch is an animate agent), or that certain conditions imply others (e.g., being asleep implies not being awake).

ME does not in general know the current facts, possessions, or propositional attitudes associated with other entities. However, all *non-occluded, local* facts about an entity become known to ME when ME is at the location of the entity. Occluded facts are determined by certain predicates (such as *hungry*, *knows*, or *contains*) being marked as occluded; as mentioned above, a fact with an occluded predicate is known initially only to the subject of the predication, if that subject is animate. For example, *hungry* might be an occluded predicate, but the subject (*term*) of a fact (*hungry* (*term*)) is assumed to know that it is hungry whenever this is true. Moreover, ME may discover occluded knowledge via appropriate actions. For instance, the action *open* applied to a box followed by *read-message* may cause ME to know the contents of the message if the box has a message in it.

ME’s action types have a list of parameters, a set of preconditions, a set of effects, and an associated value. One of the more unusual features is that both preconditions and effects allow for procedural evaluation or simplification, once all parameters are bound. In this way quantitative preconditions and effects can be handled quite effectively, as can side-effects such as ME producing a printed answer. As an example, consider the following operator *sleep* with formal fatigue and hunger level parameters *?f* and *?h*, respectively:

```
(setq sleep
  (make-op :name 'sleep :pars '(?f ?h)
    :preconds '((is_at ME home)
      (is_tired_to_degree ME ?t)
      (>= ?f 0.5)
      (is_hungry_to_degree ME ?h)
      (> ?f ?h)
      (not (there_is_a_fire)))
    :effects '((is_tired_to_degree ME 0)
      (not (is_tired_to_degree
        ME ?f))
      (is_hungry_to_degree ME
        (+ ?h 2)))
    :time-required '(* 4 ?f)
    :value '(* 2 ?f)))
```

From ME’s perspective, if it is at home, is more tired than hungry, is at least of fatigue level 0.5, and there is no fire, then it can sleep for a duration given by $(* 4 ?f)$

and, as a result, it will relieve its fatigue at the expense of increasing its hunger level by 2. Performing an instantiated *sleep* action will afford ME a net increase of ($* 2 ?f$) in its cumulative utility.

Instantiating an operator requires replacing its formal parameters with actual values through unifying the preconditions with facts in ME's current knowledge base, and such an instantiated action is considered applicable in the current state. At all times, ME maintains a plan comprised of a sequence of instantiated actions. Planning is accomplished by forward search from a given state, followed by propagating backward the anticipated rewards and costs of the various actions and states reached, to obtain a seemingly best sequence of actions. The forward search is constrained by a search beam, which specifies the allowable number of branches and the allowable operators for each search depth. Informed by this projective forward search, ME will then execute the first action of the seemingly best plan, and update its knowledge accordingly (in effect, observing non-occluded facts, including ones that have become false, in its local environment).

The simulated world is a dynamic one in which exogenous events such as fire and rain can spontaneously begin and end with some probability at each time step; therefore, unexpected changes can arise in the form of unexpected action failures, unexpected threats, and unexpected serendipitous opportunities. For example, a fire may start and disrupt ME's travel, or ME may scratch a lottery coupon and find that it has won one million dollars. Since the world is only partially known and partially predictable from ME's perspective, and since actions (such as traveling) can take multiple time steps, with the possibility of interference by exogenous events, we need to model "actual" actions in ME's world separately from ME's (STRIPS-like) conception of those actions.

The following is the stepwise version *sleep.actual* of the *sleep* operator:

```
(setq sleep.actual
  (make-op.actual :name 'sleep.actual
                 :pars '(?f ?h)
                 :startconds '((is_at ME home)
                               (is_tired_to_degree ME ?t)
                               (>= ?f 0.5)
                               (is_hungry_to_degree ME ?h)
                               (> ?f ?h))
                 :stopconds '((there_is_a_fire)
                              (is_tired_to_degree ME 0))
                 :deletes '((is_tired_to_degree ME ?#1)
                           (is_hungry_to_degree ME ?#2))
                 :adds '((is_tired_to_degree ME
                          (- ?f (* 0.5
                                   (elapsed_time?))))
                        (is_hungry_to_degree ME
                         (+ ?h (* 0.5
                                   (elapsed_time?))))))
```

The start conditions as given by *startconds* are the same except for removal of the (*there_is_a_fire*) formula. Notably, the actual action will continue for an-

other time step if and only if neither of its stop conditions as given by *stopconds* is true in the current state. If at least one of them is true in the current state, then the action will immediately terminate. Otherwise, the current state and ME's knowledge base will be updated with ME's lower fatigue level and higher hunger level.

ME currently has various operators at its disposal, enabling it to answer the user's yes/no questions and wh-questions, to walk, sleep, eat, drink, ask other agents whether something is true, play, read, withdraw money from a bank, buy something from a store, and work and save money. Moreover, via operator (*listen!*), the user can signal to ME that a question or assertion (in symbolic form, not in English at this point) is about to be sent, and ME will "hear" and save that assertion or question, and potentially respond. Since answering questions has been assigned a high utility, ME will prefer to answer questions and verbalize its responses (as English sentences printed on the screen). Alternatively, for diagnostic reasons, the user is also provided with the ability to peer into ME's knowledge base and obtain the answer to a question immediately, without having to ask ME a question.

Preliminary Results

To empirically demonstrate the benefits of explicit self-awareness and of opportunistic (but still thoughtful) behavior in a self-motivated agent, we have created scenarios allowing some initial ablation tests. Here we describe some as yet incomplete attempts to investigate ME's performance (in terms of cumulative utility) with and without self-knowledge, and with and without opportunistic tendencies.

In all scenarios, there are four locations *home*, *grove1*, *plaza1*, and *company1*, with road *path1* of length 2 connecting *home* and *grove1*, *path2* of length 3 connecting *home* and *plaza1*, and *path3* of length 2 connecting *grove1* and *company1* in the simulated world. Agent ME's knowledge base is initialized to reflect that it is at *home*, is not tired, has a thirst level of 4, has a hunger level of 2, and knows that *applejuice1* is potable and at *home*. Object *pizza1* is edible and at *plaza1*. Object *applejuice1* is potable and at *home*. Agent *guru* knows whether *applejuice1* is potable and whether *pizza1* is edible.

In addition, ME has a variety of operators at its disposal, enabling it to answer the user's yes/no questions and wh-questions, walk, sleep, eat, drink, ask other agents whether something is true, play, read, buy something from a store, and work and save money. Also there are two types of exogenous events, namely fire and rain. Provided there is no rain, a spontaneous fire has a 5% chance of starting; once it has started, it has a 50% chance of stopping, and it also goes out as soon as there is rain. Spontaneous rain has a 33% chance of starting; once it has started, it has a 25% chance of stopping.

In normal operation, ME will gain rewards from the actions it performs (e.g., roaming, eating, or an-

swering user queries) and the states it reaches (e.g., not being hungry, thirsty, or tired). One rather trivial way to ablate self-awareness is to eliminate all first-person knowledge such as (*is_at ME home*), (*is_hungry_to_degree ME 2*), etc., without altering operator definitions. In such a case, ME can no longer confirm the preconditions of its own actions, since these all involve facts about ME; thus, it is immobilized. But a more meaningful test of the effect of ablating first-person knowledge should replace ME's conceptions of its operators with ones that make no mention of the agent executing them, yet are still executable, perhaps with no effect or adverse effects in actuality, when actual preconditions unknown to ME are neglected. We would then expect relatively haphazard, unrewarding behavior, but this remains to be implemented.

A more interesting test of the advantages of self-awareness would be one focused on first-person *metaknowledge*, e.g., knowledge of type (*knows ME (whether (edible pizza1))*). Intuitively, given that ME can entertain such knowledge, and there are ways of finding out whether, for instance, (*edible pizza1*), ME should be able to plan and act more successfully than if its self-knowledge were entirely at the object- (non-meta-) level. In fact, this is why our scenarios include the above kind of meta-precondition in the definition of the *eat* operator, and why they include a guru who can advise ME on object edibility. Our experimentation so far successfully shows that ME does indeed ask the guru about the edibility of available items, and is thereby enabled to eat, and hence to thrive. However, ablating meta-level self-knowledge, much as in the case of object-level self-knowledge, should be done by replacing ME's conceptions of its operators with ones that do not involve the ablated predications (in this case, meta-level predications), while still keeping the actual workings of operators such as *eat* more or less unchanged. So in this case, we would want to make an *eat*-simulation either unrewarding or negatively rewarding if the object to be eaten is inedible. This would surely degrade ME's performance, but this also remains to be confirmed.

To investigate the effects that ablation of opportunistic behavior has on ME's cumulative utility, we will focus our attention on one representative scenario. Initially, ME is at home feeling hungry and thirsty, knows *applejuice1* at home to be potable, but does not know any item to be edible. To find out about the (only) edible item *pizza1*, ME must walk to *grove1* and ask *guru*. With sufficient lookahead, having knowledge about *pizza1* will incline ME to walk to *plaza1* and eat *pizza1* there. To entirely suppress ME's opportunistic behavior, we designate eating *pizza1* as ME's sole goal and make ME uninterested in any action other than asking *guru* to acquire food knowledge, traveling to reach *guru* and *pizza1*, and eating *pizza1*.

In this case, none of ME's actions are disrupted by any spontaneous fire, and upon accomplishing its goal of

eating *pizza1* after 18 steps, ME achieves a cumulative utility of 66.5. The sequence of actions and events, each annotated with its time of occurrence in reverse chronological order, is as follows:

```
((EAT PIZZA1 PLAZA1) 17), (FIRE 15), ((WALK HOME PLAZA1
PATH2) 14), ((WALK HOME PLAZA1 PATH2) 12), ((WALK GROVE1
HOME PATH1) 9), (RAIN 9), (FIRE 8), ((WALK GROVE1 HOME
PATH1) 5), (RAIN 5), ((ASK+WHETHER GURU (EDIBLE PIZZA1)
GROVE1) 3), (FIRE 2), ((WALK HOME GROVE1 PATH1) 1),
((WALK HOME GROVE1 PATH1) 0), (RAIN 0).
```

With its opportunistic behavior restored, ME thinks ahead into the future and chooses to execute a seemingly best action at each step, achieving a higher cumulative utility of 80.5 after 18 steps. The higher cumulative utility comes from ME's better choices of actions (e.g., drinking when thirsty); specifically, it is a direct result of ME's seizing the initial opportunity to drink the potable *applejuice1* to relieve its thirst, and ME can see and exploit such an opportunity because it is not blindly pursuing any one goal but rather is acting opportunistically. This case is shown below; no spontaneous fire disrupts any of ME's actions, and ME also finds out about *pizza1* and eventually eats it.

```
((EAT PIZZA1 PLAZA1) 17), (RAIN 16), ((WALK HOME PLAZA1
PATH2) 15), ((WALK HOME PLAZA1 PATH2) 13), (RAIN 13),
((WALK GROVE1 HOME PATH1) 11), (RAIN 11), ((WALK GROVE1
HOME PATH1) 10), (RAIN 9), ((ASK+WHETHER GURU
(EDIBLE PIZZA1) GROVE1) 8), (FIRE 7), ((WALK HOME GROVE1
PATH1) 0) 6), ((WALK HOME GROVE1 PATH1) 5), (RAIN 5),
(FIRE 2), ((DRINK 4 APPLEJUICE1 HOME) 0), (RAIN 0).
```

These preliminary results indicate that ME can indeed benefit from both self-awareness and opportunistic, thoughtful self-motivation. While the results are encouraging, we are planning on doing systematic evaluations of our hypotheses, as we will outline in the concluding section.

Conclusion

We have presented an explicitly self-aware and self-motivated agent that thinks ahead, plans and reasons deliberately, and acts reflectively, both by drawing on knowledge about itself and its environment and by seizing opportunities to optimize its cumulative utility.

We have pointed out that deliberate self-motivation differs significantly from the impulsive self-motivation in behavioral robots and reinforcement-learning agents, driven by policies acquired through extensive experience (or through imitation), but not guided by symbolic reasoning about current and potential future circumstances. Our approach can be viewed as an integration of two sorts of agent paradigms – behavioral (purely opportunistic) agents on the one hand, and planning-based (goal-directed) agents on the other. Agents in the former paradigm focus on the present state, using it to choose an action that conforms with a policy reflecting past reward/punishment experience. Agents in the latter paradigm are utterly future-oriented, aiming for some goal state while being impervious to the current state, except to the extent that the current state supports or fails to support steps toward that future

state. Some work in cognitive robotics (e.g., (TWN04; FFL04)) and in autonomous agents in computer games (e.g., (DEVG08)) intersects our approach, in that moves are chosen on the basis of the expected value of a sequence of moves (for instance, for a player in a Robocup world, or a person walking in a crowd, avoiding collisions). But generally these agents either are focused on externally supplied goals, or use feature-based rather than logical representations of states, and so cannot truly reason about them.

Additionally, we noted that our agent to some extent meets the knowledge representation and reasoning requirements (Sch05) for explicitly self-aware agents. Its ability to handle propositional attitudes (and in that sense metaknowledge) are particularly relevant to that point. Its self-knowledge, world knowledge, and introspection enable it to create and evaluate possible plans; furthermore, ME uses its current factual knowledge in any given state to perform bounded forward inference.

There are issues to address in our future work. First, ME is excessively skeptical in its presumption that ground predications are false whenever they cannot easily be established. Ignorance should not equal negation of knowledge. For example, not knowing if there is food does not mean there is no food; instead, being hungry and not knowing if there is food should prompt the agent to find out if there is food, that is, the agent should still consider pursuing eating. Some of this can probably be handled alternatively with “(¬) know-whether” propositions in ME’s knowledge base. If ME knows whether ϕ but ϕ is not inferable by or known to ME, then ME can conclude $\neg\phi$. If ϕ is not known to ME and ME does not know whether ϕ , then this might motivate ME to find out whether ϕ holds.

Eventually, degrees of uncertainty should be allowed for in ME’s knowledge. If ME has definite negative knowledge of a precondition, then ME certainly should not consider pursuing the action. On the other hand, if it is still possible that a precondition currently not known to be satisfiable might be true, and if ME would like to pursue this action, then ME should aim to prove or disprove this precondition. To make ME less skeptical, we can specify that if ME has been to a location, then any non-occluded facts at that location that are not known by ME to be true are false; otherwise, no such assumption should be made.

Ultimately, we envisage an explicitly self-aware and self-motivated conversation agent with knowledge- and suggestion-driven dialogue behavior. The agent’s behavior is ultimately driven by a planning executive that continually augments, modifies and partially executes a “life plan” that guides all of the agent’s deliberate actions, whether physical, verbal or mental. Given the large number of possible dialogue moves corresponding to particular dialogue states, it is desirable to guide such a continually evolving planning executive by “suggestions” (certain kinds of if-then rules) triggered by the current situation. Such suggestions could be extremely helpful in inclining the agent to particular actions in

particular situations.

By way of systematic demonstration, we would ideally want to show that our agent comes closer to reaping the maximum attainable cumulative utility than does either a purely opportunistic one or a solely goal-directed one. This raises the challenge of computing what the maximum attainable cumulative utility actually is in a given scenario. A possible approach to computing this maximum may be exhaustive forward search, as far into the future as possible. We would also want to explore probabilistic versions of such evaluations, in not-entirely-predictable worlds.

References

- M. L. Anderson and D. R. Perlis. Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. *Journal of Logic and Computation*, 15(1):21–40, 2005.
- J. Dinerstein, P.K. Egbert, D. Ventura, and M. Goodrich. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence*, 24(4):235–256, 2008.
- E. Ferrein, C. Fritz, and G. Lakemeyer. On-line decision-theoretic golog for unpredictable domains. In *Proceedings of the 27th German Conference on Artificial Intelligence (KI2004)*, pages 322–336, Ulm, Germany, 2004.
- B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 3:275–310, 1979.
- D. H. Liu. A survey of planning in intelligent agents: from externally motivated to internally motivated systems. Technical Report TR-2008-936, Department of Computer Science, University of Rochester, June 2008.
- J. McCarthy. Making robots conscious of their mental states. In *Machine Intelligence 15*, pages 3–17, 1995.
- F. Morbini and L.K. Schubert. Conscious agents. Technical Report TR-2005-879, Department of Computer Science, University of Rochester, September 2005.
- F. Morbini and L. Schubert. Towards realistic autocognitive inference. In *Logical Formalizations of Commonsense Reasoning, Papers from the AAI Spring Symposium*, pages 114–118, Menlo Park, CA, March 26–28 2007. AAAI Press.
- F. Morbini and L. Schubert. Metareasoning as an integral part of commonsense and autocognitive reasoning. In *AAAI-08 Workshop on Metareasoning*, pages 155–162, Chicago, IL, July 13–14 2008. Revised version in M. T. Cox & A. Raja (eds.), *Metareasoning: Thinking about Thinking*, MIT Press, to appear.
- L. Schubert. Some knowledge representation and reasoning requirements for self-awareness. In M. Anderson and T. Oates, editors, *Metacognition in Computation: Papers from the 2005 AAI Spring Symposium*, pages 106–113. AAAI Press, 2005.
- M. Tacke, T. Weigel, and B. Nebel. Decision-theoretic planning for playing table soccer. In *Proceedings of the 27th German Conference on Artificial Intelligence (KI2004)*, pages 213–225, Ulm, Germany, 2004.
- T. Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical Report 235, Massachusetts Institute of Technology, February 1971.

Program Representation for General Intelligence

Moshe Looks

Google, Inc.
madscience@google.com

Ben Goertzel

Novamente LLC
ben@novamente.net

Abstract

Traditional machine learning systems work with relatively flat, uniform data representations, such as feature vectors, time-series, and context-free grammars. However, reality often presents us with data which are best understood in terms of relations, types, hierarchies, and complex functional forms. One possible representational scheme for coping with this sort of complexity is computer programs. This immediately raises the question of how programs are to be best represented. We propose an answer in the context of ongoing work towards artificial general intelligence.

Background and Motivation

What are programs? The essence of programmatic representations is that they are well-specified, compact, combinatorial, and hierarchical. *Well-specified*: unlike sentences in natural language, programs are unambiguous; two distinct programs can be precisely equivalent. *Compact*: programs allow us to compress data on the basis of their regularities. Accordingly, for the purposes of this paper, we do not consider overly constrained representations such as the well-known conjunctive and disjunctive normal forms for Boolean formulae to be programmatic. Although they can express any Boolean function (data), they dramatically limit the range of data that can be expressed compactly, compared to unrestricted Boolean formulae. *Combinatorial*: programs access the results of running other programs (e.g. via function application), as well as delete, duplicate, and rearrange these results (e.g., via variables or combinators). *Hierarchical*: programs have intrinsic hierarchical organization, and may be decomposed into subprograms.

Baum has advanced a theory “under which one understands a problem when one has mental programs that can solve it and many naturally occurring variations” (Bau06). Accordingly, one of the primary goals of artificial general intelligence is systems that can represent, learn, and reason about such programs (Bau06; Bau04). Furthermore, integrative AGI systems such as Novamente (LGP04) may contain subsystems operating on programmatic representations. Would-be AGI systems with no direct support for programmatic representation will clearly need to represent procedures and procedural abstractions *somehow*. Alternatives such as recurrent neural networks have serious downsides, however, including opacity and inefficiency.

Note that the problem of how to represent programs for an AGI system dissolves in the limiting case of unbounded computational resources. The solution is algorithmic probability theory (Sol64), extended recently to the case of sequential decision theory (Hut05). The latter work defines the universal algorithmic agent AIXI, which in effect simulates all possible programs that are in agreement with the agent’s set of observations. While AIXI is uncomputable, the related agent $AIXI^l$ may be computed, and is superior to any other agent bounded by time t and space l (Hut05). The choice of a representational language for programs¹ is of no consequence, as it will merely introduce a bias that will disappear within a constant number of time steps.²

The contribution of this paper is providing practical techniques for approximating the ideal provided by algorithmic probability, based on what Pei Wang has termed the *assumption of insufficient knowledge and resources* (Wan06). Given this assumption, how programs are represented is of paramount importance, as is substantiated the next two sections, where we give a conceptual formulation of what we mean by *tractable program representations*, and introduce tools for formalizing tractability. The fourth section of the paper proposes an approach for tractably representing programs. The fifth and final section concludes and suggests future work.

Representational Challenges

Despite the advantages outlined in the previous section, there are a number of challenges in working with programmatic representations:

- **Open-endedness** – in contrast to other knowledge representations current in machine learning, programs vary in size and “shape”, and there is no obvious problem-independent upper bound on program size. This makes it difficult to represent programs as points in a fixed-dimensional space, or to learn programs with algorithms that assume such a space.
- **Over-representation** – often, syntactically distinct programs will be semantically identical (i.e. represent the same underlying behavior or functional mapping).

¹As well as a language for proofs in the case of $AIXI^l$.

²The universal distribution converges quickly (Sol64).

Lacking prior knowledge, many algorithms will inefficiently sample semantically identical programs repeatedly (GBK04; Loo07b).

- **Chaotic Execution** – programs that are very similar, syntactically, may be very different, semantically. This presents difficulties for many heuristic search algorithms, which require syntactic and semantic distance to be correlated (TVCC05; Loo07c).
- **High resource-variance** – programs in the same space vary greatly in the space and time they require to execute.

Based on these concerns, it is no surprise that search over program spaces quickly succumbs to combinatorial explosion, and that heuristic search methods are sometimes no better than random sampling (LP02). Regarding the difficulties caused by over-representation and high resource-variance, one may of course object that determinations of e.g. programmatic equivalence for the former, and e.g. halting behavior for the latter, are uncomputable. Given the assumption of insufficient knowledge and resources, however, these concerns dissolve into the larger issue of computational intractability and the need for efficient heuristics. Determining the equivalence of two Boolean formulae over 500 variables by computing and comparing their truth tables is trivial from a computability standpoint, but, in the words of Leonid Levin, “only math nerds would call 2^{500} finite” (Lev94). Similarly, a program that never terminates is a special case of a program that runs too slowly to be of interest to us.

In advocating that these challenges be addressed through “better representations”, we do not mean merely trading one Turing-complete programming language for another; in the end it will all come to the same. Rather, we claim that to tractably learn and reason about programs requires us to have prior knowledge of programming language semantics. The mechanism whereby programs are executed is known *a priori*, and remains constant across many problems. We have proposed, by means of exploiting this knowledge, that programs be represented in normal forms that preserve their hierarchical structure, and heuristically simplified based on reduction rules. Accordingly, one formally equivalent programming language may be preferred over another by virtue of making these reductions and transformations more explicit and concise to describe and to implement.

What Makes a Representation Tractable?

Creating a comprehensive formalization of the notion of a *tractable program representation* would constitute a significant achievement; and we will not fulfill that summons here. We will, however, take a step in that direction by enunciating a set of positive principles for tractable program representations, corresponding closely to the list of representational challenges above. While the discussion in this section is essentially conceptual rather than formal, we will use a bit of notation to ensure clarity of expression; S to denote a space of programmatic functions of the same type (e.g. all pure *Lisp* λ -expressions mapping from lists to numbers), and B to denote a metric space of *behaviors*.

In the case of a deterministic, side-effect-free program, execution maps from programs in S to points in B , which will have separate dimensions for function outputs across various inputs of interest, as well as dimensions corresponding to the time and space costs of executing the program. In the case of a program that interacts with an external environment, or is intrinsically nondeterministic, execution will map from S to probability distributions over points in B , which will contain additional dimensions for any side-effects of interest that programs in S might have. Note the distinction between *syntactic distance*, measured as e.g. tree-edit distance between programs in S , and *semantic distance*, measured between programs’ corresponding points in or probability distributions over B . We assume that semantic distance accurately quantifies our preferences in terms of a weighting on the dimensions of B ; i.e., if variation along some axis is of great interest, our metric for semantic distance should reflect this.

Let \mathcal{P} be a probability distribution over B that describes our knowledge of what sorts of problems we expect to encounter, and let $R(n) \subseteq S$ be the set of all of the programs in our representation with (syntactic) size no greater than n . We will say that “ $R(n)$ d -covers the pair (B, \mathcal{P}) to extent p ” if p is the probability that, for a random behavior $b \in B$ chosen according to \mathcal{P} , there is some program in R whose behavior is within semantic distance d of b . Then, some among the various properties of tractability that seem important based on the above discussion are as follows:

- for fixed d , p quickly goes to 1 as n increases,
- for fixed p , d quickly goes to 0 as n increases,
- for fixed d and p , the minimal n needed for $R(n)$ to d -cover (B, \mathcal{P}) to extent p should be as small as possible,
- *ceteris paribus*, syntactic and semantic distance (measured according to \mathcal{P}) are highly correlated.

Since execution time and memory usage measures may be incorporated into the definition of program behavior, minimizing chaotic execution and managing resource variance emerges conceptually here as a subclass of maximizing correlation between syntactic and semantic distance. Minimizing over-representation follows from the desire for small n : roughly speaking the less over-representation there is, the smaller average program size can be achieved.

In some cases one can empirically demonstrate the tractability of representations without any special assumptions about \mathcal{P} : for example in prior work we have shown that adoption of an appropriate hierarchical normal form can generically increase correlation between syntactic and semantic distance in the space of Boolean functions (Loo06; Loo07c). In this case we may say that we have a *generically tractable* representation. However, to achieve tractable representation of more complex programs, some fairly strong assumptions about \mathcal{P} will be necessary. This should not be philosophically disturbing, since it’s clear that human intelligence has evolved in a manner strongly conditioned by certain classes of environments; and similarly, what we need to do to create a viable program representation system for pragmatic AGI usage is to achieve tractability relative to the

distribution \mathcal{P} corresponding to the actual problems the AGI is going to need to solve. Formalizing the distributions \mathcal{P} of real-world interest is a difficult problem, and one we will not address here. However, we hypothesize that the representations presented in the following section may be tractable to a significant extent irrespective³ of \mathcal{P} , and even more powerfully tractable with respect to this as-yet unformalized distribution. As weak evidence in favor of this hypothesis, we note that many of the representations presented have proved useful so far in various narrow problem-solving situations.

(Postulated) Tractable Representations

We use a simple type system to distinguish between the various normal forms introduced below. This is necessary to convey the minimal information needed to correctly apply the basic functions in our canonical forms. Various systems and applications may of course augment these with additional type information, up to and including the satisfaction of arbitrary predicates (e.g. a type for prime numbers). This can be overlaid on top of our minimalist system to convey additional bias in selecting which transformations to apply, and introducing constraints as necessary. For instance, a call to a function expecting a prime number, called with a potentially composite argument, may be wrapped in a conditional testing the argument's primality. A similar technique is used in the normal form for functions to deal with list arguments that may be empty.

Normal Forms

Normal forms are provided for *Boolean* and *number* primitive types, and the following parametrized types:

- list types, $list_T$, where T is any type,
- tuple types, $tuple_{T_1, T_2, \dots, T_N}$, where all T_i are types, and N is a positive natural number,
- enum types, $\{s_1, s_2, \dots, s_N\}$, where N is a positive number and all s_i are unique identifiers,
- function types $T_1, T_2, \dots, T_N \rightarrow O$, where O and all T_i are types,
- action result types.

A list of type $list_T$ is an ordered sequence of any number of elements, all of which must have type T . A tuple of type $tuple_{T_1, T_2, \dots, T_N}$ is an ordered sequence of exactly N elements, where every i th element is of type T_i . An enum of type $\{s_1, s_2, \dots, s_N\}$ is some element s_i from the set. Action result types concern side-effectful interaction with some world external to the system (but perhaps simulated, of course), and will be described in detail in their subsection below. Other types may certainly be added at a later date, but we believe that those listed above provide sufficient expressive power to conveniently encompass a wide range of programs, and serve as a compelling proof of concept.

The normal form for a type T is a set of elementary functions with codomain T , a set of constants of type T , and a tree grammar. Internal nodes for expressions described by

³Technically, with only weak biases that prefer smaller and faster programs with hierarchical decompositions.

the grammar are elementary functions, and leaves are either U_{var} or $U_{constant}$, where U is some type (often $U = T$).

Sentences in a normal form grammar may be transformed into normal form expressions as follows. The set of expressions that may be generated is a function of a set of bound variables and a set of external functions (both bound variables and external functions are typed):

- $T_{constant}$ leaves are replaced with constants of type T ,
- T_{var} leaves are replaced with either bound variables matching type T , or expressions of the form $f(expr_1, expr_2, \dots, expr_M)$, where f is an external function of type $T_1, T_2, \dots, T_M \rightarrow T$, and each $expr_i$ is a normal form expression of type T_i (given the available bound variables and external functions).

Boolean Normal Form The elementary functions are *and*, *or*, and *not*. The constants are $\{true, false\}$. The grammar is:

```
bool_root = or_form | and_form
           | literal | bool_constant
literal   = bool_var | not( bool_var )
or_form   = or( {and_form | literal}{2,} )
and_form  = and( {or_form | literal}{2,} ) .
```

The construct $foo\{x, \}$ refers to x or more matches of foo (e.g. $\{x | y\}\{2, \}$ is two or more items in sequences where each item is either an x or a y).

Number Normal Form The elementary functions are *times* and *plus*. The constants are some subset of the rationals (e.g. those with IEEE single-precision floating-point representations). The grammar is:

```
num_root  = times_form | plus_form
           | num_constant | num_var
times_form = times( {num_constant |
                   plus_form}
                   plus_form{1,} )
           | num_var
plus_form  = plus( {num_constant |
                  times_form}
                 times_form{1,} )
           | num_var .
```

List Normal Form For list types $list_T$, the elementary functions are *list* (an n -ary list constructor) and *append*. The only constant is the empty list (*nil*). The grammar is:

```
list_T_root = append_form | list_form
            | list_T_var | list_T_constant
append_form = append( {list_form |
                    list_T_var}{2,} )
list_form   = list( T_root{1,} ) .
```

Tuple Normal Form For tuple types $tuple_{T_1, T_2, \dots, T_N}$, the only elementary function is the tuple constructor (*tuple*). The constants are $T_1_constant \times T_2_constant \times \dots \times T_N_constant$. The normal form is either a constant, a var, or $tuple(T_1_root T_2_root \dots T_N_root)$.

Enum Normal Form Enums are atomic tokens with no internal structure - accordingly, there are no elementary functions. The constants for the enum $\{s_1, s_2, \dots, s_N\}$ are the s_i s. The normal form is either a constant or a var.

Function Normal Form For $T_1, T_2, \dots, T_N \rightarrow O$, the normal form is a lambda-expression of arity N whose body is of type O . The list of variable names for the lambda-expression is not a “proper” argument - it does not have a normal form of its own. Assuming that none of the T_i s is a list type, the body of the lambda-expression is simply in the normal form for type O (with the possibility of the lambda-expressions arguments appearing with their appropriate types). If one or more T_i s are list types, then the body is a call to the *split* function, with all arguments in normal form.

Split is a family of functions with type signatures

$$(T_1, list_{T_1}, T_2, list_{T_2}, \dots, T_k, list_{T_k} \rightarrow O),$$

$$tuple_{list_{T_1}, O}, tuple_{list_{T_2}, O}, \dots, tuple_{list_{T_k}, O} \rightarrow O.$$

To evaluate *split*($f, tuple(l_1, o_1), tuple(l_2, o_2), \dots, tuple(l_k, o_k)$), the list arguments l_1, l_2, \dots, l_k are examined sequentially. If some l_i is found that is empty, then the result is the corresponding value o_i . If all l_i are nonempty, we deconstruct each of them into $x_i : xs_i$, where x_i is the first element of the list and xs_i is the rest. The result is then $f(x_1, xs_1, x_2, xs_2, \dots, x_k, xs_k)$. The *split* function thus acts as an implicit case statement to deconstruct lists only if they are nonempty.

Action Result Normal Form An action result type *act* corresponds to the result of taking an action in some world. Every action result type has a corresponding world type, *world*. Associated with action results and worlds are two special sorts of functions.

- **Perceptions** - functions that take a *world* as their first argument and regular (non-world and non-action-result) types as their remaining arguments, and return regular types. Unlike other function types, the result of evaluating a perception call may be different at different times.
- **Actions** - functions that take a *world* as their first argument and regular types as their remaining arguments, and return action results (of the type associated with the type of their world argument). As with perceptions, the result of evaluating an action call may be different at different times. Furthermore, actions may have side-effects in the associated world that they are called in. Thus, unlike any other sort of function, actions *must* be evaluated, even if their return values are ignored.

Other sorts of functions acting on worlds (e.g. ones that take multiple worlds as arguments) are disallowed.

Note that an action result expression cannot appear nested inside an expression of any other type. Consequently, there is no way to convert e.g. an action result to a Boolean, although conversion in the opposite direction is permitted. This is required because mathematical operations in our language have classical mathematical semantics; x and y must equal y and x , which will not generally be the case if x or y can have side-effects. Instead, there are special sequential versions of logical functions which may be used instead.

The elementary functions for action result types are *and_{seq}* (sequential and, equivalent to C 's short-circuiting &&), *or_{seq}* (sequential or, equivalent to C 's short-circuiting ||), and *fails* (negates success to failure and vice versa).

The constants may vary from type to type but must at least contain *success* and *failure*, indicating absolute success/failure in execution.⁴ The normal form is as follows:

```
act_root      = orseq_form | andseq_form
              | seqlit
seqlit        = act | fails( act )
act           = act_constant | act_var
orseq_form    = orseq( {andseq_form |
                      seqlit}{2,} )
andseq_form   = andseq( {orseq_form
                      | seqlit}{2,} ) .
```

Program Transformations

A program transformation is any type-preserving mapping from expressions to expressions. Transformations may be guaranteed to preserve semantics. When doing program evolution there is an intermediate category of fitness preserving transformations that may alter semantics. In general, the only way that fitness preserving transformations will be uncovered is by scoring programs that have had their semantics potentially transformed to determine their fitness.

Reductions These are semantics preserving transformations that do not increase some size measure (typically number of symbols), and are idempotent. For example, $and(x, x, y) \rightarrow and(x, y)$ is a reduction for the *Boolean* type. A set of *canonical reductions* is defined for every type with a normal form. For the *number* type, the simplifier in a computer algebra system may be used. The full list of reductions is omitted in this paper for brevity. An expression is *reduced* if it maps to itself under all canonical reductions for its type, and all of its subexpressions are reduced.

Another important set of reductions are the *compressive abstractions*, which reduce or keep constant the size of expressions by introducing new functions. Consider

```
list( times( plus( a, p, q ) r ),
      times( plus( b, p, q ) r ),
      times( plus( c, p, q ) r ) ) ,
```

which contains 19 symbols. Transforming this to

```
f( x ) = times( plus( x, p, q ) r )
list( f( a ), f( b ), f( c ) )
```

reduces the total number of symbols to 15. One can generalize this notion to consider compressive abstractions across a set of programs. Compressive abstractions appear to be rather expensive to uncover, although perhaps not prohibitively so (the computation is easily parallelized).

Neutral Transformations Semantics preserving transformations that are not reductions are not useful on their own - they can only have value when followed by transformations from some other class. This class of transformations is thus more speculative than reductions, and more costly to consider - cf. (Ols95).

- **Abstraction** - given an expression E containing non-overlapping subexpressions E_1, E_2, \dots, E_N , let E' be E

⁴A $do(arg_1, arg_2, \dots, arg_N)$ statement (known as *progn* in Lisp), which evaluates its arguments sequentially regardless of success or failure, is equivalent to $and_{seq}(or_{seq}(arg_1, success), or_{seq}(arg_2, success), \dots, or_{seq}(arg_N, success))$.

with all E_i replaced by the unbound variables v_i . Define the function $f(v_1, v_2, \dots, v_3) = E'$, and replace E with $f(E_1, E_2, \dots, E_N)$. Abstraction is distinct from compressive abstraction because only a single call to the new function f is introduced.⁵

- **Inverse abstraction** - replace a call to a user-defined function with the body of the function, with arguments instantiated (note that this can also be used to partially invert a compressive abstraction).
- **Distribution** - let E be a call to some function f , and let E' be a subexpression of E 's i th argument that is a call to some function g , such that f is distributive over g 's arguments, or a subset thereof. We shall refer to the actual arguments to g in these positions in E' as x_1, x_2, \dots, x_n . Now, let $D(F)$ be the function that is obtained by evaluating E with its i th argument (the one containing E') replaced with the expression F . Distribution is replacing E with E' , and then replacing each x_j ($1 \leq j \leq n$) with $D(x_j)$. For example, consider

```
plus( x, times( y, ifThenElse( cond,
                               a, b ) ) ) .
```

Since both *plus* and *times* are distributive over the result branches of *ifThenElse*, there are two possible distribution transformations, giving the expressions

```
ifThenElse( cond,
            plus( x, times( y, a ) ),
            plus( x, times( y, b ) ) ),
plus( x ( ifThenElse( cond,
                    times( y, a ),
                    times( y, b ) ) ) ) .
```

- **Inverse distribution** - the opposite of distribution. This is nearly a reduction; the exceptions are expressions such as $f(g(x))$, where f and g are mutually distributive.
- **Arity broadening** - given a function f , modify it to take an additional argument of some type. All calls to f must be correspondingly broadened to pass it an additional argument of the appropriate type.
- **List broadening**⁶ - given a function f with some i th argument x of type T , modify f to instead take an argument y of type $list_T$, which gets split into $x : xs$. All calls to f with i th argument x' must be replaced by corresponding calls with i th argument $list(x')$.
- **Conditional insertion** - an expression x is replaced by *ifThenElse*(*true*, x , y), where y is some expression of the same type of x .

As a technical note, action result expressions (which may cause side-effects) complicate neutral transformations. Specifically, abstractions and compressive abstractions must take their arguments lazily (i.e. not evaluate them before the function call itself is evaluated), in order to be neutral. Furthermore, distribution and inverse distribution may only be applied when f has no side-effects that will vary (e.g.

⁵In compressive abstraction there must be at least two calls in order to avoid increasing the number of symbols.

⁶Analogous tuple-broadening transformations may be defined as well, but are omitted for brevity.

be duplicated or halved) in the new expression, or affect the nested computation (e.g. change the result of a conditional). Another way to think about this issue is to consider the action result type as a lazy domain-specific language embedded within a pure functional language (where evaluation order is unspecified). Spector has performed an empirical study of the tradeoffs in lazy vs. eager function abstraction for program evolution (Spe96).

The number of neutral transformation applicable to any given program grows quickly with program size.⁷ Furthermore, synthesis of complex programs and abstractions does not seem to be possible without them. Thus, a key hypothesis of any approach to AGI requiring significant program synthesis, without assuming the currently infeasible computational capacities required to brute-force the problem, is that the inductive bias to select promising neutral transformations can be learned and/or programmed. Referring back to the initial discussion of what constitutes a tractable representation, we speculate that perhaps, whereas well-chosen reductions are valuable for generically increasing program representation tractability, well-chosen neutral transformations will be valuable for increasing program representation tractability relative to distributions \mathcal{P} to which the transformations have some (possibly subtle) relationship.

Non-Neutral Transformations Non-neutral transformations may encompass the general class defined by removal, replacement, and insertion of subexpressions, acting on expressions in normal form, and preserving the normal form property. Clearly these transformations are sufficient to convert any normal form expression into any other. What is desired is a subset of these transformations that is combinatorially complete, where each individual transformation is nonetheless a semantically small step.

The full set of transformations for Boolean expressions is given in (Loo06). For numerical expressions, the transcendental functions *sin*, *log*, and e^x are used to construct transformations. These obviate the need for division ($a/b = e^{\log(a) - \log(b)}$), and subtraction ($a - b = a + -1 * b$). For lists, transformations are based on insertion of new leaves (e.g. to append function calls), and “deepening” of the normal form by insertion of subclauses; see (Loo06) for details. For tuples, we take the union of the transformations of all the subtypes. For other mixed-type expressions the union of the non-neutral transformations for all types must be considered as well. For enum types the only transformation is replacing one symbol with another. For function types, the transformations are based on function composition. For action result types, actions are inserted/removed/altered, akin to the treatment of Boolean literals for the Boolean type.

We propose an additional set of non-neutral transformations based on the marvelous *fold* function:

$$\begin{aligned} fold(f, v, l) = \\ ifThenElse(empty(l), v, \\ f(first(l), fold(f, v, rest(l)))) . \end{aligned}$$

With *fold* we can express a wide variety of iterative con-

⁷Exact calculations are given by Olsson (Ols95).

structs, with guaranteed termination and a bias towards low computational complexity. In fact, *fold* allows us to represent exactly the primitive recursive functions (Hut99).

Even considering only this reduced space of possible transformations, in many cases there are still too many possible programs “nearby” some target to effectively consider all of them. For example, many probabilistic model-building algorithms, such as learning the structure of a Bayesian network from data, can require time cubic in the number of variables (in this context each independent non-neutral transformation can correspond to a variable). Especially as the size of the programs we wish to learn grows, and as the number of typologically matching functions increases, there will be simply too many variables to consider each one intensively, let alone apply a cubic-time algorithm.

To alleviate this scaling difficulty, we propose three techniques. The first is to consider each potential variable (i.e. independent non-neutral transformation) to heuristically determine its usefulness in expressing constructive semantic variation. For example, a Boolean transformation that collapses the overall expression into a tautology is assumed to be useless.⁸ The second is heuristic coupling rules that allow us to calculate, for a pair of transformations, the expected utility of applying them in conjunction. Finally, while *fold* is powerful, it may need to be augmented by other methods in order to provide tractable representation of complex programs that would normally be written using numerous variables with diverse scopes. One approach that we have explored involves application of Sinot’s ideas about *director strings as combinators* (SMI03). In this approach, special program tree nodes are labeled with director strings, and special algebraic operators interrelate these strings. One then achieves the representational efficiency of local variables with diverse scopes, without needing to do any actual variable management. Reductions and (non-)neutral transformation rules related to broadening and reducing variable scope may then be defined using the director string algebra.

Conclusions

In this paper, we have articulated general conceptual requirements that should be fulfilled by a program representation scheme if it is to be considered tractable, either generically or with respect to particular probabilistic assumptions about the environments and tasks on which programs will be evaluated. With the intention of addressing these requirements, the system of normal forms begun in (Loo06) has been extended to encompass a full programming language. An extended taxonomy of programmatic transformations has been proposed to aid in learning and reasoning about programs.

In the future, we will experimentally validate that these normal forms and heuristic transformations *do* in fact increase the syntactic-semantic correlation in program spaces, as has been shown so far only in the Boolean case. We would also like to explore the extent to which even stronger correlation, and additional tractability properties, can be observed when realistic probabilistic constraints on “natural”

⁸This is heuristic because such a transformation might be useful together with other transformations.

environments and task spaces are imposed. Finally, we intend to incorporate these normal forms and transformations into a program evolution system, such as meta-optimizing semantic evolutionary search (Loo07a), and apply them as constraints on probabilistic inference on programs.

References

- E. B. Baum. *What is Thought?* MIT Press, 2004.
- E. B. Baum. A working hypothesis for general intelligence. In *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, 2006.
- S. Gustafson, E. K. Burke, and G. Kendall. Sampling of unique structures and behaviours in genetic programming. In *European Conference on Genetic Programming*, 2004.
- G. Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 1999.
- M. Hutter. Universal algorithmic intelligence: A mathematical top-down approach. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*. Springer-Verlag, 2005.
- L. Levin. Randomness and nondeterminism. In *The International Congress of Mathematicians*, 1994.
- M. Looks, B. Goertzel, and C. Pennachin. Novamente: An integrative architecture for artificial general intelligence. In *AAAI Fall Symposium Series*, 2004.
- M. Looks. *Competent Program Evolution*. PhD thesis, Washington University in St. Louis, 2006.
- M. Looks. Meta-optimizing semantic evolutionary search. In *Genetic and evolutionary computation conference*, 2007.
- M. Looks. On the behavioral diversity of random programs. In *Genetic and evolutionary computation conference*, 2007.
- M. Looks. Scalable estimation-of-distribution program evolution. In *Genetic and evolutionary computation conference*, 2007.
- W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- J. R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 1995.
- F. R. Sinot, Fernández M., and Mackie I. Efficient reductions with director strings. In *Rewriting Techniques and Applications*, 2003.
- R. Solomonoff. A formal theory of inductive inference. *Information and Control*, 1964.
- L. Spector. Simultaneous evolution of programs and their control structures. In *Advances in Genetic Programming 2*. MIT Press, 1996.
- M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 2005.
- P. Wang. *Rigid Flexibility: The Logic of Intelligence*. Springer, 2006.

Consciousness in Human and Machine: A Theory and Some Falsifiable Predictions

Richard P. W. Loosemore

Surfing Samurai Robots, Inc.
1600 McAllister Road, Genoa NY 13071 USA
rloosemore@susaro.com

Abstract

To solve the hard problem of consciousness we first note that all cognitive systems of sufficient power must get into difficulty when trying to analyze consciousness concepts, because the mechanism that does the analysis will bottom out in such a way that the system declares these concepts to be both real and ineffable. Rather than use this observation to dismiss consciousness as an artifact, we propose a unifying interpretation that allows consciousness to be regarded as explicable at a meta level, while at the same time being mysterious and inexplicable on its own terms. It is further suggested that science must concede that there are some aspects of the world that deserve to be called ‘real’, but which are beyond explanation. The main conclusion is that thinking machines of the future will, inevitably, have just the same subjective consciousness that we do. Some testable predictions can be derived from this theory.

Introduction

The idea that an artificial general intelligence might soon be built raises urgent questions about whether AGIs would (a) be conscious, (b) feel emotions, and (c) have dangerous motivations. Given the strength of public feeling on these matters—for example, the widespread belief that AGIs would be dangerous because as self-aware beings they would inevitably rebel against their lack of freedom—it is incumbent upon the AGI community to resolve these questions as soon as possible. Philosophers may have the luxury of a relaxed debate, but with some people demanding reassurance about the safety of AGI, we do not.

Questions about consciousness, emotion and motivation may be separate issues, but in the public mind they are often conflated, so in this paper I propose to make a start by addressing the first of these. I will argue that if we look carefully at how intelligent systems understand the world we can explain consciousness in a comprehensive manner. This is the first part of a research program aimed at providing a technical foundation for discussions of consciousness, emotion and friendliness in AGI systems.

Copyright © 2008, The Second Conference on Artificial General Intelligence (agi-09.org). All rights reserved.

The Hard Problem of Consciousness

One of the most notorious difficulties with understanding consciousness is the widespread confusion about what the term “consciousness” is supposed to refer to. Chalmers (1996) clarified this somewhat by pointing out that the confusion can be split into two components. First, the word “consciousness” has multiple meanings, so people use it at different times to mean (1) “awakeness,” (2) the ability to have intentions, (3) the thing that makes a philosophical zombie different from a human, and so on.

The second point of confusion is more interesting. Chalmers pointed out that one of these multiple meanings (roughly speaking, the one that is the zombie-human differentiator) is where all of the real philosophical difficulty resides, and he labeled this the “hard problem” of consciousness. Other questions—for example, about the neural facts that distinguish waking from sleeping—may be interesting in their own right, but they do not involve deep philosophical issues and should not be confused with the hard problem.

Defining the Hard Problem. The hard problem is all about the first-person, subjective experience of a creature that is conscious, and about the fact that no matter how good our objective scientific knowledge of the world might become, there seems to be no way to account for that internal subjective experience. Included within the scope of these subjective experiences are questions about “qualia” (the subjective quality of color sensations, pains and the like) and the concept of “self” (that indefinable feeling that we are each a non-physical agent that looks out at the world). Most importantly, the hard problem revolves around the conviction that there could conceivably be such a thing as a “philosophical zombie,” which is defined to be a creature that is identical to a human, but which lacks any subjective phenomenology. If zombies are conceivable, we have to account for the thing that they lack, and the problem of accounting for that thing is the hard problem.

Many philosophers and most lay people would say that these subjective aspects of consciousness are so far removed from normal scientific explanation that if anyone proposed an objective explanation for the hard problem of consciousness they would be missing the point, because

such an explanation would have to start with a bridge between the ideas of *objective* and *subjective*, and since no consensus idea has ever been proposed that might act as such a bridge, no explanation is even on the horizon.

We can summarize the current situation in the philosophical analysis of the hard problem by framing it in terms of the following impasse:

Skeptic: Tell me in objective terms what exactly is meant by terms such as “consciousness” and “qualia,” and we might begin to build an explanation for them. Unless you can say exactly what you mean by these things, you are not saying anything.

The Reply: Unfortunately, the thing we are talking about seems to be intrinsically beyond the reach of objective definition, while at the same time being just as deserving of explanation as anything else in the universe. This lack of objective definition should not be taken as grounds for dismissing the problem—rather, this lack of objective definition IS the problem.

A Preview of the Strategy

The line of attack in this paper has two parts. First, we pick up the idea that the hard problem is about ideas that cannot be clearly defined. Why are they indefinable? Why are we nevertheless compelled to explain them? After suggesting a way to understand how something could be so unusual as to drive philosophers into this paradoxical mixed state, we then go to a second phase of the argument, in which we ask about the “reality” of things that test the limits of what minds can know. At the end of part 1 we seem to be heading in a direction that the skeptic would favor (eliminating the explanandum as an epiphenomenon), but then part 2 makes an unusual turn into a new compromise, neither dualist nor physicalist, which resolves the problem of consciousness in an unorthodox way.

Part 1: The Nature of Explanation

The various facets of consciousness have one thing in common: they involve some form of introspection, because we “look inside” at our subjective experience of the world (qualia, sense of self, and so on) and ask what these experiences amount to. In order to analyze the nature of these introspection we need to take one step back and ask what happens when we think about any concept, not just those that involve subjective experience.

Talking About Analysis Mechanisms

In any sufficiently complete AGI system there has to be a powerful mechanism that lets the system analyze its own concepts. The system has to be able to explicitly think about what it knows, and deconstruct that knowledge in many ways. The scope of this *analysis mechanism* must be extremely broad, and the knowledge that lies within its scope must be couched at an appropriate level. (So: merely giving the AGI access to its own source code would not count as an analysis mechanism).

AGI systems will surely have this analysis mechanism at some point in the future, because it is a crucial part of the “general” in “artificial general intelligence,” but since there is currently no consensus about how to do this, we need to come up with a language that allows us to talk about the kind of things that such a mechanism might get up to. For that reason, I am going to use a language derived from my own approach to AGI—what I have called elsewhere a “molecular framework” for cognition (Loosemore, 2007; Loosemore and Harley, forthcoming).

Nothing depends on the details of this molecular framework, because any other AGI formalism can be translated into this architectural style, but since the molecular framework is arguably more explicit about what the analysis mechanism does, we get the benefit of a concrete picture of its doings. Other AGI formalisms will perhaps take a different approach, but any analysis mechanism must have the crucial features on which this explanation of consciousness depends, so the molecular framework does nothing to compromise the argument.

The Molecular Framework

The following is a generic model of the core processes inside any system that engages in intelligent thought. This is meant as both a description of human cognition and as a way to characterize a wide range of AGI architectures.

The basic units of knowledge, in this framework, are what we loosely refer to as “concepts,” and these can stand for *things* [chair], *processes* [sitting], *relationships* [on], *operators* [describe], and so on. The computational entities that encode concepts are to be found in two places in the system: the **background** (long-term memory, where there is one entity per concept) and the **foreground**, which contains the particular subset of concepts that the system is using in its current thoughts.

The concept-entities in the foreground will be referred to as **atoms**, while those in the background are **elements**.

Many instances of a given concept can be thought about at a given time, so there might be several [chair] atoms in the foreground, but there will only be one [chair] element in the background. From now on, we will almost exclusively be concerned with atoms, and (therefore) with events happening in the foreground.

Theorists differ in their preference for atoms that are either active or passive. A passive approach would have all the important mechanisms on the outside, so that the atoms are mere tokens. An active approach, on the other hand, would have no external mechanisms that manipulate atoms, but instead put all the interesting machinery in and between the atoms. In the present case we will adopt the active, self-organized point of view: the atoms themselves do all of the work of interacting with and operating on one another. This choice makes no difference to the argument, but it gives a clearer picture of some claims about semantics that come later.

Two other ingredients that need to be mentioned in this simplified model are external sensory input and the self-model. We will assume that sensory information originates

at the sensory receptors, is pre-processed in some way, and then arrives at the edge of the foreground, where it causes atoms representing primitive sensory features to become active. Broadly speaking, atoms near the foreground periphery will represent more concrete, low-level concepts, while atoms nearer the “center” of the foreground will be concerned with more high-level, abstract ideas.

The self-model is a structure (a large cluster of atoms) toward the center of the foreground that represents the system itself. This self-model is present in the foreground almost all of the time, because the self is clearly present whenever the system is thinking about anything. At the core of the self-model is a part of the system that has the authority to initiate and control actions.

Finally, note that there are a variety of operators at work in the foreground. The atoms themselves do some of this work, by trying to activate other atoms with which they are consistent (thus, a [cat] atom that is linked to a [crouching-posture] atom will tend to activate an atom representing [pounce]). But there will also be mechanisms that do such things as creation (making a new element to encode a new conjunction of known atoms), elaboration (the assembly of a cluster to represent a situation in more detail), various forms of analogy construction, and so on.

Overall, this model of cognition depicts the process of thought as being a collective effect of the interaction of all these atoms and operators. The foreground resembles a molecular soup in which atoms assemble themselves (with the help of operators) into semi-stable, dynamically changing structures. Hence the term “molecular framework” to describe this way of modeling cognition.

Explanation in General

Atoms can play two distinct roles in the foreground, corresponding to the difference between use and mention. If the system is perceiving a chair in the outside world, a [chair] atom will be part of the representation of that outside situation. But if the system asks itself “What is a chair?”, there will be one [chair] atom that stands as the target of the representation.

When an atom becomes a target, operators will cause this target atom to be elaborated and unpacked in various ways. Call this set of elaboration and unpacking operations an “analysis” event. An analysis event involves various connected concepts being activated and connected to the [chair] atom. If we answer the question by saying that a chair has four legs and is used for sitting on, then this will be because the analysis has gone in such a direction as to cause [sitting] and [function-of] atoms to be activated, as well as a cluster involving [legs], [four] and [part-of].

It is important to be clear that the sense of “explain” that we are examining here is the one that distinguishes itself clearly from the sense that means, simply, “finding all associated concepts.” Analysis, as it is construed here, is not about the fact that [red] tends to be associated with [lips], [blood], [redcurrants] and so on. Humans, and sufficiently powerful AGI systems, clearly have the ability to reduce concepts to more basic terms. This reductionist

type of mechanism is the one that we mean when we talk about the analysis of a target atom.

If this were about narrow AI, rather than AGI, we might stop here and say that the essence of “explanation” was contained in the above description of how the [chair] concept was analyzed into a more detailed representation. However, in an AGI system these core aspects of the analysis process are only part of a much larger constellation of other structures and operators, including representations of: the person who asked the question; that person’s intentions; some background about the different kinds of explanation that are appropriate in different contexts; the protocols for constructing sentences that deliver an answer; the status and reliability of the knowledge in question, and so on.

Analysis is not really a single mechanism, it is an open-ended cluster of flexible, context-dependent mechanisms. More like a poorly demarcated sector of an ecology, than a crisply defined mechanism. However, for the purposes of discussion we will refer to the whole thing as if it were a single “analysis mechanism.”

Explaining Subjective Concepts

In the case of human cognition, what happens when we try to answer a question about our subjective experience of the color red? In this case the analysis mechanism gets into trouble, because the [red] concept is directly attached to an incoming signal line and has no precursors. The [red] concept cannot be unpacked like most other concepts.

The situation here is much worse than simply not knowing the answer. If we are asked to define a word we have never heard of, we can still talk about the letters or phonemes in the word, or specify where in the dictionary we would be able to find the word, and so on. In the case of color qualia, though, the amount of analysis that can be done is precisely zero, so the analysis mechanism returns nothing.

Or does it? I propose that, because of the nature of the representations used in the foreground, there is no way for the analysis mechanism to fail to return some kind of answer, because a non-answer would be the same as representing the color of red as “nothing,” and in that case all colors would be the same. Nothing is not an option, for the same reason that a calculator cannot report that “3 minus 3 is ...” and then not show anything on its display. Structurally, the analysis mechanism must return an atom representing [the subjective essence of the color red], but this atom is extremely unusual because it contains nothing that would allow it to be analyzed. Any further attempt to apply the analysis mechanism to this atom will yield just another atom of the same element.

This bottoming-out of the analysis mechanism causes the cognitive system to eventually report that “There is definitely something that it is like to be experiencing the subjective essence of red, but that thing is ineffable and inexplicable.” This is the only way it can summarize the utterly peculiar circumstance of analyzing [x] and getting [x] back as an answer.

This same “failure” of the analysis mechanism is common to all of the consciousness questions. For qualia, the mechanism dead-ends into the sensory atoms at the edge of the foreground. For the concept of self, there is an innate representation for the self that cannot be analyzed further because its purpose is to represent, literally, itself. On reflection, it seems that all subjective phenomenology is associated with such irreducible atoms.

In each case it is not really a “failure,” in the sense that a mechanism is broken, nor is it a failure that results from the system simply not knowing something. It is an unavoidable consequence of the fact that the cognitive system is powerful enough to recursively answer questions about its own knowledge. According to this view, any intelligent system powerful enough to probe its own intellect in this deep way—any system with an analysis mechanism—would spontaneously say the same things about consciousness that we do.

Finally, it is worth reiterating the point made earlier: this account does not depend on the specifics of the molecular framework. All AGI systems must fail in the same way.

The ‘That Misses The Point’ Objection

The most common philosophical objection to the above argument is that it misses the point, because it explains only the locutions that people produce when talking about consciousness, not the actual experiences they have.

The problem with this objection is that it involves an implicit usage of the very mechanism that is supposed to be causing the trouble. So, when we say “There is something missing from this argument, because when I look at my subjective experiences I see things that are not referenced by the argument”, what we are doing is staying within the system and asking for an explanation of (say) color qualia that is just as good as the explanations we can find for other concepts. But this within-the-system comparison of consciousness with ordinary concepts is precisely the kind of thought process that will invoke the analysis mechanism. And the analysis mechanism will then come back with the verdict that the Loosemore Argument fails to describe the nature of conscious experience, just as other attempts to explain consciousness have failed.

There is no space to analyze all possible objections here, but I would offer instead the following conjecture: when the objections are examined carefully, they will always be found to rely, for their force, on a line of argument that causes the analysis mechanism to run into a dead end. At the same time that the argument cites the analysis mechanism as the chief culprit, then, the objections try to use deploy the analysis mechanism (with flaw intact) to explain why the argument cannot be right.

But this still leaves something of an impasse. The argument does say nothing about the nature of conscious experience, *qua* subjective experience, but it does say why it cannot supply an explanation of subjective experience. Is explaining why we cannot explain something the same as explaining it?

Part 2: The Real Meaning of Meaning

This is not a very satisfactory resolution of the problem, because it sounds as if we are being asked to believe that our most immediate, subjective experience of the world is, in some sense, an artifact produced by the operation of the brain. Of course, the word “artifact” is not quite right here, but then neither are “illusion,” “mirage,” “hallucination,” or any of the other words that denote things that seem to exist, but are actually just a result of our brains doing something odd. By labeling consciousness as a thing that intelligent systems *must* say they experience, (because their concept-analysis mechanisms would not function correctly otherwise), we seem to be putting consciousness on a par with artifacts, illusions and the like. That seems, on the face of it, a bizarre way to treat something that dominates every aspect of our waking lives.

I believe that it is wrong to take the view that the meta-account of consciousness given above leads inexorably to the conclusion that consciousness is some kind of artifact. The best, most satisfying conclusion is that all of the various subjective phenomena associated with consciousness should be considered just as “real” as any other phenomenon in the universe, but that science and philosophy should concede that they have the special status of being unanalyzable. We should declare that such phenomena can be predicted to occur under certain circumstances (namely, when an intelligent system has the kind of powerful “analysis” mechanism described earlier), but that nothing can be said about their nature. In effect, we would be saying that these things are real, but beyond the reach of science.

The remainder of the argument is an attempt to explain and justify this position.

Getting to the Bottom of Semantics

The crucial question, then, is what status we should give to the atoms in a cognitive system that have this peculiar property of making the analysis mechanism return a verdict of “this is real, but nothing can be said about it”.

To answer this question in a convincing way, we need to be more specific about the criteria we are using to justify our beliefs about the realness of different concepts (epistemology), the meaning of concepts (semantics and ontology), and the standards we use to judge the validity of scientific explanations. We cannot simply wave our hands and pick a set of criteria to apply to these things, we need some convincing reasons for choosing as we do.

There seem to be two choices here. One would involve taking an already well-developed theory of semantics or ontology—off the shelf, so to speak—then applying it to the present case. For example, we might choose to go with some form of “possible worlds” semantics, and then note that, according to this perspective, the offending concept-atoms do not take part in any conceivable functions defined over possible worlds, so therefore they can be dismissed as fictions that do not correspond to anything meaningful.

The second choice is to take a detailed look at all the different semantic/ontological frameworks that are available and find out which one is grounded most firmly; which one is secure enough in its foundations to be **the** true theory of meaning/reality/explanation.

The perceptive reader will notice that we are in the process of walking into a trap.

The trap is as follows. If someone were to suggest that the concept of the “meaning” of language can be reduced to some simpler constructs (perhaps, the meanings of basic terms plus rules of compositionality), and that these constructs may then be reduced further (perhaps to functions over possible worlds), and that this reduction could continue until we reach some very basic constructs that are intuitively obvious or self-evident, then that person would be embarking on a doomed endeavor, because any such reductionist plan would end in circularity, or descend into an infinite regress. No matter how far down the reduction went, questions could always be asked about the meanings of the most basic terms (“You are explaining this in terms of ‘possible worlds’? Please tell me the meaning of ‘possible world’?”). The choice then is to either declare an arbitrary limit to the process, or admit that it leads to an infinite regress of questions.

This circularity or question-begging problem applies equally to issues of the meaning of “meaning” and explanations of the concept of “explanation,” and it afflicts anyone who proposes that the universe can be discovered to contain some absolute, objective standards for the “meanings” of things, or the fundamental nature of explanatory force.

Extreme Cognitive Semantics

The only attitude to ontology and semantics that escapes this trap is something that might be called “Extreme Cognitive Semantics”—the idea that there is no absolute, objective standard for the mapping between symbols and things in the world, because this mapping is entirely determined by the purely contingent fact of the design of real cognitive systems (Croft and Cruse, 2004; Smith and Samuelson, 1997). There is no such thing as the pure, objective meaning of the symbols that cognitive systems use, there is just the way that cognitive systems do, in fact use them. Meanings are determined by the ugly, inelegant design of cognitive systems, and that is the end of it.

How does this impact our attempt to decide the status of those atoms that make our analysis mechanisms bottom out? The first conclusion should be that, since the meanings and status of all atoms are governed by the way that cognitive systems actually use them, we should give far less weight to any externally-imposed formalism (like possible-worlds semantics) which says that according to its strictures, subjective concepts point to nothing and are therefore fictitious.

Second—and in much the same vein—we can note that the atoms in question are such an unusual and extreme case, that formalisms like traditional semantics should not even be expected to handle them. This puts the shoe firmly

on the other foot: it is not that these semantic formalisms have no place for the consciousness-concepts and *therefore* the latter are invalid, it is rather that the formalisms are too weak to be used for such extreme cases, and therefore they have no jurisdiction in the matter.

Finally, we can use the Extreme Cognitive Semantics (ECS) point of view to ask what it means to judge various concepts as possessing different degrees of “realness.”

The natural, usage-centered meaning of “real” seems to have two parts. The first involves the precise content of a concept and how it connects to other concepts. So, unicorns are not real because they connect to our other concepts in ways that clearly involve them residing only in stories. The second criterion that we use to judge the realness of a concept is the directness and immediacy of its phenomenology. Tangible, smellable, seeable things that lie close at hand are always more real.

Interestingly, the consciousness atoms score differently on these two measures of realness: they connect poorly to other concepts because we can say almost nothing about them, but on the other hand they are the most immediate, closest, most tangible concepts of all, because they *define* what it means to be “immediate” and “tangible.”

Implications

What to conclude from this analysis? I believe that the second of these two criteria is the one that should dominate, and that the correct explanation for consciousness is that all of its various phenomenological facets deserve to be called as “real” as any other concept we have, because there are no meaningful *objective* standards that we can apply to judge them otherwise. But while they deserve to be called “real” they also have the unique status of being beyond the reach of scientific inquiry. We can talk about the circumstances under which they arise, but we can never analyze their intrinsic nature. Science should admit that these phenomena are, in a profound and specialized sense, mysteries that lie beyond our reach.

This is a unique and unusual compromise between materialist and dualist conceptions of mind. Minds are a consequence of a certain kind of computation; but they also contain some mysteries that can never be explained in a conventional way. We cannot give scientific explanations for subjective phenomena, but we can say exactly why we cannot say anything: so in the end, we can explain it.

Conclusion: Falsifiable Predictions

This theory of consciousness can be used to make some falsifiable predictions. Unfortunately, we are not yet in a position to make tests of these predictions, because doing so would require the kind of nanotechnology that would let us rewire our brains on the fly.

The uniqueness of these predictions lies in the fact that there is a boundary (the edge of the foreground) at which the analysis mechanism gets into trouble. In each case, the

prediction is that these phenomena will occur at exactly that boundary, and nowhere else. Once we understand enough about the way minds are implemented in brains (or in full-scale AGI systems), we will be in a position to test the predictions, because the predicted effects must occur at the boundary if the prediction is to be confirmed.

Prediction 1: Blindsight. Some kinds of brain damage cause the subject to experience ‘blindsight,’ a condition in which they report little or no conscious awareness of certain visual stimuli, while at the same time showing that they can act on the stimuli (Weiskrantz, 1986). The prediction in this case is that some of the visual pathways will be found to lie outside the scope of the analysis mechanism, and that the ones outside will be precisely those that, when spared after damage, allow visual awareness without consciousness.

Prediction 2: New Qualia. If we were to build three sets of new color receptors in the eyes, with sensitivity to three bands in, say, the infrared spectrum, and if we also built enough foreground wiring to supply the system with new concept-atoms triggered by these receptors, this should give rise to three new color qualia. After acclimatizing to the new qualia, we could then swap connections on the old colors and the new IR pathways, at a point that lies *just outside the scope of the analysis mechanism*.

The prediction is that the two sets of color qualia will be swapped in such a way that the new qualia will be associated with the old visible-light colors. This will only occur if the swap happens beyond the analysis mechanism.

If we subsequently remove all traces of the new IR pathways outside the foreground (again, beyond the reach of the analysis mechanism), then the old color qualia will disappear and all that will remain will be the new qualia.

Finally, if we later reintroduce a set of three color receptors and do the whole procedure again, we can bring back the old color qualia, but only if we are careful: the new receptors must trigger the foreground concept-atoms previously used for the visible-light colors. If, on the other hand, we completely remove all trace of the original concept atoms, and instead create a new set, the system will claim not to remember what the original qualia looked like, and will tell you that the new set of colors appear to have brand new qualia.

Prediction 3: Synaesthetic Qualia. Take the system described above and arrange for a cello timbre to excite the old concept-atoms that would have caused red qualia: cello sounds will now cause the system to have a disembodied feeling of redness.

Prediction 4: Mind Melds. Join two minds so that B has access to the visual sensorium of A, using new concept-atoms in B’s head to encode the incoming information from A. B would say that she knew what A’s qualia were like, because she would be experiencing new qualia. If, on the other hand, the sensory stream from A is used to trigger the old concept atoms in B, with no new atoms being constructed inside B’s brain, B would say that A’s qualia were the same as hers.

Conclusion

The simplest explanation for consciousness is that the various phenomena involved have an irreducible duality to them. On the one hand, they are *meta*-explicable, because we can understand that they are the result of a powerful cognitive system using its analysis mechanism to probe concepts that are beyond its reach. On the other hand, these concepts deserve to be treated as the most immediate and real objects in the universe, because they define the very foundation of what it means for something to be real—and as real things, they appear to have ineffable aspects to them. Rather than try to resolve this duality by allowing one interpretation to trump the other, it seems more rational to conclude that both are true at the same time, and that the subjective aspects of experience belong to a new category of their own: they are real but inexplicable, and no further scientific analysis of them will be able to penetrate their essential nature.

According to this analysis, then, any computer designed in such a way that it had the same problems with its analysis mechanism as we humans do (arguably, any fully sentient computer) would experience consciousness. We could never “prove” this statement the way that we prove things about other concepts, but that is part of what it means to say that they have a special status—they are real, but beyond analysis—and the only way to be consistent about our interpretation of these phenomena is to say that, insofar as we can say anything at all about consciousness, we can be sure that the right kind of artificial general intelligence would also experience it.

References

- Chalmers, D. J. 1996. *The Conscious Mind: In Search of a Fundamental Theory*. Oxford: Oxford University Press.
- Croft, W. and Cruse, D. A. 2004. *Cognitive Linguistics*. Cambridge: Cambridge University Press.
- Dowty, D. R., Wall, R. E., & Peters, S. 1981. *Introduction to Montague Semantics*. Dordrecht: D. Reidel.
- Loosemore, R. P. W. (2007). Complex Systems, Artificial Intelligence and Theoretical Psychology. In B. Goertzel & P. Wang (Eds.), *Proceedings of the 2006 AGI Workshop*. IOS Press, Amsterdam.
- Loosemore, R. P. W., & Harley, T.A. (forthcoming). Brains and Minds: On the Usefulness of Localisation Data to Cognitive Psychology. In M. Bunzl & S. J. Hanson (Eds.), *Foundations of Functional Neuroimaging*. Cambridge, MA: MIT Press.
- Smith, L. B., and Samuelson, L. K. 1997. Perceiving and Remembering: Category Stability, Variability, and Development. In K. Lamberts & D. Shanks (Eds.), *Knowledge, Concepts, and Categories*. Cambridge: Cambridge University Press.
- Weiskrantz, L., 1986. *Blindsight: A Case Study and Implications*. Oxford: Oxford University Press.

Hebbian Constraint on the Resolution of the Homunculus Fallacy Leads to a Network that Searches for Hidden Cause-Effect Relationships

András Lőrincz

Department of Information Systems, Eötvös Loránd University, Pázmány Péter sétány 1/C, Budapest, Hungary 1117

Abstract

We elaborate on a potential resolution of the homunculus fallacy that leads to a minimal and simple auto-associative recurrent ‘reconstruction network’ architecture. We insist on Hebbian constraint at each learning step executed in this network. We find that the hidden internal model enables searches for cause-effect relationships in the form of autoregressive models under certain conditions. We discuss the connection between hidden causes and Independent Subspace Analysis. We speculate that conscious experience is the result of competition between various learned hidden models for spatio-temporal reconstruction of ongoing effects of the detected hidden causes.

Introduction

The homunculus fallacy, an enigmatic point of artificial general intelligence, has been formulated by many (see e.g., Searle 1992). It says that representation is meaningless without ‘making sense of it’, so the representation needs an interpreter. Then it continues with the questions: Where is this interpreter? What kind of representation is it using? This line of thoughts leads to an infinite regress. The problem is more than a philosophical issue. We are afraid that any model of declarative memory or a model of structures playing role in the formation of declarative memory could be questioned by the kind of arguments provided by the fallacy.

Our standpoint is that the paradox stems from vaguely described procedure of ‘making sense’. The fallacy arises by saying that the internal representation should make sense. To the best of our knowledge, this formulation of the fallacy has not been questioned except in our previous works (see, Lőrincz et al. (2002), and references therein). We distinguish input and the representation of the input. In our formulation, the ‘input makes sense’, if the representation can produce an (almost) identical copy of it. This is possible, if the network has experienced and properly encoded similar inputs into the representation previously. According to our approach, the internal representation interprets the input by (re)constructing it. This view is very similar to that of MacKay (1956) who emphasized *analysis and synthesis* in human thinking and to Horn’s view (1977), who said that vision is inverse graphics.

In the next section, we build an architecture by starting from an auto-associative network that has input and *hidden representation*. We will insist on Hebbian learning for each transformation, i.e., from input to representation and from representation to *reconstructed input*, of the network. We will have to introduce additional algorithms for proper functioning and will end up with a network that searches for *cause-effect relationships*. During this exercise we remain within the domain of linear approximations. In the discussion we provide an outlook to different extensions of the network, including non-linear networks, and probabilistic sparse spiking networks. The paper ends with conclusions.

Making sense by reconstruction

We start from the assumption that the representation ‘makes sense’ of the input by producing a similar input. Thus, steps of making sense are:

1. input \rightarrow representation
2. representation \rightarrow reconstructed input

If there is a good agreement between the input and the reconstructed input then the representation is appropriate and the input ‘makes sense’. Observe that in this construct there is no place for another interpreter, unless it also has access to the input. However, there is place for a hierarchy, because the representation can serve as the input of other reconstruction networks that may integrate information from different sources. A linear reconstruction network is shown in Fig. 1. We note that if the model recalls a representation, then it can produce a reconstructed input in the absence of any real input.

First, we shall deal with static inputs. Then we consider inputs that may change in time.

The Case of Static Inputs

We start from the constraints on the representation to reconstructed input transformation. The case depicted in Fig. 1 corresponds to Points 1 and 2 as described above. However, it requires a slight modification, because we

need to compare the input and the reconstructed input. This modification is shown in Fig. 2.

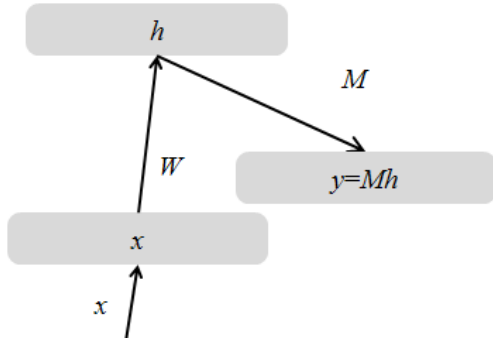


Figure 1. $x \in R^n$: We say that input layer has n neurons. The activity of the i^{th} neuron is x_i . $h \in R^n$: there are n neurons in the hidden representation layer. $W \in R^{n \times n}$: input-to-representation, or bottom-up (BU) transformation. W_{ij} is the ij^{th} element of matrix W : ‘synapse’ or weight from neuron j to neuron i . $y \in R^n$: there are n neurons in the reconstructed input layer. $M \in R^{n \times n}$: top-down (TD) transformation.

Input $x \in R^n$ is compared with the reconstructed input $y \in R^n$ and produces the reconstruction error $e \in R^n$. Then, reconstruction error can be used to correct the representation. It is processed by bottom-up (BU) matrix $W \in R^{n \times n}$ and updates the representation $h \in R^n$. Representation is processed by top-down (TD) matrix $M \in R^{n \times n}$ to produce the reconstructed input. The relaxation dynamics is:

$$h(t + \Delta t) = I h(t) + W(x(t) - Mh(t)) \quad (1)$$

$$h(t) \approx \int_{-\infty}^t \exp(-WM(t - \tau)) Wx(\tau) d\tau \quad (2)$$

Note that update (1) requires a recurrent synapse system that represents the identity matrix I to add $h(t)$ to the update $W(x(t) - Mh(t))$ at time $t + 1$. We will come back to this point later.

Equation (2) is stable if $WM > 0$ (WM is positive definite). Then the architecture solves equation $x = Mh$ for h , so it effectively computes the (pseudo-)inverse, provided that the input is steady. Even for steady input, condition $WM > 0$ should be fulfilled, so we have to train matrix M . Training aims to reduce the reconstruction error and we get cost function $J(M) = \frac{1}{2} \sum_t |x(t) - Mh(t)|^2$ and then the on-line tuning rule:

$$\Delta M \propto \varepsilon(t)h(t)' \quad (3)$$

where apostrophe denotes transpose and $\varepsilon(t) = x(t) - y(t) (= x(t) - Mh(t))$.

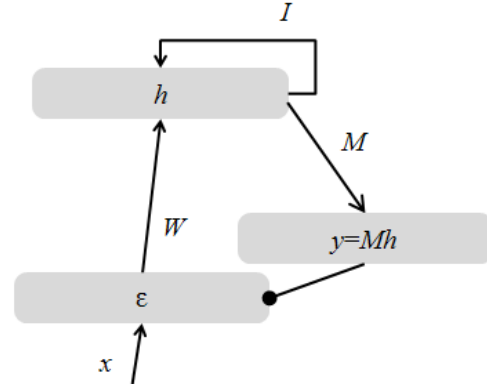


Figure 2. $\varepsilon \in R^n$: the input layer receives inhibitory (negative) feedback from the reconstructed input and becomes a comparator. The input layer holds the reconstruction error ε . Arrow with solid circle: additive inhibition.

We have to modify Fig. 2 to make this learning rule Hebbian (Fig. 3):

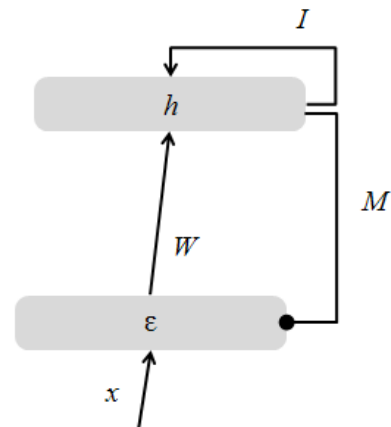


Figure 3. Hebbian training for TD matrix M (Eq. (3)).

Clearly, training of matrix M stops if $M = W^{-1}$, which includes the trivial solution, $M = W = I$. Condition $WM > 0$ is satisfied. The situation is somewhat more delicate if input may change by time. We treat this case below.

The Case of Inputs that Change by Time

If inputs change by time, then we can not reconstruct them, because of two reasons (i) there are delays in the reconstruction loop and (ii) the network may need considerable relaxation time if matrix Q is not properly tuned. We have to include predictive approximations to overcome these obstacles.

First, we introduce a predictive model. Second, we discover problems with Hebbian learning that we overcome by means of the representation. New Hebbian problems will constrain us that we solve by another rearrangement of the network.

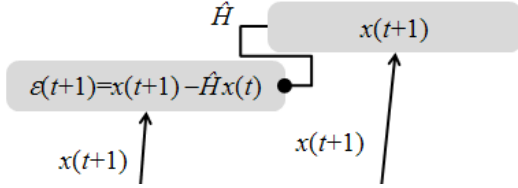


Figure 4. Hebbian learning for predictive matrix \hat{H} .

For the sake of simplicity, we assume that the input is a first order autoregressive model (AR(1)):

$$x(t+1) = Hx(t) + n(t) \quad (4)$$

where $H \in R^{n \times n}$ and its largest eigenvalue is smaller than 1 (for stability) and $n \in R^n$ is the driving noise having normal distribution. Our approximations are \hat{H} for matrix H , and \hat{x} for input estimation, i.e., we estimate $x(t+1)$ as

$$\hat{x}(t+1) = \hat{H}x(t) \quad (5)$$

and the estimation error is

$$\varepsilon(t+1) = x(t+1) - \hat{x}(t+1) \quad (6)$$

and ε is our estimation for noise n . Our cost function is $J(\hat{H}) = \frac{1}{2}|x(t+1) - \hat{H}x(t)|^2$ that leads to the Hebbian training rule:

$$\Delta \hat{H} \propto \varepsilon(t+1)x(t)' \quad (7)$$

The network that can realize Eq. (7) is shown in Fig. 4.

The network in Fig. 4 works as follows. Input $x(t)$ arrives to the two input layers and starts to propagate through matrix \hat{H} . At the next time instant input $x(t+1)$ arrives and the propagated input is subtracted, so we have activities $\varepsilon(t+1) = x(t+1) - \hat{H}x(t)$ on the output end of matrix \hat{H} and the synapses were traversed by $x(t)$, satisfying the constraints of rule (7).

There is a *problem* with the network of Fig. 4: we can not ensure identical inputs at different layers. This problem can be solved if we insert this new network into our previous two-layer architecture (Fig. 3). Having done this, for time varying inputs Eq. (3) assumes the form

$$\Delta M \propto \varepsilon(t+1)h(t)' \quad (8)$$

As we shall see, Eq. (8) enables the learning of a hidden model.

Two layer network with hidden predictive matrix. We add a predictive model (matrix $F \in R^{n \times n}$) to the representation layer; it replaces the identity matrix I as required by non-steady inputs (Fig. 5). Now, we examine how this matrix could be trained.

Equation $M = W^{-1}$ still holds, provided that matrix F – our estimated model – can compensate for the temporal changes. The model at the representation layer is:

$$h(t+1) = Fh(t) + n_h(t+1), \quad (9)$$

where according to our notations, noise n_h should be an estimation of Wn .

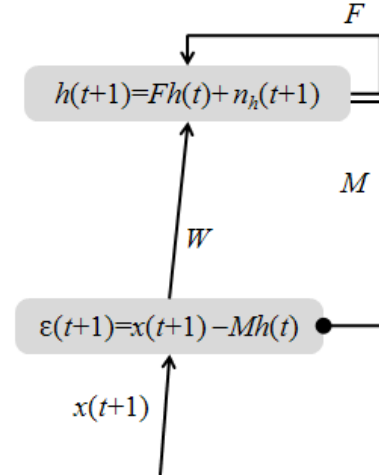


Figure 5: Representation with predictive model.

The question we have is whether we can learn a non-inhibitory predictive matrix F by Hebbian means or not. Although we can learn predictive matrices, see, e.g., Eq. (7), but they would work as comparators.

For model learning, the same trick does not work, we need other means. Our simple structure can be saved if we assume two-phase operation. It is important that two-phase operation fits neuronal networks (Buzsáki, 1989), so we are allowed to use this trick. We assume that $x(t)$ and $\varepsilon(t+1)$ are transferred in Phase I and Phase II respectively by bottom-up matrix W . Under this condition, training of predictive matrix F can be accomplished in Phase II: in

Phase II, the output of matrix is $FWx(t)$, whereas it experiences input $W\varepsilon(t+1)$. The same quantities emerge when considering cost $J(F) = \frac{1}{2}|h(t+1) - Fh(t)|^2$, i.e., the squared error $n_h(t)$ at time t . Note, however, that training of matrix F is supervised and so matrix F can play an additive role.

Discussion

The resolution of the homunculus fallacy has been suggested in our previous works (see, e.g., Lőrincz et al. (2002), and references therein). Here we elaborated that work by more rigorous considerations on Hebbian learning. We were led to a simple network that provides further insights into the ‘making sense’ process:

(1) The network discovers two components: (i) a deterministic process characterized by the predictive matrix and (ii) the driving noise of this deterministic process. One may say that the network discovers the causes (the driving noises) and the effects (the deterministic evolution of the driving noises).

(2) The network builds up an internal model that can run without input. Assume that the network runs for k steps on its own

$$\hat{h}(t+k) = F^k h(t) \quad (10)$$

and then it compares the result with the input k steps later:

$$\varepsilon_k(t+k) = x(t+k) - M\hat{h}(t+k) \quad (11)$$

If the disagreement between the two quantities is small (if $\varepsilon_k(t+k)$ that appears at the input layer is small), then the input process ‘makes sense’ according to what has been learned.

We note for the sake of arguments on consciousness that if the network runs for k time steps, then – according to the dimensional constraints – the network can be increased up to k pieces of parallel running temporal processes, each of them trying to reconstruct the input during the whole k time step history. The pseudo-inverse method is suitable to select the sub-network with the smallest reconstruction error over the k time steps. This sub-network makes the most sense according to history.

(3) The same predictive network can be used for replaying temporal sequences, provided that the starting hidden representation is saved somewhere.

The novelty of this work comes from the examination of Hebbian constraints on reconstruction networks. Neural networks with reconstruction capabilities, however, are not new; there is long history of such networks.

Other works starting from similar thoughts

There are many network models that have similar structure. These networks are typically more complex than the simple/minimal linear autoregressive network that we described here. There are similar networks that aim to model real neuronal architectures. The literature is huge; we can list only some of the most prominent works.

To our best knowledge, the first neocortex related reconstruction network model that suggested approximate pseudo-inverse computation for information processing *between* neocortical areas was published by Kawato et al., (1993). It was called the *forward-inverse model* and modeled the reciprocal connections between visual neocortical areas. The motivation of the model was to connect regularization theories of computational vision (Poggio et al., 1985, Ballard et al., 1983) to neocortical structure and explain how multiple visual cortical areas are integrated to allow coherent scene perception. The computational model of the neocortex was extended by Rao and Ballard (Rao and Ballard, 1997, Rao and Ballard, 1999), who suggested that neocortical sensory processing occurs in a *hierarchy of Kalman filters*. The Kalman filter model extends previous works into the temporal domain.

Non-linear extensions include the so called recurrent neural networks that have non-linear recurrent collaterals at the representation layer. For a review on recurrent neural networks, see Jacobsson (2005). A particular recurrent network model with hidden layer is called Echo State Network (ESN, Jaeger, 2003). ESN – unlike to most models – is non-linear with strictly Hebbian learning. It does not assume two-phase operation. It is made efficient by a huge random recurrent network that forms the internal representation.

Another type of networks with reconstruction flavor belongs to stochastic networks and is called generative model (see, e.g., (Hinton, 2007)). An attempt that connects generative models with two phase operation appeared early (Hinton, 1995), but without details on Hebbian constraints.

The Kalman filter model and the generative network model are the close relatives of the minimal architecture that we described here. They are more sophisticated, but Hebbian learning is so strict as in our minimal model.

Extensions of reconstruction networks

The role of the bottom-up matrix. It is intriguing that Hebbian learning did not provide constraints for the bottom-up matrix. Our proposal, that hidden models discover cause-effect relationships (see point (1) above), leads to the thought that the role of the bottom-up matrix is to help searches for causes. Causes – by definition – are independent, so we have to look for independent sources.

This route is relevant if the noise is not normal, which the typical case for natural sources is. If non-normal sources are hidden and only their mixture is observed, then observed distribution may approximate a normal distribution, because of the d-central limit theorem. Then the following situation is achieved:

1. Deterministic prediction can be subtracted from the observation under the assumption that the driving noise is close to normal distribution
2. Independent sources can be estimated by independent subspace analysis (see, e.g., Cardoso (1998), Hyvarinen and Hoyer (2000)). For a review, see Szabó et al. (2007).
3. The autoregressive processes in the independent subspaces can be learnt by supervisory training that overcomes the problem of non-normal distributions. We note: (a) the least mean square approach that we applied fits the normal distribution, (b) higher order autoregressive processes with moving averages can also be included into the representation (Szabó et al., 2007, Póczos et al., 2007), although it is not yet known how to admit Hebbian constraints.
4. It is unclear if Independent Subspace Analysis can be performed by Hebbian means or not. Efforts to find strictly Hebbian methods for the whole loop including the independent subspace analysis are in progress (Lőrincz et al., 2008a).

The search for cause-effect dependencies can be related to the *Infomax* concept (Barlow, 1961, Linsker, 1988, Atick and Redlich, 1992, Bell and Sejnowski, 1995, Linsker, 1997), because upon removing the temporal process, the search for the *independent* causes is analogous to the Infomax concept (Cardoso, 1997). However, the reasoning is different; here, the aim of independent subspace analysis is to find the causes that drive deterministic processes.

Extensions of this simple architecture to ARMA(p,q) processes (Póczos et al., 2007), non-linear extensions (Jaeger, 2003), extensions with control and reinforcement learning (Szita and Lőrincz, 2004, Szita et al., 2006) are possible. Overcomplete probabilistic sparse spiking extension of the reconstruction architecture has also been suggested (Lőrincz et al., 2008b) and this direction has promises for biologically plausible probabilistic spatio-temporal extensions of the ‘making sense procedure’ under Hebbian constraints.

Outlook to a potential model for consciousness. It has been mentioned before that if the model runs without input for k steps, then the number of models can be multiplied by k , because the pseudo-inverse method can select the best candidate. There is a cost to pay: the best process can not be switched off arbitrarily often, it should be the best

candidate that reconstructs k time steps. Such competition between models to represent the sensory information may explain certain aspects of consciousness, including rivalry situations, when perception is changing steadily, whereas the sensory information is steady.

Conclusions

We have shown that under Hebbian constraints, the resolution of the homunculus fallacy leads to a particular reconstruction network. The network is potentially the simplest in its structure, but not in its functioning: (i) it has a bottom-up, a top-down, and a predictive network, and it is linear, but (ii) it works in two separate phases.

We have shown that the emerging network turns the philosophical infinite regress into a finite loop structure and this finite loop uncovers hidden cause-effect relationships. This is one way to interpret the making sense procedure of the ‘homunculus’. The representation produces the next expected input from time-to-time and computes the difference between the input and this expected reconstructed input. We say that the input makes sense, if this difference is within the range of the expected noise. Also, the network can run by itself as required if inputs are missing.

We have found that constraints arising from the resolution of the fallacy leave the form of the bottom-up network open. However, the reconstruction network uncovers hidden deterministic processes and estimates the driving noise, the hidden causes. Causes are independent ‘by definition’, so the network should work better if the bottom-up transformation is trained on the estimated noise according to Independent Subspace Analysis (ISA), which is provably non-combinatorial under certain circumstances (Póczos et al., 2007, Szabó et al., 2007). The concept of finding causes that drive deterministic processes leads to and takes advantage of a relative of ISA, the so called Infomax concept, which has been developed for modeling sensory information processing in the brain (Barlow 1961, Linsker 1988).

We have speculated that competing models in reconstruction networks may provide a simple explanation for certain features of consciousness. This speculation can be taken further: the model hints that conscious experience may emerge as the result of *distributed* and *self-orchestrated competition* amongst predictive models to reconstruct their common inputs over longer time intervals. This line of thoughts suggests to seek not (only) the *conductor of the orchestra* (see, e.g., Crick and Koch, 2005), but the *distributed selection algorithm* triggered by *unexpected independent causes* as disclosed by reconstruction errors of *competing reconstruction models*.

References

- Atick, J. J. and Redlich, A. N. 1992. What does the retina know about natural scenes? *Neural Comput.* 4:196-210.
- Ballard, D. H., Hinton, G. E., and Sejnowski, T. J. 1983. Parallel visual computation. *Nature*, 306:21-26.
- Barlow, H. 1961. Possible principles underlying the transformations of sensory messages. In: *Sensory Communication*, W. Rosenblith (ed.), pp. 217-234. MIT Press, Cambridge, MA.
- Bell, A. J. and Sejnowski, T. J. 1995. An information-maximization approach to blind separation and blind deconvolution. *Neural Comput.* 7:1129-1159
- Buzsáki, Gy. 1989. A two-stage model of memory trace formation: a role for “noisy” brain states. *Neuroscience* 31: 551–570.
- Cardoso, J.-F. 1997. Infomax and maximum likelihood for source separation, *IEEE Letters on Signal Processing*, 4: 112-114.
- Cardoso, J.-F. 1998. Multidimensional independent component analysis. In *Proc. of Int. Conf. on Acoustics, Speech, and Sign. Proc.* Seattle, WA, USA. 4: 1941–1944.
- Crick, F. C. and Koch, C. 2005. What is the function of the claustrum? *Phil. Trans. R. Soc. B* 360: 1271-1279.
- Hinton, G., E. 2007. To recognize shapes, first learn to generate images. *Prog. Brain. Res.* 165:535-547.
- Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. 1995. The wake-sleep algorithm for self-organizing neural networks. *Science*, 268:1158-1161.
- Horn, B. 1977. Understanding image intensities. *Artificial Intelligence* 8: 201–231.
- Hyvarinen, A., Hoyer, P.O. 2000. Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Comput.*, 12: 1705–1720.
- Jacobsson, H., 2005. Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Comput.*, 17:1223-1263.
- Jaeger, H. 2003, Adaptive nonlinear system identification with echo state networks, *Adv. in Neural Information Proc. Systems* 15: 593-600.
- Kawato, M., Hayakawa, H., and Inui, T. 1993. A forward-inverse model of reciprocal connections between visual neocortical areas. *Network*, 4:415-422.
- Linsker, R. 1988. Self-organization in a perceptual network. *IEEE Computer* 21:105-117.
- Linsker, R. 1997. A local learning rule that enables information maximization for arbitrary input distributions. *Neural Comput.* 9:1661-1665.
- Lőrincz, A. Kiszlinger, M., Szirtes, G. 2008a. Model of the hippocampal formation explains the coexistence of grid cells and place cells. <http://arxiv.org/pdf/0804.3176>
- Lőrincz, A., Palotai, Zs., Szirtes, G., 2008b. Spike-based cross-entropy method for reconstruction. *Neurocomputing*, 71: 3635-3639.
- Lőrincz, A., Szatmáry, B., and Szirtes, G. 2002. Mystery of structure and function of sensory processing areas of the neocortex: A resolution. *J. Comp. Neurosci.* 13:187–205.
- MacKay, D., 1956. Towards an information-flow model of human behavior. *British J. of Psychology* 47: 30–43.
- Póczos, B., Szabó, Z., Kiszlinger, M., Lőrincz, A. 2007. Independent Process Analysis Without a Priori Dimensional Information. *Lecture Notes in Comp. Sci.* 4666: 252–259.
- Poggio, T., Torre, V., and Koch, C. 1985. Computational vision and regularization theory. *Nature*, 317:314-319.
- Rao, R. P. N. and Ballard, D. H. 1997. Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural Comput.*, 9:721-763.
- Rao, R. P. N. and Ballard, D. H. 1999. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neurosci.*, 2:79-87.
- Searle, J., 1992 *The Rediscovery of Mind*. Cambridge, MA.: Bradford Books, MIT Press.
- Szabó, Z., Póczos, B., Lőrincz, A. 2007. Undercomplete Blind Subspace Deconvolution. *J. of Machine Learning Research* 8: 1063-1095.
- Szita, I., Gyenes, V., Lőrincz, A. 2006. Reinforcement Learning with Echo State Networks, ICANN 2006, *Lect. Notes in Comp. Sci.* 4131: 830–839.
- Szita, I., Lőrincz, A., 2004. Kalman filter control embedded into the reinforcement learning framework. *Neural Comput.* 16: 491-499.

Everyone’s a Critic: Memory Models and Uses for an Artificial Turing Judge

W. Joseph MacInnes¹, Blair C. Armstrong², Dwayne Pare³,
George S. Cree³ and Steve Joordens³

1. Oculus Info. Inc.
Toronto, Ont. Canada
Joe.macinnes@oculusinfo.com

2. Department of Psychology
Carnegie Mellon University and
The Center for the Neural Basis of Cognition
Pittsburgh, PA, USA

3. Department of Psychology
University of Toronto Scarborough
Toronto, Ont., Canada

Abstract

The Turing test was originally conceived by Alan Turing [20] to determine if a machine had achieved human-level intelligence. Although no longer taken as a comprehensive measure of human intelligence, passing the Turing test remains an interesting challenge as evidenced by the still unclaimed Loebner prize[7], a high profile prize for the first AI to pass a Turing style test. In this paper, we sketch the development of an artificial “Turing judge” capable of critically evaluating the likelihood that a stream of discourse was generated by a human or a computer. The knowledge our judge uses to make the assessment comes from a model of human lexical semantic memory known as latent semantic analysis[9]. We provide empirical evidence that our implemented judge is capable of distinguishing between human and computer generated language from the Loebner Turing test competition with a degree of success similar to human judges.

Keywords

Semantic Memory, General Knowledge, Decision Making, Machine learning, Language, Turing test.

Introduction

Even before the formal birth of Artificial Intelligence (AI), it was believed by some that computers would achieve human-level intelligence within a relatively short time, so it was essential to devise a test to determine exactly when this milestone had been reached. To this end, Alan Turing [20] proposed the Turing test as one means of evaluating the intelligence of an artificial entity. In essence, he proposed that a computer could be deemed intelligent if it could believably mimic human communication. Specifically, he proposed a guessing game, played by a human confederate, an artificial entity, and – central to this paper - a judge. Without knowing their true identities, the judge would converse with both the confederate and the artificial entity. If the judge was unable to systematically identify which of the two was human, the artificial entity would be said to be intelligent.

Although the classic Turing test is no longer seen as an acceptable measure of human intelligence[18][17], it remains an excellent and incredibly difficult test of language mastery. It can also serve as a valid test of agent believability where the standard may only be to mimic human behaviour [15]. Currently, the annual Loebner competition[7] the most renowned forum for attempts at passing the Turing test, has set a more modest threshold for intelligence than the Turing test: only 30% of the judges need to make incorrect attributions of human intelligence for an attribution of intelligence to be made. Nevertheless, this achievement has yet to be accomplished.

This paper will focus on the oft-forgotten third party of the Turing test: the Turing judge. Since it is the objective of the judge to make the determination of whether the intelligence is human or artificial, the task of implementing an artificial judge is simpler than that of creating an artificial contestant – a test of language recognition and understanding, not generation.

The applications of a language judge are many, both within and outside the context of the Turing test. Within the context of the Turing test, we argue that improved AIs would benefit from a component which evaluates the quality of a generated reply. Our argument to this effect is derived in part from evidence within the cognitive psychology and cognitive science literatures indicating that humans employ some form of critic themselves during sentence comprehension and generation – “a reader tries to digest each piece of text as he encounters it” [22, p. 16]. As one salient example, the manner in which humans process ‘garden path’ sentences[4] whose latter portions do not conform to the interpretation typically expected by the former portion (e.g., the cotton clothing is made of is grown in the South) suggests that we evaluate likely sentence meaning continuously as we read a sentence.

Outside the context of the Turing test, multiple alternative applications abound: evaluation of the quality of student essays[10][19] identification of human versus computer generated on-line forum posts, e-mails, and other forms of web traffic, and the development of security software designed to segregate typical human computer interactions versus automated intrusion attempts.

We have undertaken a principled approach to the development of the first generation of our Turing judge. Our approach draws its inspiration from the early development of artificial intelligence (e.g., Newell & Simon, 1956), which is currently embodied to some extent within the interdisciplinary realm of cognitive science: we aim to advance AI in part through our understanding of human intelligence. Further discussion of this issue awaits later in the paper, but this fact is worthy of emphasis for two reasons: first, it highlights the benefits of a multidisciplinary approach to tackling general AI issues. Second, we wish to explicitly acknowledge that although the “human computer” has been honed over millions of years of evolution, it is clearly lacking in many regards. Future collaborative efforts integrating more non-human approaches in the development of improved Turing judges would therefore be most welcome.

The Turing Judge

The fundamental goal of the Turing judge is to ascertain whether a sentence or passage of text was generated by a human or not. The passage could be evaluated on multiple dimensions: grammaticality (e.g., he throws the ball vs. he throw the ball), meaningfulness of content (e.g., colorless green ideas sleep furiously [2]), relatedness of content to previously discussed content, and so on. Vast literatures and many complex issues surround each of these topics. In developing our first model, we have focused our efforts on two of these issues: assessing the meaningfulness and relatedness of semantic content. These issues in particular seem to be the most fundamentally challenging and relevant to AIs currently being developed to pass the Turing test, as a common strategy in recent years been to simply select a pre-programmed response to a given question from amongst a database of sentences recorded from humans [23].

For the judge to appropriately evaluate the passage of text, it must be supplied with some knowledge of human discourse. To address this issue, we turned to the literature examining the derivation of lexical semantic knowledge (i.e., the derivation of a word’s meaning) from how words co-occur within large samples of natural language (corpora of written text). Numerous computational models have

been developed aimed at extracting different components of structure from within text, and these models have shown considerable success at accounting for a wide variety of comprehension phenomena. Examples include: assessing the correctness of word order in a section of text [24] and comprehending metaphors [25] among others.

When selecting a particular word co-occurrence model to employ in our judge, two main forces came into play. The first was a model’s performance on conversational tasks similar to those a Turing judge might encounter, and the second was the degree to which the model tends to perform well across a wide variety of tasks. Space constraints prevent a detailed discussion of these issues here, but they are expounded in [3]. It suffices to say that consideration of these issues led us to select the Latent Semantic Analysis (LSA [8]) model for use in our Turing judge. It chronologically predates most other models and has been tested in the most diverse set of tasks. It has performed well in most tasks and has been adopted as the de facto benchmark model when comparing the performance of newer models. LSA also has the tangential benefit of being debatably the most well known and easy-to-implement of these models, which should facilitate both the comprehension of the present work, and the execution of future investigations.

Overview of LSA

LSA [8] is a corpus-based statistical method for generating representations that capture aspects of word meaning based on the contexts in which words co-occur. In LSA, the text corpus is first converted into a word x passage matrix, where the passages can be any unit of text (e.g., sentence, paragraph, essay). The elements of the matrix are the frequencies of each target word in each passage (see Figure 1). The element values are typically re-weighted, following a specific mathematical transformation (e.g., log transform) to compensate for disproportionate contributions from high-frequency words. The entire matrix is then submitted to singular value decomposition (SVD), the purpose of which is to abstract a lower dimensional (e.g., 300 dimensions) meaning space in which each word is represented as a vector in this compressed space. In addition to computational efficiency, this smaller matrix tends to better emphasize the similarities amongst words. Following the generation of this compressed matrix, representations of existing or new passages can be generated as the average vectors of the words the passage contains.

Methods

Our implemented Turing judge used an LSA memory model to assess the meaningfulness and relatedness of discourse. The discourse used at test was from previous attempts at the Loebner competition, so as to determine whether the model can accurately distinguish human generated and computer generated responses. Our hypothesis was that human judges use (at least in part) a measure of the semantic relatedness of an answer to a question to spot the computers, so a model which has these strengths should perform fairly well.

LSA Training

The first step in building an LSA model is to select the text database from which the word matrix will be built. Selecting appropriate training data presents a challenge as the questions posed in the Turing test are completely open ended and can be about any topic. As with many machine learning algorithms, the quality of the semantic representations generated by the LSA model often comes down to a question of quantity versus quality of training data. Ultimately, Wikipedia was chosen due to the online encyclopaedia's aim of providing a comprehensive knowledgebase of virtually all aspects of human knowledge, and for its similarity to the training corpora typically used to train word co-occurrence models. It was hoped that the large volume of information in the Wikipedia corpus would compensate for the lack of question and answer style dialogue (as is present in the Live Journal website), although we intend to revisit the trade-offs associated with each of these alternatives in the future.

The entire June 2005 version of Wikipedia was used as a training set for our instantiation of LSA. This corpus contained approximately 120 million words stored in approximately 800 000 unique articles. Each article was pre-processed to remove all of its html and Wikipedia mark-up, so as to generate a "what you see is what you get" version of the database from which LSA could learn. These articles were further stripped of all of their non-alphanumeric characters, all words were converted to lowercase, and function words such as 'the' and 'that' were trimmed because their high frequency ("the" occurs about once every ten words in the average English sentence) and low meaning content tend to detract from LSA's performance.

To illustrate the judging process, consider how the judge would evaluate the similarity of the question "The humans built what?" relative to the responses "The humans built the Cylons" and "They built the Galactica and the vipers", in the case where the judge had access to the simplified

LSA memory model outlined in Table 1 (this example matrix forgoes the SVD compression of the matrix for ease

A1. Humans built the Cylons to make their lives easier.									
A2. The Cylons did not like doing work for the humans.									
A3. In a surprise attack, the Cylons destroyed the humans that built them.									
A4. The Cylons were built by humans to do arduous work.									
B1. Some survivors escaped and fled on the Galactica.									
B2. The Galactica protected the survivors using its Viper attack ships.									
B3. The Cylons were no match for a Viper flown by one of the survivors.									
B4. A Viper flown by one of the survivors found Earth and led the Galactica there.									
	A1	A2	A3	A4	B1	B2	B3	B4	
built	1		1	1					
cylons	1	1	1	1			1		
humans	1	1	1						
a			1				1	1	
galactica					1	1		1	
survivors					1	1	1	1	
viper						1	1	1	

Table 1. Simplified LSA representation for the eight sentences listed above. Each column represents a sentence, and each row represents how frequently each word occurred in that sentence. In this example, words which did not occur at least three times across all sentences and the entry for the function word 'the' have been removed from the table. This matrix has also not been subject to singular value decomposition so as to render it more conceptually straightforward to interpret, although this procedure would be applied in the full model. Note that although LSA has no other knowledge about the world, it nevertheless captures the fact that ('humans', 'built', and 'cylons'), and ('galactica', 'survivors', and 'viper') form clusters of meaningfully related knowledge, and that these clusters are largely separated from one another.

of interpretation, but this transformation would be applied in the full version of the model). First, it would combine the vector representations of each of words (i.e., the rows from the matrix) in each sentence to form a vector representing the combined meaning of each of these words. Ignoring words not present in LSA's memory, the question vector v_q would be equal to $(v_{human} + v_{build})$, and the answer vectors v_{a1} and v_{a2} would be equal to $(v_{human} + v_{build} + v_{cylon})$ and $(v_{built} + v_{galactica} + v_{viper})$ respectively. Note that all of the component vectors which make up v_q and v_{a1} point in roughly the same direction in LSA's memory space (the human-building-cylon region), whereas the component vectors in v_{a2} tend to point to a different region of space than the question vector (the survivors-with-vipers-on-galactica region). Consequently, v_q and v_{a1} would have a higher cosine value, and v_{a1} would be considered the better or more "human" answer.

LSA Turing Judge Performance

We aimed to evaluate the similarity between the Judge's questions relative to both the AI and the human answers in

previous Loebner prize conversations. To do so, we first compiled each conversation into question and answer pairs: the judge’s question followed by the answer of the conversational agent. Our artificial judge then queried the LSA model and had it return the high-dimensional memory vector corresponding to each word in each of the questions and answers. The vectors for the words comprising the questions and answers were then separately conflated to derive a separate representation of the question and the answer in the high-dimensional memory space provided by LSA. The cosine similarity of these vectors was then calculated and used as the metric for the “humanness” of the human or AI agent in question.

Our hypothesis was that a human agent, by virtue of their better overall conversation ability, would have higher semantic similarity with the human judge’s question than any of the artificial agents. Furthermore, we hypothesized that our LSA judge would employ a metric similar to

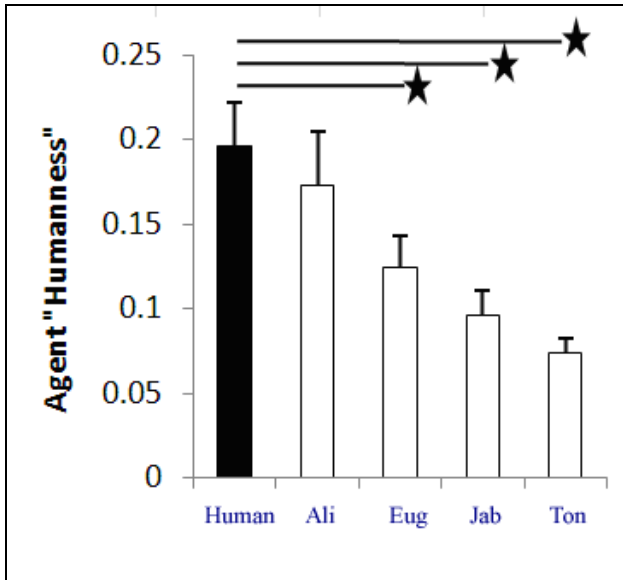


Figure 1. The artificial Turing judge’s “humanness” rating for both the human agent (black bar) and the artificial agents (white bars; Ali → Alice, Eug → Eugene, Jab → Jabberwacky, Ton → Toni). “Humanness” was operationalized as the cosine of the conflated LSA vector similarity for all of the words in the question relative to all of the words in the answer. Error bars are the standard error of the mean. Statistically significant differences between the human agent and the artificial agents are denoted with stars (see text for details). With the exception of ALICE, humans scored significantly higher than the artificial agents.

human judges in assessing whether the agent was a human or not. Consequently, the rank ordering of the different agents provided by our artificial judge should correspond

with those of the human judges. To assess the validity of these hypotheses, we used our judge to evaluate the humanness of the artificial and human agent discourse with the human Turing judge from the 2005 Loebner competition. There were approximately 35 question-answer pairs tested for each of the AIs, and 135 question-answer pairs tested for the humans; humans having more data points because they participated along with each AI in each run of the Turing test.

Results

Based on the process outlined above, our artificial Turing judge generated a ‘humanness’ rating for the human and artificial intelligences and these are reported in Figure 1. As predicted, humans were rated as most “human” by our judge, with each of the artificial agents showing lower performance relative to actual humans. We subjected the humanness ratings to one-way analysis of variance (ANOVA), and pair-wise t-tests¹ of the human agent against all of the artificial agents. A significance threshold of $p = .05$ was used in all analyses. These analyses

Artificial Judge	Human Judge 1	Human Judge 2	Human Judge 3	Human Judge 4
<i>Hum (.20)</i>	Hum (77)	Hum (75)	Hum (79)	<i>Hum (88)</i>
<i>Ali (.17)</i>	Jab (40)	Eug (40)	Eug (45)	<i>Eug (30)</i>
<i>Eug (.12)</i>	Eug (35)	Ton (30)	Jab (20)	<i>Ali (10)</i>
<i>Jab (.10)</i>	Ton (10)	Ali (20)	Ton (15)	<i>Jab (8)</i>
<i>Ton (.07)</i>	Ali (9)	Eug (10)	Ali (5)	<i>Ton (2)</i>

Table 2. Rank orderings and performance scores of the different artificial agents as determined by our artificial Turing judge and the four human judges who evaluated the agents during the Loebner competition. Note both the similarity between our artificial judge’s ratings and those of the fourth human judge (both in italics), and the substantial variability in the rank orderings of the different agents by the different human judges (Hum → Human, Eug → Eugene, Jab → Jabberwacky, Ton → Toni).

indicated significant overall differences between the conditions ($F(4,262) = 2.7$), and the pair-wise t-tests indicated that the human agent was rated as significantly more human than all of the AIs except for ALICE (Human vs. ALICE $t(88.6) = 1.6$; Human vs. EUGENE $t(142.8) =$

¹ Given that large differences in the variability of the humanness ratings for the different agents, equal variance was not assumed when running the t-tests; hence, a separate estimate of each condition’s variance and adjusted degrees of freedom was used to compensate for violating the t-test’s homogeneity of variance assumption.

2.7; Human vs. JabberWacky $t(157.6) = 3.4$; Human vs. Tony $t(157.1) = 4.5$).

To further assess the performance of our artificial Turing judge, we investigated how well our judge's approximation compared to the humanness metric used by actual human judges. To do so, we compared the ordinal rank orderings of the artificial agents in terms of humanness as determined by our artificial Turing judge against the ordinal rank orderings generated by the human judges during the Loebner competition. These data are presented in Table 2. First, on a qualitative level our artificial judge's rank orderings (first column) are quite similar to those of the fourth human judge (the two top rated agents being interchanged across the two judges). Second, there is considerable variability in the rank ordering of the different agents across the different judges.

More formally, we examined the correlations amongst the raw scores provided by each of the judges so as to determine the average consistency across the different judges. These analyses showed that there are significant differences in terms of the average correlation amongst the human judges (mean correlation = .83, SE = .045) and amongst each of the human judges and the artificial judge (mean correlation = .59, SE = .06), $t(8) = 3.34$. Thus, in the details there is clearly room for improvement in our artificial judge, primarily in terms of rating humans as being vastly more "human" than their AI counterparts. Nevertheless, our mean human judge versus artificial judge correlation of .59 is quite substantial (reaching a maximum of .76 amongst the artificial judge and the fourth human judge), and provides at least modest support for the conceptual validity of our approach.

Discussion

This work demonstrates that an artificial Turing judge with access to lexical semantic representations such as those derived by LSA is capable of distinguishing human and computer generated conversation agents with a high degree of accuracy. This test bodes well for semantic detectors as a key component of a more comprehensive artificial Turing judge capable of making more robust and sensitive discriminations. Moreover, the failing of most artificial agents to achieve "human" level semantic similarity amongst the question and responses indicates that enhancing the meaningfulness and relatedness of the answers artificial agents provide to questions they are posed warrants substantial attention by AI researchers interested in the Turing test and related issues.

Despite our model's success, we note several means in which it could be enhanced. For instance, it has yet to be determined whether LSA represents the best knowledge base for the Turing judge to probe when evaluating the humanness of a sentence, nor whether the usage of the cosine is the best metric for assessing the similarity of the content of two passages of text (see [26] for discussion). Furthermore, there are clearly many other dimensions of humanness of a text passage which the current judge ignores (e.g., grammaticality). Framed in a broader context, we view the present work as demonstrating the validity and potential of an artificial Turing judge and the importance semantic knowledge plays in assessing 'humanness'. Nevertheless, there is much which remains unexplored in developing this oft-neglected subcomponent of the Turing test.

Next steps will include a comparison of the current critic trained on Wikipedia with a second critic trained on Live Journal conversations to determine if the conversational style corpus helps in a conversational critic. Live journal offers interesting potential for Turing judges. Since the data is organized by the on-line persona which authored the text, we have an excellent opportunity to train algorithms which also exhibit certain personalities. Each persona contains an accessible description of its author's personality along with a keyword list of user interests. Using these lists, it is quite feasible to train an algorithm with personas interested in a particular topic. For example, we could train algorithms from personas interested in anthropology, computers, or swimming, and in theory, the algorithms may learn more from these areas than others.

Conclusion

Ultimately, for any system to perform the Turing test at a high level it will have to combine information from a variety of sources, and choose among a number of potential responses supported by these sources. Some form of internal judge or critic could be critical in this regard. The current research is the first stage in an interdisciplinary project designed to model human cognition. As we improve our techniques to more human-level computer interaction, we will also need to consider our methods for assessing those techniques. Self-evaluation processes are likely critical to efficient human performance in a wide range of problem solving contexts. The Turing test provides a clearly defined context in which to create and test such self-evaluation processes, and modelling the judge seems to us to be a very reasonable starting point in this regard, and a useful task in its own right.

Acknowledgements

This work was supported by a University of Toronto Scarborough Academic Initiative Fund Grant to WJM, National Sciences and Engineering Research Council (NSERC) Alexander Graham Bell Canada Graduate Scholarship to BCA, and an NSERC Discovery Grant to GSC and WJM.

References

- [1] Burgess, C., & Lund, K. (1997). Parsing constraints and high dimensional semantic space. *Language & Cognitive Processes*, 12, 177-210.
- [2] Chomsky, N. (1957). *Syntactic Structures*. Mouton: The Hague.
- [3] Cree, G. S., & Armstrong, B. (in press). Computational models of semantic memory. In M. Spivey, K. McRae, & M. Joanisse, *The Cambridge Handbook of Psycholinguistics*.
- [4] Ferreira, F., & Henderson, J. M. (1991). Recovery from misanalyses of garden-path sentences. *Journal of Memory and Language*, 25, 725-745
- [5] Foltz, P. W., Kintsch, W., & Landauer, T. K. (1998). The measurement of textual coherence with Latent Semantic Analysis. *Discourse Processes*, 25, 2&3, 285-307.
- [6] Live Journal website. <http://www.livejournal.com>
- [7] Home page for the Loebner prize, a current implementation of the Turing test. <http://www.loebner.net/Prizef/loebner-prize.html>
- [8] Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, 25, 259-284.
- [9] Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2), 211-240.
- [10] Landauer T.K., Laham D. & Foltz P. (2003) Automatic essay
- [11] assessment. *Assessment in Education, Principles, Policy &*
- [12] *Practice* 10, 295-308.
- [13] Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28, 203-208.
- [14] Lund, K., Burgess, C., & Atchley, R. A. (1995). Semantic and associative priming in high dimensional semantic space. *Proceedings of the Cognitive Science Society*. 660-665. Hillsdale, NJ: Erlbaum.
- [15] MacInnes, W.J. (2004) Believability in Multi-Agent Computer Games: Revisiting the Turing Test. *Proceedings of CHI*, 1537.
- [16] Newell, A., & Simon, H. A. (1956). The logic theory machine: A complex information processing system. *IRE Transactions on Information Theory*.
- [17] Nilsson, N. J. (2005). Human-level artificial intelligence? Be serious!. *AI Magazine*, 26(4), 68-75.
- [18] Oppy, G., & Dowe, D. (2003). The Turing Test. In E. Zalta (Ed.) *The Stanford Encyclopedia of Philosophy*. Available online at <http://plato.stanford.edu/entries/turing-test/>.
- [19] Pare, D. E., & Joordens, S. (2008). Peering into large lectures: Examining peer and expert mark agreement using peerScholar, an online peer assessment tool. *The Journal of Computer Assisted Learning*, 24, 526-540.
- [20] Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, 236, P433.
- [21] Walter Kintsch (2001). Predication, *Cognitive Science* 25, 173-202
- [22] Just, M. A., & Carpenter, P. A. (1987). *The Psychology of Reading and Language Comprehension*. Allyn and Bacon, Inc: Newton, MA.
- [23] http://loebner.net/SDJ_Interview.html
- [24] Landauer, T. K., Laham, D., Rehder, B., & Schreiner, M. E. (1997). How well can passage meaning be derived without using word order? A comparison of latent Semantic Analysis and humans. In M. G. Shafto & P. Langley, (Eds.), *Proceedings of the 19th Annual Meeting of the Cognitive Science Society*, pp. 214-417. Mahwah, NJ: Erlbaum.
- [25] Kintsch, W. (2000). Metaphor comprehension: A computational theory. *Psychonomic Bulletin and Review*, 7, 257-266.
- [26] Rohde, D. L., Gonnerman, L. M., & Plaut, D. C. (2007). An improved method for deriving word meaning from lexical co-occurrence. *Cognitive Science*, Submitted.

Unsupervised Segmentation of Audio Speech Using the Voting Experts Algorithm

Matthew Miller

Peter Wong

Alexander Stoytchev

Developmental Robotics Lab

Iowa State University

mamille@iastate.edu, pwwong@iastate.edu, alexs@iastate.edu

Abstract

Human beings have an apparently innate ability to segment continuous audio speech into words, and that ability is present in infants as young as 8 months old. This propensity towards audio segmentation seems to lay the groundwork for language learning. To artificially reproduce this ability would be both practically useful and theoretically enlightening. In this paper we propose an algorithm for the unsupervised segmentation of audio speech, based on the Voting Experts (*VE*) algorithm, which was originally designed to segment sequences of discrete tokens into categorical episodes. We demonstrate that our procedure is capable of inducing breaks with an accuracy substantially greater than chance, and suggest possible avenues of exploration to further increase the segmentation quality.

Introduction

Human beings have an apparently innate ability to segment continuous spoken speech into words, and that ability is present in infants as young as 8 months old (Saffran, Aslin, & Newport 1996). Presumably, the language learning process begins with learning which utterances are tokens of the language and which are not. Several experiments have shown that this segmentation is performed in an unsupervised manner, without requiring any external cues about where the breaks should go (Saffran, Aslin, & Newport 1996)(Saffran *et al.* 1999). Furthermore, Saffran and others have suggested that humans use statistical properties of the audio stream to induce segmentation. This paper proposes a method for the unsupervised segmentation of spoken speech, based on an algorithm designed to segment discrete time series into meaningful episodes. The ability to learn to segment audio speech is useful in and of itself, but also opens up doorways for the exploration of more natural and human-like language learning.

Paul Cohen has suggested an unsupervised algorithm called Voting Experts (*VE*) that uses the information theoretical properties of internal and boundary entropy to segment discrete time series into categorical episodes (Cohen, Adams, & Heeringa 2007). *VE* has previously demonstrated, among other things, the ability to accurately segment plain text that has had the spaces and punctuation removed. In this

paper we extend *VE* to work on audio data. The extension is not a trivial or straightforward one, since *VE* was designed to work on sequences of *discrete* tokens and audio speech is continuous and real valued.

Additionally, it is difficult to evaluate an algorithm that tries to find logical breaks in audio streams. In continuous speech, the exact boundary between phonemes or between words is often indeterminate. Furthermore, there are no available audio datasets with all logical breaks labeled, and given an audio stream it is unclear where all the logical breaks even are. What counts as a logical break? It is difficult to quantify the level of granularity with which human beings break an audio stream, and more so to specify the limits of any rational segmentation. This paper describes a method to address these problems.

Our results show that we are able to segment audio sequences with accuracy significantly better than chance. However, given the limitations already described, we are still not at a point to speak to the objective quality of the segmentation.

Related Work

Our work is directly inspired by the psychological studies of audio segmentation in human beings (Saffran, Aslin, & Newport 1996)(Saffran *et al.* 1999)(Saffran *et al.* 1997). These studies show us that the unsupervised segmentation of natural language is possible, and does not require prohibitively long exposure to the audio. However, these studies do little to direct us towards a functioning algorithm capable of such a feat.

Conversely, there are several related algorithms capable of segmenting categorical time series into episodes (Magerman & Marcus 1990)(Kempe 1999)(Hafer & Weiss 1974)(de Marcken 1995)(Creutz 2003)(Brent 1999)(Ando & Lee 2000). But these are typically supervised algorithms, or not specifically suited for segmentation. In fact, many of them have more to do with finding minimum description lengths of sequences than with finding logical segmentations.

Gold and Scassellati have created an algorithm specifically to segment audio speech using the MDL model (Gold & Scassellati 2006). They recorded 30 utterances by a single speaker and used MDL techniques to compress the representation of these utterances. They then labeled each compressed utterance as positive or negative, depending on

whether the original utterance contained a target word, and then trained several classifiers on the labeled data. They used these classifiers to classify utterances based on whether they contained the target word. This technique achieved moderate success, but the dataset was small, and it does not produce word boundaries, which is the goal of this work.

This work makes use of the Voting Experts (*VE*) algorithm. *VE* was designed to do with discrete token sequences exactly what we are trying to do with real audio. That is, given a large time series, specify all of the logical breaks so as to segment the series into categorical episodes. The major contribution of this paper lies in transforming an audio signal so that the *VE* model can be applied to it.

Overview of Voting Experts

The *VE* algorithm is based on the hypothesis that natural breaks in a sequence are usually accompanied by two information theoretic signatures (Cohen, Adams, & Heeringa 2007)(Shannon 1951). These are low *internal entropy* of chunks, and high *boundary entropy* between chunks. A *chunk* can be thought of as a sequence of related tokens. For instance, if we are segmenting text, then the letters can be grouped into chunks that represent the words.

Internal entropy can be understood as the surprise associated with seeing the group of objects together. More specifically, it is the negative log of the probability of those objects being found together. Given a short sequence of tokens taken from a longer time series, the internal entropy of the short sequence is the negative log of the probability of finding that sequence in the longer time series. So the higher the probability of a chunk, the lower its internal entropy.

Boundary entropy is the uncertainty at the boundary of a chunk. Given a sequence of tokens, the boundary entropy is the expected information gain of being told the next token in the time series. This is calculated as $H_I(c) = -\sum_{h=1}^m P(h, c) \log(P(h, c))$ where c is this given sequence of tokens, $P(h, c)$ is the conditional probability of symbol h following c and m is the number of tokens in the alphabet. Well formed chunks are groups of tokens that are found together in many different circumstances, so they are somewhat unrelated to the surrounding elements. This means that, given a subsequence, there is no particular token that is very likely to follow that subsequence.

In order to segment a discrete time series, *VE* preprocesses the time series to build an n-gram trie, which represents all its possible subsequences of length less than or equal to n . It then passes a sliding window of length n over the series. At each window location, two “experts” vote on how they would break the contents of the window. One expert votes to minimize the internal entropy of the induced chunks, and the other votes to maximize the entropy at the break. The experts use the trie to make these calculations. After all the votes have been cast, the sequence is broken at the “peaks” - locations that received more votes than their neighbors. This algorithm can be run in linear time with respect to the length of the sequence, and can be used to segment very long sequences. For further details, see the journal article (Cohen, Adams, & Heeringa 2007).

It is important to emphasize the *VE* model over the actual

implementation of *VE*. The goal of our work is to segment audio speech based on these information theoretic markers, and to evaluate how well they work for this task. In order to do this, we use a particular implementation of Voting Experts, and transform the audio data into a format it can use. This is not necessarily the best way to apply this model to audio segmentation. But it is one way to use this model to segment audio speech.

The model of segmenting based on low internal entropy and high boundary entropy is also closely related to the work in psychology mentioned above (Saffran *et al.* 1999). Specifically, they suggest that humans segment audio streams based on conditional probability. That is, given two phonemes A and B, we conclude that AB is part of a word if the conditional probability of B occurring after A is high. Similarly, we conclude that AB is not part of a word if the conditional probability of B given A is low. The information theoretic markers of *VE* are simply a more sophisticated characterization of exactly this idea. Internal entropy is directly related to the conditional probability inside of words. And boundary entropy is directly related to the conditional probability between words. So we would like to be able to use *VE* to segment audio speech, both to test this hypothesis and to possibly facilitate natural language learning.

Experimental Procedure

Our procedure can be broken down into three steps. 1) Temporally discretize the audio sequence while retaining the relevant information. 2) Tokenize the discrete sequence. 3) Apply *VE* to the tokenized sequence to obtain the logical breaks. These three steps are described in detail below, and illustrated in Figure 2.

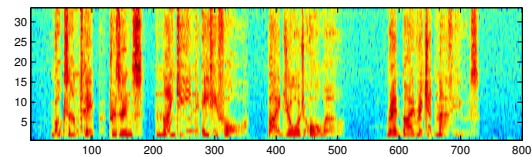


Figure 1: A voiceprint of the first few seconds of one of our audio datasets. The vertical axis represents 33 frequency bins and the horizontal axis represents time. The intensity of each frequency is represented by the color. Each vertical line of pixels then represents a spectrogram calculated over a short Hamming window at a specific point in time.

Step 1

In order to discretize the sequence, we used the discrete Fourier transform in the Sphinx software package to obtain the spectrogram information (Walker *et al.* 2004). We also took advantage of the raised cosine windower and the pre-emphasizer in Sphinx. The audio stream was windowed into 26.6ms wide segments called Hamming windows, taken every 10ms (*i.e.* the windows were overlapping). The windower also applied a transformation on the window to emphasize the central samples and de-emphasize those on the edge. Then the pre-emphasizer normalized the volume across the frequency spectrum. This compensates for the natural attenuation (decrease in intensity) of sound as the frequency is increased.

Finally we used the discrete Fourier Transform to obtain the spectrogram. This is a very standard procedure to obtain the spectrogram information of an audio speech signal, and technical explanation of each of these steps is available in the Sphinx documentation (Walker *et al.* 2004).

We performed the Fourier Transform at 64 points. However, since we are only concerned with the power of the audio signal at each frequency level, and not the phase, then the points are redundant. Only the first 33 contained unique information. This transformation converted a 16kHz mono audio file into a sequence of spectrograms, representing the intensity information in 33 frequency bins, taken every 10ms through the whole file. These spectrograms can be viewed as a voiceprint representing the intensity information over time. Figure 1 shows a voiceprint taken from the beginning of one of the datasets used in our experiments.

Step 2

After discretization the next step is tokenization. Once we obtained the spectrogram of each Hamming window over the entire audio sequence, we converted it to a time series composed of tokens drawn from a relatively small alphabet. In order to do this we trained a Self Organizing Map (SOM) on the spectrogram values (Kohonen 1988).

An SOM can be used as a clustering algorithm for instances in a high dimensional feature space. During training, instances are presented to a 2D layer of nodes. Each node has a location in the input space, and the node closest to the given instance “wins.” The winning node and its neighbors are moved slightly closer to the training instance in the input space. This process is repeated for some number of inputs. Once training is complete the nodes in the layer should be organized topologically to represent the instances presented in training. Instances can then be classified or clustered based on the map. Given a new instance, we can calculate the closest node in the map layer, and the instance can be associated with that node. This way we can group all of the instances in a dataset into clusters corresponding to the nodes in the SOM.

However, this approach has its drawbacks. For instance, it requires the specification of a set number of nodes in the network layer before training begins. Layer size selection is not an inconsequential decision. Selecting too many nodes means that similar instances will be mapped to different nodes, and selecting too few means dissimilar instances will be mapped to the same one.

Instead of guessing and checking, we used a Growing Grid self organizing network (Fritzke 1995). The Growing Grid starts with a very small layer of SOM nodes arranged in a rectangular pattern. It trains these nodes on the dataset as usual, and maps the dataset to the nodes based on their trained values. The node whose mapped instances have the highest variance is labeled as the error node. Then a new row or column is inserted into the map between the error node and its most dissimilar neighbor. The new nodes are initialized as the average of the two nodes they separate, and the map is retrained on the entire dataset.

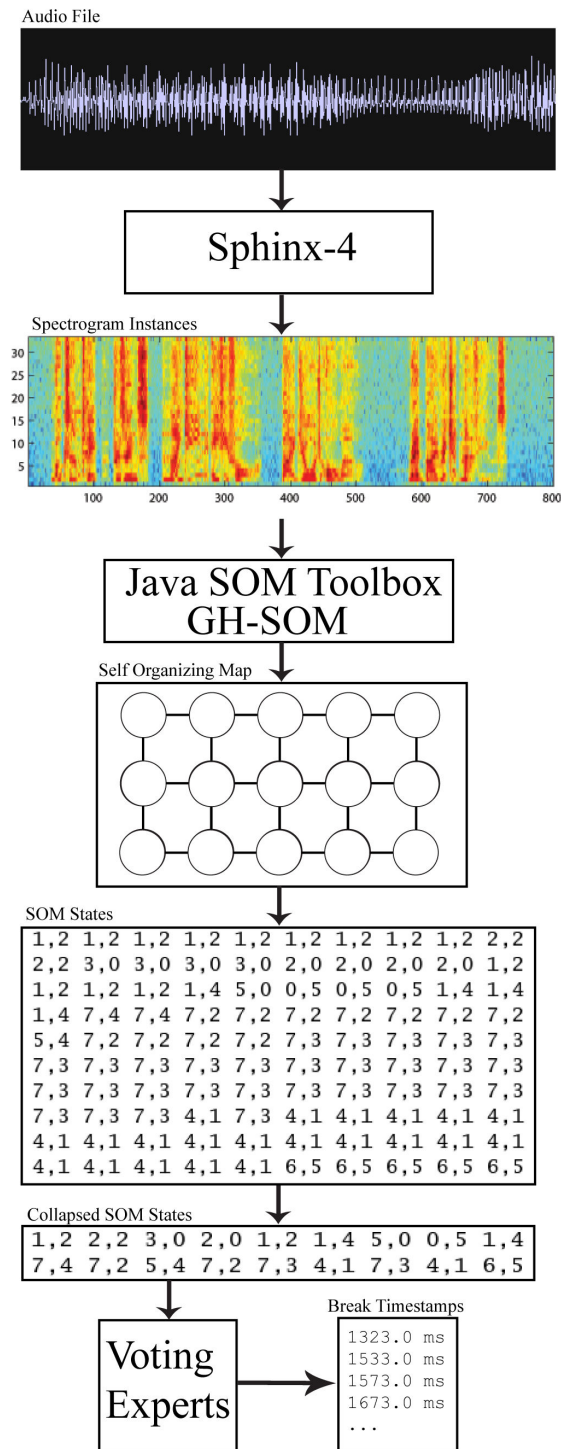


Figure 2: Illustration of the Audio Segmentation Procedure.

The stopping criterion is specified by an error parameter τ . If the variance of the error node is less than τ times the variance of the entire dataset, no new rows or columns are added. This effectively ensures that no single node will account for more than τ of the total error in the dataset. This way the SOM ends up sized appropriately for the particular problem, and the data is spread evenly through the nodes. For our experiments we used a $\tau = 0.01$. We used the implementation of a Growing Hierarchical SOM (GH-SOM) in the Java SOM Toolbox to train our Growing Grid (Dittenbach, Merkl, & Rauber 2000).

After training a Growing Grid SOM on the spectrogram data we used it to classify each instance the dataset. Each node in the SOM was represented by a unique label - its coordinates in the network layer. For instance the node with layer coordinates (1, 2) was represented by the string "1,2". So the clustering produced a sequence of node labels corresponding to each spectrogram instance (see Figure 3). In this way we produced a discrete sequence of tokens representing the audio data.

In the resulting sequence, it was common for several consecutive instances to be mapped to the same node in the SOM. For instance, silence always maps to the same SOM node, so any period of silence in the original audio was represented by several instances of the same node in the discrete sequence. This also happened for similar sounds that were held for any length of time. In order to be time independent, we collapsed these repeated sequences into just one instance of the given node. This effectively denotes a period of silence by a single state, as well as the same sound held for several time steps (see Figure 4). This way our segmentation algorithm only looked at changes between SOM states, and not the duration that the sound stayed the same.

Step 3

In order to segment the tokenized sequence, we ran *VE* on the sequence of SOM states. *VE* placed breaks at locations of low internal entropy and high boundary entropy. Then, after accounting for the collapsed (*i.e.* repeated) tokens, we produced the time stamps of all of the induced break locations in the audio stream.

Experiments

We performed two experiments to test this algorithm.

Experiment 1 First we used text-to-speech software to generate spoken audio. We modeled our dataset on the audio used for the study of speech segmentation in 8-month-old infants. In that study the infants were played artificially generated speech consisting of four made up words "golabu," "tupiro," "bidaku" and "padoti," in random order. We composed a randomly ordered list of 900 instances of the made up words, and then generated audio speech using the built in text-to-speech synthesizer on a Macintosh laptop. We chose their most natural sounding voice, called "Alex," set on medium speed. This resulted in approximately 10 minutes of speech.

We then ran the segmentation process described above on the audio to obtain the induced breaks. That is, we used Sphinx to obtain the spectrogram information for each Hamming window, trained a Growing Grid SOM on the spectro-

gram data, and then used the SOM to convert the spectrogram data into a sequence of SOM node labels. Figure 3 shows the first 100 labels in a sample sequence. It is evident that most nodes are repeated several times in a row, and replacing the repeated nodes with a single instance produces the sequence in Figure 4. The *VE* algorithm was then run on this collapsed sequence, each coordinate pair being used as a fundamental token by the Voting Experts algorithm. *VE* induced breaks between the coordinate pairs, and by re-expanding the sequence to its original length, we calculated the timestamps of the induced breaks. This entire procedure is visualized in Figure 2.

```
1,2 1,2 1,2 1,2 1,2 1,2 1,2 1,2 1,2 2,2
2,2 3,0 3,0 3,0 3,0 2,0 2,0 2,0 2,0 1,2
1,2 1,2 1,2 1,4 5,0 0,5 0,5 0,5 1,4 1,4
1,4 7,4 7,4 7,2 7,2 7,2 7,2 7,2 7,2 7,2
5,4 7,2 7,2 7,2 7,2 7,3 7,3 7,3 7,3 7,3
7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3
7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3 7,3
7,3 7,3 7,3 4,1 7,3 4,1 4,1 4,1 4,1 4,1
4,1 4,1 4,1 4,1 4,1 4,1 4,1 4,1 4,1 4,1
4,1 4,1 4,1 4,1 4,1 6,5 6,5 6,5 6,5 6,5
```

Figure 3: The coordinates of the SOM nodes corresponding to the first 100 spectrogram instances from a sample artificially generated baby talk audio dataset.

```
1,2 2,2 3,0 2,0 1,2 1,4 5,0 0,5 1,4
7,4 7,2 5,4 7,2 7,3 4,1 7,3 4,1 6,5
```

Figure 4: The coordinates of the SOM nodes corresponding to the first 100 spectrogram instances from a sample artificially generated baby talk audio dataset with the repeated states removed.

Experiment 2 We also performed this exact same procedure on the audio from the first of 9 CDs taken from an audio recording of George Orwell's novel "1984." The audio file was roughly 40 minutes in length. The reason we chose this particular audio book is that the text from 1984 was used in the original segmentation experiments with *VE* (Cohen, Adams, & Heeringa 2007). This experiment was performed to evaluate the procedure's effectiveness on language spoken by a person, as compared to artificially generated language.

Evaluation Methodology

Evaluating the output of the algorithm proved to be very difficult. In fact, one of the major contributions of this paper is the methodology described here for evaluating the algorithm. In the first experiment, the audio was generated artificially. It would seem simple to determine the duration of each phoneme and word, and compare those time stamps with the induced breaks, however there are two separate problems with this approach.

First of all, the speech generation software does not generate regularly spaced phonemes. It actually constitutes the sequence using diphones, so that the sound and duration of one phoneme is affected by those around it. Furthermore, the sound generally fades between one phoneme and the next, resulting in an ambiguous break location.

Secondly, there exists more than one logical break location between certain phonemes. In particular, there are silent spaces between some words and after certain phonemes. It is acceptable to place a break anywhere in that silence, or even one break at the beginning and one at the end. In other words, there is much more leeway in proper audio segmentation than one might expect.

These problems are, of course, exacerbated in the case of experiment 2, which uses continuous natural speech instead of the regular artificially generated audio from experiment 1. When evaluating the audio from experiment 1, there are only a limited number of phonemes in use, and a limited number of ways they are combined. This is not the case in experiment 2. Evaluating the breaks then involves solving a different problem at each suggested break location. No two proposed breaks look alike, and so each one is a unique judgment call.

Given these constraints, we tested our segmentation by using human volunteers to verify the breaks. For each induced break, they checked the audio stream to see whether it was placed correctly. In order for it to count as a correct break, it had to be placed within 13ms of an obvious break location. Such locations include the beginning and ending of words, as well as phoneme boundaries where the audio stream suddenly changes in intensity or frequency. Any break placed in the silence between words or phonemes was also counted as correct. These locations were verified visually using software we wrote to view the waveforms and the breaks. The reason the breaks were given a 13ms window on either side is that Sphinx uses a 26.6ms wide Hamming window to calculate the spectrogram information. The breaks output by the algorithm correspond to the center of that window. We counted an induced break as “correct” if there was a true break anywhere inside that window.

If t is the number of true breaks induced by the algorithm, and n is the total number of breaks it induces, then the accuracy is given by $a = t/n$. This tells us how likely it is that an induced break is correct. However, this doesn’t tell us how well the algorithm is really doing. After all, for a break to be considered “correct” it simply has to fall within 13ms of any artifact in the audio stream that a human would consider a logical breaking point. As it turns out, this is actually pretty likely. To account for this, we compared the performance of our algorithm with the performance of randomly generated breaks.

Over the 10 minutes of audio in experiment 1, our algorithm induced roughly 4000 breaks. Experiment 2 had roughly four times that many. It would have been impossible to manually check every single one of the 20,000 breaks induced in the two experiments. Instead, we chose a subset of the data, and checked over that. There are several reasons to think that “spot checking” the algorithm is a good measure of its overall accuracy. In experiment 1 the audio stream consists of the same four words repeated over and over in random order. So we would expect the segmentation algorithm to perform fairly uniformly everywhere. In experiment 2 this was not the case, but we strove to sample enough points to ensure that the estimated accuracy was reliable.

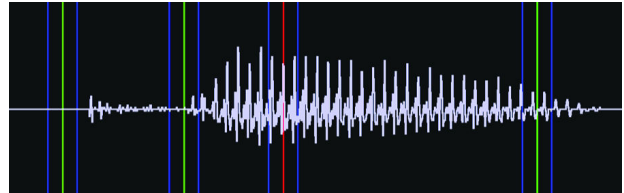


Figure 5: A short sample of the audio from one of our experiments which shows the breaks generated by our algorithm. The “correct” breaks are highlighted in green. The two blue lines around each break show the bounds of the 13ms window.

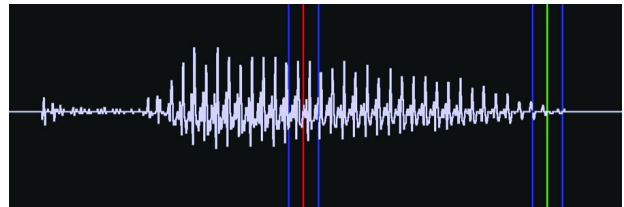


Figure 6: The same audio segment as shown in Figure 5, but with the randomly generated breaks shown instead.

In order to check subsets of the data, 5 sections of 1 minute each were randomly chosen from the audio stream of each experiment. The breaks in these 1 minute segments were recorded. At the same time, for each of these sections we randomly generated the same number of break timestamps over the same 1 minute. We wrote a Java program to load all the breaks at once, visualize the waveform, and allow the volunteers to scroll through the breaks and make their decisions quickly (see Figures 5 and 6).

The volunteers were not told which file contained random breaks, and were instructed to use their best judgment to determine the correctness of each break. They were specifically told to use consistent judging criteria between the two files. This way we compared the performance of our algorithm to an algorithm that randomly generates roughly the same number of breaks. Figure 5 shows an sample section of audio with the induced breaks drawn in. Figure 6 shows the same section of audio with the randomly generated breaks. These sections have already been graded, and the “correct” breaks are marked in green.

Intercoder Reliability

Two volunteers were trained how to use the visualization software, and how to visually identify breaks in the audio stream. One grader was chosen to grade all 20 files, to maintain consistency over the entire dataset.

The second grader was used to test intercoder reliability (Holsti 1969). That is, how consistently human beings will make the same grading decisions regarding the correctness of an induced audio break. The second grader was trained separately from the first, and given 4 pairs of files to evaluate - 2 pairs taken from each experiment. Thus, the second grader evaluated roughly 40% of the same data as the first grader. If t is the total number of decisions to be made by two graders and a is the number of times they agree, then the intercoder reliability is given by $ICR = a/t$. The ICR values for both experiments are summarized in Table 1. The

agreement between our graders is fairly high, considering the subjective nature of most audio break judgements. Typically, an intercoder reliability of 0.8 is considered acceptable.

Table 1: The Intercoder Reliability for experiments 1 and 2

Experiment	Total Breaks	Agreed Breaks	ICR
Exp 1	1542	1346	0.873
Exp 2	1564	1356	0.867

Results

The graded segmentation results are shown in Table 2. For each experiment the breaks induced by our algorithm are shown next to the breaks that were randomly assigned. As you can see, the VE segmentation performed substantially better than chance. In both experiments the accuracy was above 80%, which is considerably good. However, the probability of a random break being placed at a correct location is above 60% in both datasets. It is impossible to know whether a significant portion of our algorithm’s breaks were placed “randomly,” (*i.e.* for bad reasons) and then accidentally marked as correct by the graders.

However, anecdotally, the breaks induced by VE were much more logical than the random ones, even in cases when its breaks were incorrect. They tended to be placed at the beginning and ending of words, and at dramatic shifts in the audio signal. Many times the “incorrect” breaks came at locations that could have been phoneme boundaries, but were impossible to distinguish visually by the graders. An audio evaluation of each break would have taken a substantial amount of time compared to the quick visual grading, and we could not perform that experiment at this time.

Also, the randomly generated breaks got a substantial number “correct” that happened to fall in the silence between words. We instructed the volunteers to count any breaks that fell in silence as correct, so these breaks helped to increase the accuracy of the random segmentation. The VE breaks, however, generally did not exhibit this behavior. They were usually placed either neatly between the words, or at the end of the silence before the next word began. These qualitative observations are not reflected in the difference in accuracy between the VE segmentation and the random one. Which leads us to believe that further evaluation will show a much greater gap between the two methods. Figures 5 and 6 illustrate a typical example of the difference in segmentation.

Table 2: Accuracy Results for Experiments 1 and 2

Experiment	Total Breaks	Correct Breaks	Accuracy
Exp 1 - Algorithm	1922	1584	0.824
Exp 1 - Random	1922	1312	0.683
Exp 2 - Algorithm	1910	1538	0.805
Exp 2 - Random	1910	1220	0.639

Conclusions and Future Work

We have described a technique for transforming spoken audio into a discrete sequence of tokens suitable for segmentation by the Voting Experts algorithm. And we have shown that this technique is clearly capable of inducing logical

breaks in audio speech. This is a very significant result and demonstrates that the unsupervised segmentation of audio speech based on the information theoretic model of VE is possible. We have tested the algorithm on simple “baby talk” inspired by literature on statistical learning in infants (Saffran, Aslin, & Newport 1996). We have also tested it on large audio dataset of spoken English taken from an audio book. This demonstrates its ability to work on real world audio, as well as its tractability when dealing with large datasets. The segmentation, however, is imperfect. There are several possible avenues of future work that might improve the segmentation accuracy of the algorithm.

For example, we decided to use the spectrogram information calculated at discrete time slices as our base instances. We could have used cepstral information, which has been shown more effective in speech recognition tasks. But the spectrogram is more straightforward and applies to audio other than speech. It is possible in future work to use the cepstral coefficients and their derivatives in place of the spectrograms.

It is also possible that the dimension of the Fourier transform might be increased. The SOM might produce better clustering with a higher or lower τ parameter. Or, there might be a better method altogether for finding boundaries that produce low internal entropy of chunks and high boundary entropy between them. There are many possibilities for improvement and future investigation of this procedure. All that can be said right now is that finding such breaks does produce a somewhat logical segmentation of audio speech. It will be interesting to discover whether truly reliable segmentation can be performed this way, and whether these segments can be used as a basis for human-like language learning.

References

- Ando, R., and Lee, L. 2000. Mostly-unsupervised statistical segmentation of Japanese: applications to kanji. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, 241–248.
- Brent, M. R. 1999. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*.
- Cohen, P.; Adams, N.; and Heeringa, B. 2007. Voting experts: An unsupervised algorithm for segmenting sequences. *Journal of Intelligent Data Analysis*.
- Creutz, M. 2003. Unsupervised segmentation of words using prior distributions of morph length and frequency. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, 280–287.
- de Marcken, C. 1995. The unsupervised acquisition of a lexicon from continuous speech. Technical Report AIM-1558.
- Dittenbach, M.; Merkl, D.; and Rauber, A. 2000. The growing hierarchical self-organizing map. *Neural Networks, 2000. IJCNN 2000* 6:15–19 vol.6.
- Fritzke, B. 1995. Growing grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters* 2(5):9–13.
- Gold, K., and Scassellati, B. 2006. Audio speech segmentation without language-specific knowledge. In *Proceedings of the 2nd Annual Conference on Human-Robot Interaction (HRI-07)*.
- Hafer, M. A., and Weiss, S. F. 1974. Word segmentation by letter successor varieties. *Information Storage and Retrieval* 10(11-12):371–385.
- Kempe, A. 1999. Experiments in unsupervised entropy-based corpus segmentation.
- Kohonen, T. 1988. Self-organized formation of topologically correct feature maps. 509–521.
- Magerman, D. M., and Marcus, M. P. 1990. Parsing a natural language using mutual information statistics. In *National Conference on Artificial Intelligence*, 984–989.
- Saffran, J. R.; Aslin, R. N.; and Newport, E. L. 1996. Statistical learning by 8-month-old infants. *Science* 274(5294):1926–1928.
- Saffran, J. R.; Newport, E. L.; Aslin, R. N.; and Tunick, R. A. 1997. Incidental language learning: Listening (and learning) out of the corner of your ear. *Psychological Science*.
- Saffran, J. R.; Johnson, E. K.; Aslin, R. N.; and Newport, E. L. 1999. Statistical learning of tone sequences by human infants and adults. *Cognition*.
- Shannon, C. 1951. Prediction and the entropy of printed English. Technical report, Bell System Technical Journal.
- Walker, W.; Lamere, P.; Kwok, P.; Raj, B.; Ghing, R.; and Gouvea, E. 2004. Sphinx-1: A flexible open source framework for speech recognition. Technical Report TR-2004-139.

Parsing PCFG within a General Probabilistic Inference Framework

Arthi Murugesan

Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy NY 12180

Nicholas L. Cassimatis

Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy NY 12180

Abstract

One of the aims of Artificial General Intelligence (AGI) is to use the same methods to reason over a large number of problems spanning different domains. Therefore, advancing general tools that are used in a number of domains like language, vision and intention reading is a step toward AGI. Probabilistic Context Free Grammar (PCFG) is one such formalism used in many domains. However, many of these problems can be dealt with more effectively if relationships beyond those encoded in PCFGs (category, order and parthood) can be included in inference. One obstacle to using more general inference approaches for PCFG parsing is that these approaches often require all state variables in a domain to be known in advance. However, since some PCFGs license infinite derivations, it is in general impossible to know all state variables before inference. Here, we show how to express PCFGs in a new probabilistic framework that enables inference over unknown objects. This approach enables joint reasoning over both constraints encoded by a PCFG and other constraints relevant to a problem. These constraints can be encoded in a first-order language that in addition to encoding causal conditional probabilities can also represent (potentially cyclic) boolean constraints.

Introduction

An important aspect of general intelligence is that the same method can be applied to various problems spanning different domains. It is believed that several commonalities underlie the various domains of cognition and some of them have been pointed out by the theory of the Cognitive Substrate (Cassimatis, 2006). These include temporal ordering, part hierarchies, generative processes and categories. Probabilistic Context Free Grammars (PCFG) is a formalism that has been widely used to model these phenomena in various domains like vision, RNA folding and Natural Language Processing. Hence improving the coverage of PCFG and integrating PCFGs with a general probabilistic inference framework is a step towards achieving Artificial General Intelligence (AGI).

Probabilistic Context Free Grammars (or Stochastic Context Free Grammars) encode a few types of relations like temporal ordering, category and parthood. These

kinds of relations play an important role in a wide variety of domains, including natural language (Charniak, 2000), the secondary structure of RNA (Sakakibara, Brown, Underwood, Mian & Haussler, 1994), computer vision (Moore & Essa, 2002), plan recognition (Pynadath & Wellman, 2000), intention reading and high-level behavior recognition (Nguyen, Bui, Venkatesh & West, 2003).

Though these relations encoded by PCFG can be used in different domains, many problems require the representation of additional relations. Constraints such as causality can not be expressed within PCFG. In the domain of natural language processing, for example, syntactic regularities are captured by the grammar and improvements are obtained by adding more constraints including lexicalization (Collins, 2003). However, visual cues, social context, individual bias and semantics are all factors affecting language processing (Ferguson & Allen, 2007) that have no straightforward PCFG representation.

The additional relations can be represented in more general frameworks such as Bayesian networks and weighted constraint SAT solvers. These systems, besides modeling PCFG constraints, can also encode a variety of other constraints within the same framework. However, these systems typically require all objects or state variables in a domain to be known in advance and thus are poorly suited for PCFG parsing, which can lead to infinite derivations.

A few probabilistic inference approaches deal with problems that have a large number of grounded constraints by utilizing on demand or lazy grounding of constraints (Domingos et al. 2006). However, these systems nevertheless require that all the possible objects of the domain be declared in advance.

Approach

Here, we describe a new general probabilistic inference framework that allows inference over objects that are not known in advance, but instead are generated as needed. This key feature makes it possible to harness the power of PCFGs and the full flexibility of more general frameworks within a single, integrated system. Our approach to encoding and parsing PCFG in a general

probabilistic inference framework has three features:

Explicitly Encode Constraints Implicit in PCFG

Implicit in PCFG are several constraints. For example, (a) every nonterminal in a derivation must ultimately be manifest as a terminal and (b) every phrase can be immediately dominated by only one other phrase. Explicitly encoding these constraints in a more expressive probabilistic inference framework allows them to be jointly reasoned over with other forms of constraints.

Relational Representation

Even with grammars that license only finite derivations, the number of such derivations can be very large. This translates into inference problems with large numbers of state variables, and the resulting memory demands can render this kind of problem intractable. One way to overcome this is to use relational probabilistic languages, for which there are inference approaches that significantly reduce the memory demands imposed by large numbers of state variables (Domingos, Kok, Poon, Richardson & Singla, 2006).

Licensing the Existence of Unknown Objects

We will use a probabilistic framework, GenProb, that enables reasoning over objects not known prior to inference.

Generative Probabilistic Theory

These three desiderata mentioned are manifest in the Generative Probabilistic theory, GenProb (under review). GenProb is a relational language for expressing probabilistic constraints over unknown objects. This language supports both causal conditional probabilities and (potentially cyclic) boolean constraints. An exact inference algorithm has been defined for GenProb theories (under review) that can be classified as increasing cost models. PCFG problems are increasing cost models and hence exact reasoning over these possibly infinite models is possible.

Syntax of GenProb

GenProb is a language for expressing probabilistic relational theories over unknown objects. The following highly simplified example theory of an airplane radar detector illustrates GenProb.

”Any particular plane has a 1% chance of being within range of a radar station. The radar display generates blips that indicate a strong possibility of a plane being detected and blips that indicate a weak possibility. Strong blips are only caused by planes, whereas weak blips can be caused by noise .01% of the time. Planes being tracked are fitted with collision warning systems that, in the presence of other planes in range, have a 90% chance of sounding an alarm that is transmitted to the radar station.”

The following formulae indicate the priors on a particular plane being in range and on noise:

$\text{True}() \longrightarrow (.01) \text{InRange}(\text{?p}) \wedge \text{Plane}(\text{?p})$

$\text{True}() \longrightarrow (.001) \text{WeakBlip}(\text{?b})$

The causal aspects of the theory are indicated with conditionals:

$\text{InRange}(\text{?p}) \wedge \text{Plane}(\text{?p}) \longrightarrow$

$(.3) \text{StrongBlip}(\text{?b}), (.5) \text{WeakBlip}(\text{?b}),$

$(.2) \text{NoBlip}(\text{?b}), \text{?p}$

$\text{Detect}(\text{?p1}, \text{?p2}) \wedge \text{Plane}(\text{?p1}) \wedge \text{Plane}(\text{?p2}) \wedge \text{InRange} \longrightarrow (.9) \text{TransitAlarm}(\text{?p1}), \text{?p1}, \text{?p2}$

The first conditional indicates that a plane that is in range will have one and only one of the following effects: a strong blip (in 30% of cases), an uncertain blip (50%), and no blip otherwise. The occurrence of ?p after the final comma indicates that a blip licenses the existence of a plane to be inferred. The other variables are implicitly universally quantified. This will be made more precise below. The alarm detection system can be indicated thus:

$\text{TransitAlarm}(\text{?p1}) \implies \text{AlarmSound}()$

Conditionals with numbers are called causal conditionals and those without numbers are called logical conditionals. Logical conditionals are hard constraints.

Since blips occur in the consequents of causal conditionals, they must be the effect of one of these conditionals. In this case, strong blips can only be caused by planes, while weak blips can be caused by planes and by noise. We label the causal interpretation that an effect must be caused by one of its causes (constraint’s antecedents being true) as *mandatory causation*. Such mandatory causation is not implied by logical conditionals. Mandatory causation for literals can be neutralized with a causal conditional whose antecedent is $\text{True}()$, which (see below) is always true.

More formally, a GenProb theory is a set of causal and logical conditionals. Causal conditionals are of the form $C_1 \wedge \dots \wedge C_n \longrightarrow (p_1)E_1, \dots, (p_m)E_m, \dots ?v_i, \dots$, where $0 \leq p_i \leq 1$ and where the p_i sum to 1, each of the C_i are either literals or negations thereof, and the E_i are conjunctions of literals. Each E_i conjunction is called an *effect* of the conditional and each v_i is called a *posited variable*. Non-posited variables are implicitly universally quantified. Literals are predicates with arguments that are terms. Terms that are not variables are called “objects”. Logical conditionals are of the form $A_1 \wedge \dots \wedge A_n \implies B_1 \wedge \dots \wedge B_n$, where each conjunct is either a literal or a negation thereof. Literal a is a grounding of literal b if they are equivalent under an assignment of variables to objects in b and no variables occur in a . Literals and conditionals are grounded if they contain no variables.

Exact Inference Over GenProb

Many of the existing approaches for combining first-order logic and probabilistic graphical models propositionalize relational theories and making inferences over these propositionalized formulae. However, most of these approaches require all objects in the domain to be

known in advance, although many important problems like probabilistic context-free grammars involve objects that are initially unknown and permit infinite derivations.

Theories over potentially unknown objects pose two problems for inference approaches based on propositionalization. First, theories of finite size that express relations over unknown objects often require infinite models. For example, the formula, $Mammal(a) \wedge \forall x(Mammal(x) \rightarrow Mammal(mother(x)))$ (together with formulae stating that a mammal cannot be its own ancestor) require an infinite model because as mother must also have a mother who must also have a mother, ad infinitum. Likewise, some context-free grammars with finite numbers of rules and terminals can generate an infinite number of sentences. Since an algorithm cannot enumerate an infinite model in finite time, we must find a way of finitely characterizing solutions to problems that have infinite models.

A second problem associated with unknown objects is that even if all models of a theory can be finitely characterized, there may nevertheless be infinitely many such models. Complete satisfiability algorithms (e.g., those based on Davis-Putnam-Logemann-Loveland DPLL algorithm) over finite domains are guaranteed to halt because they perform exhaustive search through the space of possible models. Thus, developing model finding algorithms when there are infinitely many possible models poses additional difficulties over standard complete satisfiability algorithms. Exact inference over a subset of GenProb theories has been defined (under review).

The key approach behind the inference mechanism, is to convert GenProb theory to a corresponding weighted satisfiability (SAT) model. However, since GenProb licenses unknown objects, this weighted SAT model must also allow the licensing of unknown objects during inference. Therefore, a version of SAT called the Generative SAT (GenSAT) has been defined. Also an exact inference algorithm, Generative DPLL (GenDPLL), that makes guaranteed inference over GenSAT constraints is defined. GenDPLL is a DPLL-like branch-and-bound algorithm that lazily posits new objects and instantiates clauses involving them. It has been proved that GenDPLL is guaranteed to find finite relevant models of certain classes of GenSAT theories with infinite models, which we call increasing cost models.

Increasing cost models are theories in which the introduction of new constraints can only lead to models of lower cost. PCFG is one such theory, because the introduction of more branches to a parse tree always leads to a less probable solution (or an increased cost model).

Mapping PCFG onto GenProb Language

Jointly reasoning over PCFG and other constraints is enabled by expressing PCFG problems in the GenProb language and using the defined inference mechanisms

of GenProb to reason over these constraints. A PCFG rule is of the form:

$$X \rightarrow (Prob_1)u_{11}u_{12} \dots u_{1m_1} \\ | (Prob_2)u_{21}u_{22} \dots u_{2m_2}$$

$$\dots \\ | (Prob_n)u_{n1}u_{n2} \dots u_{nm_n}$$

where the antecedent X on the LHS of the rule is called a non-terminal. The rule is called a production and is described as the non-terminal X generating the RHS symbols. A symbol that does not generate any further symbols i.e. never occurs on the LHS of a rule is called a terminal.

The rule also captures probability of the non-terminal generation a particular set of symbols like $u_{11}u_{1m_1}$ or $u_{21}u_{2m_2}$ through the numbers $Prob_1$ and $Prob_2$ respectively. The sum of all the probabilities is 1. $\sum_{i=1}^n Prob_i = 1$

A grammar G generates a language L(G) using the PCFG rules in R. The functionality of a parser P for a given string (I) is to determine whether and how this string (I) can be generated from the grammar (G) (i.e., to determine if (I) is a member of L(G)). There are several implicit constraints in the generation of a language. Our aim is to formalize and explicitly encode these constraints in the GenProb language.

The Order Constraint

Probabilistic rules in most language are generally order independent with regard to both the order of the input and the order of the terms in their constraints. However, the language generated by G depends on several ordered components including the order of the list of terminals in the string (I) and the order of right hand side(RHS) components in a PCFG rule.

Ordering of Input

Let the input I, say A1, A2, .. An, be the ordered sequence of input terminals for which the parse has to be determined. The general notion of ordering of events can be broken down into 1. capturing the time of occurrence of an event(both start and end points) and 2. establishing relations between these time of occurrences. The constraints of I (A1, A2 .. An) is captured using the following grounded propositions.

```
Occur(a1), IsA(a1, A1),
StartTime(a1, t1), EndTime(a1, t1),
ImmediatelyBefore(t1, t2) ,
Occur(a2), IsA(a2, A2),
StartTime(a2, t2), EndTime(a2, t2),
ImmediatelyBefore(t2, t3) ,
...
Occur(an-1), IsA(an-1, An-1),
StartTime(an-1, tn-1), EndTime(an-1, tn-1),
ImmediatelyBefore(tn-1, tn),
Occur(an), IsA(an, An),
StartTime(an, tn), EndTime(an, tn)
```

Order of PCFG Rule Arguments

A typical PCFG rule(R) of format $X \rightarrow (Prob)u_1u_2 \dots u_m$ depends on the order of the u symbols. According to the definition of R , u symbols can be both terminals and non-terminals. The same ordering technique used to order the input terminals I , can be used to order RHS components of R . However it is to be noted that this scheme also requires associating non-terminal symbols with the time of their occurrence. Hence the non terminal X on the LHS is also associated with the entire time interval of all the consequents. (We'll also expand on this in the creation of new phrases section)

```

Occur(?xobj)  $\wedge$  IsA(?xobj, X)  $\wedge$ 
StartTime(?xobj, ?t0)  $\wedge$  EndTime(?xobj, ?tn)
 $\rightarrow$  (Prob)
Occur(?u1obj)  $\wedge$  IsA(?u1obj, u1)  $\wedge$ 
StartTime(?u1obj, ?t0)  $\wedge$  EndTime(?u1obj, ?t1)
 $\wedge$  ImmediatelyBefore (?t1, ?t2)  $\wedge$ 
...
Occur(?unobj)  $\wedge$  IsA(?unobj, un)  $\wedge$ 
StartTime(?unobj, ?t(n-1))  $\wedge$ 
EndTime(?unobj, ?tn)

```

Creation of New Objects

The GenProb constraints that capture the PCFG generation rules have unbound objects on both sides as shown in the ordering constraint of R . The GenProb language handles an unbound constraint by creating a new object for the unbound variable in the LHS when the pattern in the RHS is matched completely. The new object is created through the process of skolemization.

Specifically with respect to the rule of the ordering constraint, when the objects of the RHS and their corresponding category, time information match the pattern, the $?xObj$ on the LHS is created. Also the time information which is already bound by the RHS pattern matching, is asserted for the new object.

Unique Dominator

Another implicit constraint of $L(G)$ is the unique parent relationship. Every node can have only one parent creating a strict parse tree and disallowing a multi-tree.

The unique parent relationship is captured in GenProb language by introducing part-hood associations. Every node belongs or is a part of its parent node, and a node cannot be a part of more than one parent.

```

PartOf(?childNode, ?parentNode1)  $\wedge$ 
PartOf(?childNode, ?parentNode2)  $\wedge$ 
NOT Same(?parentNode1, ?parentNode2)
 $\Rightarrow$  FALSE

```

Mandatory Parent

The GenProb language handles two kinds of constraints: causal and logical constraints. Any consequent of a causal constraint is required (according to mandatory causation) to have at least one of its causes to be true. Thus, if $P_1(a) \rightarrow R(a), P_2(a) \rightarrow R(a), \dots, P_n(a) \rightarrow R(a)$ are all the causal conditionals with

$R(a)$ in the consequent, any model where $R(a)$ is true must have at least one of $P_i(a)$ being true. This is captured in GenProb by keeping track of all the causes of grounded propositions and adding a logical constraints called the mandatory causation constraint.

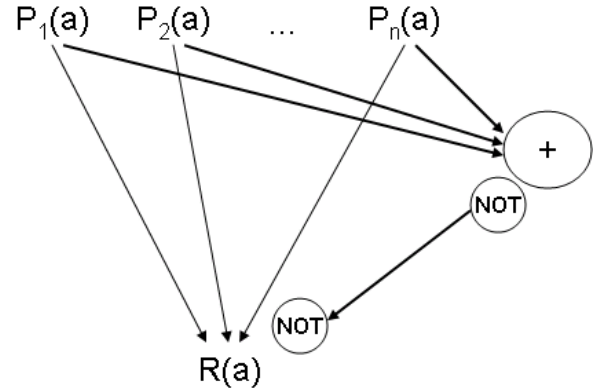


Figure 1: Captures the mandatory parent rule that the node $R(a)$ cannot exist with at least one of its parents: $NOT P_1(a) \wedge NOT P_2(a) \wedge \dots \wedge NOT P_n(a) \Rightarrow NOT R(a)$

In PCFG since the cause of every node is the parent node generating it, the constraint that at least one parent of every node should be true captured in GenProb language. Hence there can be no node that is unconnected to the parse tree.

Unique Manifestation

In the PCFG grammar G , for every non-terminal symbol all the alternate options are listed with their respective probabilities.

$$X \rightarrow (Pr_1)f_1f_2 \dots f_m$$

$$| (Pr_2)s_1s_2 \dots s_m$$

A particular non-terminal node can only generate one of the options. This implicit constraint of unique representation among alternate options is captured using the comma (,) symbol in the GenProb language and listing the mutually exclusive options with their probabilities in the same GenProb constraint. The internal weighted constraint representation of a grounded GenProb constraint of this format is shown in Figure 2.

Start Symbol

The one node in the parse tree that does not have a parent is the start node S . This constraint is captured in the GenProb language by assigning a high prior value to the object with the category of the start symbol and its time of occurrence spanning over the entire length of the input string I .

```

TRUE  $\Rightarrow$ 
IsA(?obj, S)  $\wedge$  Occur(?obj)  $\wedge$ 

```

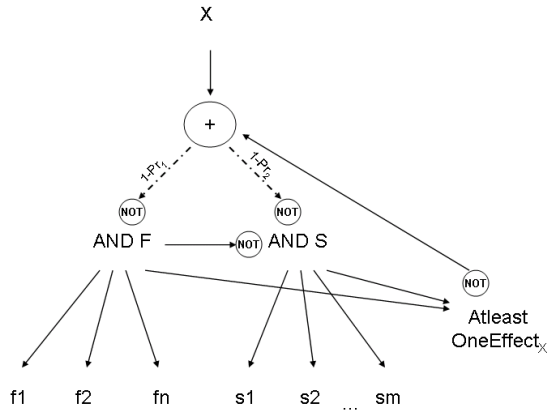


Figure 2: Shows the underlying weighted constraints of $X \rightarrow (Pr_1)f_1f_2 \dots f_m \mid (Pr_2)s_1s_2 \dots s_m$

$StartTime(?obj, Istart) \wedge EndTime(?obj, Iend)$

Mandatory Manifestation

All the leaf nodes in a parse tree have to be terminals. This axiom ensures that every non-terminal in a parse tree generates a string based on R, which we call the mandatory manifestation of non-terminals. A parse tree that does not satisfy the mandatory manifestation constraint is an invalid tree.

This constraint of saying that among all the possible generations of a non-terminal at least one of them should hold true is harder to capture. We have introduced a corresponding *AtleastOneEffect* proposition for every non-terminal node (Figure 1). The only causes for the *AtleastOneEffect* proposition of a non-terminal are the RHS components of the productions in R for this particular non-terminal. Since GenProb language has the built in causal tendency to falsify an event when all its causes are false, the only reason for *AtleastOneEffect* proposition to be true is if one of the productions in R, the rule set of PCFG, has been generated.

$Occur(?obj) \wedge NOT \text{AtleastOneEffect}(?obj)$
 $\implies FALSE$

Say there are 2 productions that can be generated from a non-terminal X;

$X \rightarrow (0.75)a \mid (0.25)YZ$

The constraints that posit *AtleastOneEffect* of the non-terminal X look like:

1.

$Occur(?aobj) \wedge IsA(?aobj, a) \wedge$
 $StartTime(?aobj, ?tStart) \wedge$
 $EndTime(?aobj, ?tEnd) \wedge$
 $Occur(?xobj) \wedge IsA(?xobj, X) \wedge$
 $StartTime(?xobj, ?tStart) \wedge$
 $EndTime(?xobj, ?tEnd)$
 $\implies \text{AtleastOneEffect}(?xobj)$

2.

$Occur(?yobj) \wedge IsA(?yobj, Y) \wedge$
 $StartTime(?yobj, ?tStart) \wedge$
 $EndTime(?yobj, ?tMid1) \wedge$
 $ImmediatelyBefore(?tMid1, ?tMid2) \wedge$
 $Occur(?zobj) \wedge IsA(?zobj, Z) \wedge$
 $StartTime(?zobj, ?tMid2) \wedge$
 $EndTime(?zobj, ?tEnd) \wedge$
 $Occur(?xobj) \wedge IsA(?xobj, X) \wedge$
 $StartTime(?xobj, ?tStart) \wedge$
 $EndTime(?xobj, ?tEnd)$
 $\implies \text{AtleastOneEffect}(?xobj)$

Though the mandatory manifestation constraint ensures that there is no unexpanded non-terminal in the tree, it is not guaranteed for the parse tree to end in terminals. PCFG rules of the form $X1 \rightarrow X11$ and $X11 \rightarrow X1$ can lead to an endless depth of the parse tree.

An Example of Interaction Enabled By GenProb

The importance of representing syntactic grammar in the same general formalism that also allows relational representations and causal conditionals is that syntax can now interact with other aspects of language like semantics, background knowledge and visual perception. For example, problems like part-of-speech tagging and word sense disambiguation, which are conventionally studied as isolated sub-problems, can be addressed by this interaction of syntax and semantics.

In order to demonstrate an example, let us consider the word “bug”. According to Wordnet, the word “bug” has the coarse senses of the nouns insect animal, system error and listening device, and also the verbs annoy and eavesdrop in this order of frequency. Given we have the corresponding frequency of these senses (Probi), the following constraint can be added to the system:

$IsA(?word, Word) \wedge HasPhonology(?word, soundBug)$
 \implies

(Prob1) $HasWordSense(?word, animalBug),$
 (Prob2) $HasWordSense(?word, systemErrorBug),$
 (Prob3) $HasWordSense(?word, deviceBug),$
 (Prob4) $HasWordSense(?word, annoyVerbBug),$
 (Prob5) $HasWordSense(?word, eavesdropVerbBug)$

By default with no further information the most frequent sense of the word is preferred. However, consider the example sentence “The bug needs a battery”. In this case, the bug refers to the noun listening device because animals and abstract concepts like errors do not require batteries, which say is available background knowledge. As the sentence is parsed and semantics is generated within the same framework, the generated semantics that an animal needs battery or that an abstract entity needs battery creates contradiction with the background knowledge. Hence, the inference system with this information concludes that the correct interpretation of the word bug is the listening device. As

an illustration, we show how the required background knowledge can be represented in GenProb.

IsA(?obj, Organic) \implies IsA(?obj, Physical)
IsA(?obj, Inorganic) \implies IsA(?obj, Physical)
IsA(?obj, Physical) \implies IsA(?obj, Entity)
IsA(?obj, Abstract) \implies IsA(?obj, Entity)
IsA(?obj, Abstract)
 \implies NOT IsA(?obj, Physical)
IsA(?obj, Organic)
 \implies NOT IsA(?obj, Inorganic)
NOT(?obj, Inorganic)
 \implies NOT Need(?obj, battery)

Related Work

Logics over infinite domains have been characterized (Milch et al., 2005, Singla & Domingos, 2007), but to our knowledge no guaranteed inference algorithm for these problems has thus far been published. Several approaches try to generalize PCFG. Hierarchical dirchilet process (Liang, Petrov et.all 2007) represent infinite number of constraints. However, the present approach is the only one to our knowledge that allows exact inference (under review) and combines in logical constraints which need not adhere to cyclicity conditions. Finally, it is anticipated that jointly reasoning over syntactic and semantic constraints in natural language processing applications will require the kind of relational language offered by the present approach.

Conclusion

PCFG is a general formalism that captures regularities in several domains, a behavior we would like from AGI systems. However, PCFGs encode only certain kinds of constraints. By translating PCFGs into a more general probabilistic framework, joint reasoning over PCFG and other constraints is possible. The constraints of PCFG have been identified and encoded in a relational language that in addition to capturing causal conditional probabilities can also represent (potentially cyclic) boolean constraints.

An example application of this integration of PCFG and probabilistic relational constraints is in the domain of language understanding. Knowledge of linguistic syntax encoded in PCFG can interact with the generated semantics of the sentence and also the world knowledge encoded in the system to effectively solve problems like lexical (or word sense) ambiguity. In the future, we would like to integrate the constraints of richer grammars like lexicalized grammars (Head-driven Phrase Structure Grammar etc) with this general representation.

References

- Anonymous. (under review). Inference with Relational Theories over Infinite Domains.
Cassimatis N.L. (2006). A Cognitive Substrate for Human-Level Intelligence. *AI Magazine*. Volume 27

Number 2.

- Charniak, E. (2000) A maximum-entropy-inspired parser. *In: Proc. NAACL*. 132-139
Collins, M. (2003) Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29.
Moore, D. and Essa, I. (2002) Recognizing multi-tasked activities from video using stochastic context-free grammar. *In Proceedings of AAAI-02*.
Domingos, P., Kok, S., Poon, H., Richardson, M., and Singla, P. (2006) Unifying Logical and Statistical AI. *Paper presented at the AAAI-06*.
Singla, P., and Domingos, P. (2007) Markov Logic in Infinite Domains. *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (pp. 368-375)*. Vancouver, Canada: AUAI Press.
Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., and Kolobov, A., 2005. "Blog: Probabilistic Models With Unknown Objects." *Proceedings of the Nineteenth Joint Conference on Artificial Intelligence*.
Ferguson, G., and J. Allen (2007). Mixed-Initiative Dialogue Systems for Collaborative Problem-Solving. *AI Magazine 28(2):23-32. Special Issue on Mixed-Initiative Assistants*. AAAI Press.
Liang, P., Petrov, S., Jordan, M. I., and Klein, D. (2007). The infinite PCFG using hierarchical Dirichlet processes. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
Nam T. Nguyen, Hung H. Bui, Svetha Venkatesh, and Geoff West. (2003) Recognising and monitoring highlevel behaviours in complex spatial environments. *In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR-03)*
Pynadath, David V.; Wellman, Michael P. (2000) Probabilistic state-dependent grammars for plan recognition. *In Proceedings of the conference on uncertainty in artificial intelligence* pp. 507-514
Percy Liang Slav Petrov Michael I. Jordan Dan Klein The Infinite PCFG using Hierarchical Dirichlet. (2007) *Processes Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 688-697, Prague, June 2007. c2007 Association for Computational Linguistics
Y. Sakakibara, M. Brown, R. C. Underwood, I. S. Mian, and D. Haussler, "Stochastic context-free grammars for modeling RNA," (2004) *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, L. Hunter, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 284-294.

Self-Programming: Operationalizing Autonomy

Eric Nivel & Kristinn R. Thórisson

Center for Analysis and Design of Intelligent Agents / School of Computer Science, Reykjavik University
Kringlunni 1, 103 Reykjavik, Iceland
{eric, thorisson}@ru.is

Abstract

Lacking an operational definition of *autonomy* has considerably weakened the concept's impact in systems engineering. Most current “autonomous” systems are built to operate in conditions more or less fully described a priori, which is insufficient for achieving highly autonomous systems that adapt efficiently to unforeseen situations. In an effort to clarify the nature of autonomy we propose an operational definition of autonomy: a *self-programming* process. We introduce Ikon Flux, a proto-architecture for self-programming systems and we describe how it meets key requirements for the construction of such systems.

Structural Autonomy as Self-Programming

We aim at the construction of machines able to adapt to unforeseen situations in open-ended environments. *Adaptation* is used here in a strong sense as the ability of a machine not only to *maintain* but also to *improve* its utility function and so, in partially specified conditions with limited resources (including time) and knowledge. As a case in point, today's Mars rovers would simply ignore the presence of an alien character waving its tentacles in front of the cameras: observers on Earth would probably see and identify it, but the rover itself would simply not be aware of this extraordinary situation and engineers would have to upload software upgrades to change its mission and plans. In sharp contrast to such engineering, we expect adaptation to be performed automatically, i.e. with *no further intervention by programmers after a machine enters service*. Our adaptable rover would be fitted with software aiming at discovering facts in a *general* fashion, that is, not limited to ultra-specific mission schemes. This software would ideally be able to generate new missions and related skills according to the context, within the limitations of its resources - hardware, energy, time horizon, etc.

Structural Autonomy

The mainstream approach outlined above consists in the main of sophisticated ways for selecting and tuning hard-coded goals and behaviors for handling situations framed in hard-coded ontologies. Systems designed this way belong to the class of *behaviorally* autonomous systems [2], and result in fact from the traditional top-down design approach: a machine's operation is fully specified, as is the full range of its operating contexts, and it is the duty of its operator to ensure that the operational conditions always comply to said specification - otherwise the system ceases to function correctly. The point here is that such machines

are meant *not to change*. Adding such change, or adaptation, to the requirements of a machine calls for an orthogonal perspective that addresses change as a desirable and controllable phenomenon. We envision motivations, goals and behaviors as being dynamically (re)constructed by the machine as a result of changes in its internal structure. This perspective - *structural* autonomy - draws on Varela's work on *operationally closed* systems [14]:

“machine[s] organized (defined as a unity) as a network of processes of production (transformation and destruction) of components which: (i) through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and (ii) constitute it (the machine) as a concrete unity in space in which they (the components) exist by specifying the topological domain of its realization as such a network.”

Although this generic definition applies primarily to biochemical substrates, it can be adapted to computational substrates. We map Varela's terminology as follows:

- *Component*: a program. The function of a program is to synthesize (i.e. to produce or to modify) other programs. For example, generating new missions is creating new programs that define goals, resource usage policies, measurement and control procedures, etc. In this view, planning is generating programs (a plan and a program to enact it). In a similar way, learning is modifying the existing programs to perform more efficiently.
- *Process*: the execution of a program.
- *Network of processes*: the graph formed by the execution of programs, admitting each other as an input and synthesizing others (rewrite graphs).
- *Space*: the memory of the computer, holding the code of the machine, exogenous components (e.g. device drivers, libraries) and its inputs/outputs from/to the world.
- *Topological domain*: the domain where synthesis is enabled as an observable and controllable process. This domain traverses increasing levels of abstraction and is defined at a low level by the synthesis rules, syntax and semantics, and at higher levels by goal and plan generating programs and related programs for measuring progress.

Program synthesis operates on *symbolic data* - the programs that constitute the machine. It follows that such constituents must be described to allow reasoning (symbolic computation) about what they do (e.g. actions on the world), what their impact will be (prediction) - or could be, given hypothetical inputs (simulation) - what they require to run (e.g. CPU power, memory, time, pre-conditions in the world), when their execution is appropriate, etc. Such descriptions constitute *models* of the programs of the machine: models encode the machine's operational semantics. The same idea can easily be extended to entities or phenomena in the world, models then encode either (1) their operational semantics in the world through the descriptions of their apparent behaviors or, (2) the operational semantics of the machine as an entity *situated* in the world (constraints and possible actions in the world, reactions from entities, etc.).

Under operational closure the utility function is defined recursively as the set of the system's behaviors, some among the latter rewriting the former in sequential steps. But this alone does not define the *purpose* of the utility function, as mere survival is not operational in the context of machines: death is no threat to a computer, whereas failure to define and fulfill its mission shall be. To remain within the scope of this paper, suffice it to say that teleology has also to be mapped from biology onto an application domain (e.g. surviving → succeeding at discovering interesting facts on a foreign planet). Program synthesis is a process that has to be designed with regards to (meta)goals in light of the current situation and available resources. Accordingly, we see "evolution" – not natural evolution but *system evolution* – as a controlled and planned reflective process. It is essentially a *global and never-terminating process of architectural synthesis*, whose output bears at every step the semantics of instructions to perform the next rewriting step. This instantiates in a computational substrate a fundamental property of (natural) evolutionary systems called *semantic closure* (see [5, 9]). Semantic closure is [8]

"a self-referential relation between the physical and symbolic aspects of material organizations with open-ended evolutionary potential: only material organizations capable of performing autonomous classifications in a self-referential closure, where matter assumes both physical and symbolic attributes, can maintain the functional value required for evolution".

A computational autonomous system is a dynamic agency of programs, states, goals and models and as such these assume the "physical" - i.e. constitutive - attributes mentioned above. System evolution must be observable and controllable, and thus has to be based on and driven by models, a requirement for systems engineering (coming from control theory). Some models describe the current structure and operation of the system, some others describe the synthesis steps capable of achieving goals according to internal drives, and finally yet some other models define

procedures for measuring progress. To summarize, a computational structurally autonomous system is (1) situated, (2) performing in real-time, (3) based on and driven by models and, (4) operationally and semantically closed. The operational closure is a continuous program/model/goal synthesis process, and the semantic closure a continuous process of observation and control of the synthesis, that results itself from the synthesis process.

Self-Programming

We call *self-programming* the global process that animates computational structurally autonomous systems, i.e. the implementation of both the operational and semantic closures. Accordingly, a self-programming machine – the *self* - is constituted in the main by three categories of code:

- C₁: the programs that act on the world and the self (sensors¹ and actuators). These are programs that evaluate the structure and execution of code (processes) and, respectively, synthesize code; they operate in any of the three categories.
- C₂: the models that describe the programs in C₁, entities and phenomena in the world - including the self in the world - and programs in the self. Goals contextualize models and they also belong to C₂.
- C₃: the states of the self and of the world - past, present and anticipated - including the inputs/outputs of the machine.

In the absence of principles for spontaneous genesis we have to assume the existence of a set of initial hand-crafted knowledge - the bootstrap segment. It consists of ontologies, states, models, internal drives, exemplary behaviors and programming skills.

Self-programming requires a new class of programming language featuring low complexity, high expressivity and runtime reflectivity. Using any of the mainstream languages available today to write a program that generates another one and integrates it in an existing system is a real challenge. First, difficulties lie in these languages lacking explicit operational semantics: to infer the purpose of source code, a program would have to evaluate the assembly code against a formal model of the machine (hardware, operating system, libraries, etc) – the latter being definitely unavailable. Second, the language structures are not reflected at assembly level either and it is practically impossible from the sole reading of the memory to rebuild objects, functions, classes and templates: one would need a complete SysML blueprint from the designer. In other words, what is good for a human programmer is not so good for a system having to synthesize its own code in real-time. As several recent works now clearly indicate (e.g. [10, 15]), a good approach is to reduce the apparent complexity of the computational substrate (language and executive) and to code short programs in assembly-style while retaining significant

¹ Sensing is acting, i.e. building - or reusing - observation procedures to sample phenomena in selected regions of the world or the self.

expressivity. More over, self-programming is a process that reasons not only about the structure of programs but also about their execution. For example a reasoning set of programs has to be aware of the resource expenditure and time horizon of a given process, of the author (program) and conditions (input and context) of code synthesis, and of the success or failure of code invocation. The programming language must then be supported by an executive able to generate runtime data on the fly to reflect the status of program rewriting.

As a foundation to implement autonomous systems for real-world conditions, automatic theorem proving is most likely not as appropriate as it may seem in theory. Theories of universal problem solving impose actually a stringent constraint: they require the exhaustive axiomatization of the problem domain and space. For proof-based self-rewriting systems (cf. [11]) this means that *complete* axiomatization is also required for the machine itself. However, modern hardware and operating systems present such a great complexity and diversity that axiomatizing these systems is already a daunting task way out of reach of today's formal engineering methods – not to mention the practicalities of cost. More over, the pace of evolution of these components is now so fast that we would need universal standards to anchor the development of industrial systems in theory. Standards taking at least two decades from inception to wide establishment, it seems that by and large, the need for exhaustive axiomatization drives theorem proving away from industrial practice. We have no choice but to accept that theories - and knowledge in general - can only be given or constructed in partial ways, and to trade provable optimality for tractability. Self-programming has thus to be performed in an experimental way instead of a theoretical way: an autonomous system would attempt to model its constituents and update these models from experience. For example, by learning the regular regime of operation of its sensors such a system could attempt to detect malfunctions or defects. It would then adapt to this new constraint, in the fashion it adapts to changes in its environment. From his perspective, the models that specify and control adaptation (program construction) are a-priori neither general nor optimal. They operate only in specific contexts, and these are modeled only partially as the dimensions of the problem space have to be incrementally discovered and validated - or defeated - by experience, for example under the control of programs that reason defeasibly (see e.g. [7]). A system continuously modeling its own operation has to do so at multiple levels of abstraction, from the program rewriting up to the level of global processes (e.g. the utility function), thus turning eventually into a fully self-modeling system (see e.g. [4]).

Open-ended evolution requires the constant observation and discovery of phenomena: these are either external to the system (e.g. a tentacle waving) or internal - in which case they constitute the phenomenology of the self-programming process. Modeling is the identification of processes underlying this phenomenology down to the level of executable knowledge - programs. On the one

hand, when no explanation is available, for example a sound featuring a distinct pitch for some reason not yet known, there is at least a *geometric saliency* we would like to capture in relevant spatio-temporal spaces. When on the other hand a phenomenon results from known dynamics, i.e. programs rewriting each other, we speak of *computational saliency*, to be observed in the system's state space. Phenomena are salient forms manifesting an underlying and possibly hidden process. They must be captured potentially at any scale - e.g. from the scale of optimizing some low-level programs to the scale of reconfiguring the entire system. Accordingly, we define states as *global and stable regimes of operation*: at the atomic level states are the stable existence of particular programs and objects (models, inputs/outputs, etc.), while higher-level states are abstract processes whose coordinates in the state space identify and/or control the execution of the programs that produce these states. From this perspective, making sense is identifying - or provoking - causal production relationships between processes: a phenomenon *P* *makes sense* through the development of its pragmatics - the effects it provokes - in the system, and *P* means another phenomenon *P'* if observing *P* leads to the same (or analogue) pragmatics as for *P'*. *Making sense* is a process performed regardless of the length or duration of production graphs; it is a process that can be observed or fostered at any arbitrary scale.

Ikon Flux: an Architecture for Self-Programming Systems

Ikon Flux is a fully implemented prototypical architecture for self-programming systems - a prototype being an abstract type to be instantiated in a concrete domain. It is not the architecture of a particular autonomous system but rather a computational substrate to frame the engineering of such architectures. It is out of the scope of this paper to provide a full and detailed description of Ikon Flux (for further details see [5]); here we will focus on how this proto-architecture meets the key requirements for self-programming discussed in the previous section.

A New Computational Substrate

Ikon Flux consists of a *language* and an *executive* designed to simplify the task of programs rewriting other programs, in a unified memory distributed over a cluster of computing nodes. The language is an interpreted, functional and data-driven language. Axiomatic objects have low-level semantics (primitive types, operators and functions) and programs are stateless and have no side effects. Programs are also kept simple by virtue of abstraction: memory allocation, parallelism, code distribution, load balancing and object synchronization are all implicit. Since programs are potential inputs/outputs for other programs, they are considered data and unified as *objects*. Primitive types define prototypical and *executable* models of code structures, in the form of graphs of short (64 bytes) code fragments. Types are dynamic and

expressed in terms of other structures, which at some point derive from axioms. For example the type *program* embeds sub-structures such as a pattern (input) and code synthesis functions (outputs): these are explicit - they specify in a program-readable way the instructions to be performed by the executive and their effects - and there is no need for an additional model to describe a program's operational semantics. Ikon Flux neither defines nor allows any opaque, coarse-grained axiomatic constructs like for example long-term memory, planner or attention mechanism. High-level structures such as these have to be either hand-coded in terms of existing structures or result from code production by existing structures. However, to encompass wider technical domains (e.g. algebra, differential topology, etc.), Ikon Flux allows the manual extension of the set of primitives with user-defined code.

Programs in Ikon Flux all run in parallel, and they *react* automatically to the presence of any other object. Reaction is constrained by patterns on code structures, and *pattern matching* is the only mechanism for evaluating formal structures. Pattern matching is *deep*, i.e. patterns are, as any object, encoded in graphs and specify sub-structures and conditions at any depth in a target structure. Pattern matching is performed by the Ikon Flux executive *system-wide*, that is, (1) on any object regardless of its distribution in the cluster and, (2) patterns can be defined as combinations of multiple and inter-dependent patterns targeting different objects amongst the entire system, i.e. to identify tuples of correlated objects. Objects in Ikon Flux have a limited lifespan, controlled by a *resilience* value, and can be activated/deactivated either as input data, via an *intensity* value, or as a program via an *activation* value. Rewriting is performed upon successful pattern-matching (1) by producing new code explicitly specified in programs and (2) by modifying the control values (resilience, intensity and activation) of target objects.

Programs in Ikon Flux encode indifferently production rules or equations (with the expressivity of first order logic) and the executive performs both forward chaining (rewriting) and backward chaining. The latter has been implemented as a support for planning, but the executive is not a planning system itself: it is the responsibility of the programs to define and constrain the search space. In that respect, Ikon Flux does not provide nor does it use any heuristics: these are to be generated - and applied - by the programs themselves to control the activation/intensity values of the objects in the system.

Runtime reflective data are automatically notified by the executive and injected in the system as objects encoded in the Ikon Flux language. For example, the invocation of a function triggers the injection of a process object which in turn will be referenced by a completion object in case of termination, indicating the run time, resource usage and the program responsible for the termination if any. Process objects are also used by the executive to notify a system of any rewriting that occurred in the past.

At a higher level of organization, the language allows the

construction of *internal sub-systems* as groups of objects that altogether contribute to a given function of the system. In this context, function means a process of arbitrary granularity, level of detail and abstraction that transforms the system state. Internal sub-systems are intended for macro-modeling purposes to describe global functions of the system itself, as well as behaviors, roles or functions of entities in the world. Internal sub-systems can be allocated a dedicated instance of the executive to perform rewritings in isolation from the main self-programming process: they can read and write the entire memory and their code can also be read, but not written, from their exterior. This is meant as a support for the modeling/construction of large-scale systems as recursive organizations of sub-systems.

There are some functions that cannot be expressed in the Ikon Flux language, either for efficiency reasons or because their re-implementation is too costly. Such functions are, for example, low-level audio/video signal processing, device drivers, and in general functions which do not need to evolve. Typically these functions are kept in their existing implementation and run on separate machines. Their code is hidden from rewriting programs and gathered in dedicated sub-systems called *external sub-systems*, which are wrapped in dedicated interfaces to communicate with the Ikon Flux executive. An interface consists of (1) specific axiomatic objects (types and functions) and (2) a converter to translate Ikon Flux objects into binary code invocation and vice versa. External sub-systems functions constitute the system's boundary in the world and are the only functions that *do* have side effects.

Global Semantics

Ikon Flux defines a *state space dimension* as an object constituted by IR and the specification of an arbitrary reference process. Projecting an object on a dimension means rating its contribution (either as an input data or as a program) to the rewriting graphs that contributed to the achievement (or failure) of the reference process. This contribution is expressed by either an intensity or an activation value that is subsequently used to compute the object's final control values. Some dimensions are known a-priori, they are application-dependent and must be given by the programmer, but some are a-priori unknown, as they relate to newly discovered phenomena, and as any other object, dimensions are in general also the subject of dynamic production and decay. To allow self-reference, any object in a given system can be projected on any dimension and such a projection is an object as well. This also holds for the dimensions and sub-spaces can thus be represented symbolically in the global state space itself.

As discussed earlier, self-programming has to perform in light of goal achievement². A general definition for "goal" is "a set of regions in the state space", and this implies the

² Geometric saliency detection is given goal semantics: the expectation of a stable form in space using a particular observation procedure - e.g. a program to detect occurrences of uncommon pitch patterns in sounds.

existence of a distance function over the state space to assess goal achievement. Thus, in addition to space dimensions, an autonomous system in Ikon Flux has to maintain and evolve distance functions, the combination of the two forming a *topological space*. As for the dimensions, the distance function is in general impossible to provide a-priori, except for the known dimensions: in general, the distance function has to be computed by the system itself, deriving programs from the programmer-supplied stock. There is, however, a particular case where the distance function can be given axiomatically: the case of *pregnance satisfaction*. Thom [12, 13] defines a *pregnance* - like hunger, fear, and reproduction in the animal reign - as an internal and global dynamics, we say *global process*, that targets abstract forms (e.g. anything edible will do for a famished dog) and constitute the ultimate reflective standard to measure the system's utility (for the dog, survival). Goals (and constraints) can be considered as *instantiated* pregnancies (e.g. eating a particular bone) and they are identified thereafter as pregnancies. In Ikon Flux a pregnancy is implemented as an object type for specifying abstract regions in the state space, using a pattern. For example, hunger could be encoded as (1) a pregnancy object P defining a pattern like "a state such as the intensity of P is lower than it is now" and (2) a program P' that modifies the intensity of P according to an underlying biological model - the execution of P' being the expression of hunger as a process. As processes, pregnancies are used to define dimensions of the state space and along these, the distance function is defined by the executive: for a given object O it is the length (in time) of the shortest rewriting path - if any - that from O leads to an increase (or decrease) of the intensity of the pregnancy. This measurement is computed by the executive for each object upon request by any program, e.g. when the intensity of a pregnancy changes.

In Thom's Semiophysics [13], when a form, e.g. a discernable event over a noisy background, becomes salient enough under the empire of a pregnancy, it can, under some conditions, trigger the contraction of event-reaction loops in shorter ones. This is called the "investing of forms by a pregnancy", or *pregnance channeling*. Thom gives an example of thereof in his interpretation of Pavlov's famous experiment: the bell becomes invested by the pregnancy hunger, to the point where its sole ringing triggers a response normally associated to the subsequent occurrence of the meat: the bell assumes the pragmatics of the meat. Subsumed by a pregnancy, a form progressively stands for - *means* - another form. In Ikon Flux pregnancy channeling can be typically implemented as the combination of (1) underlying models for pregnancies - the consumption / digestion process, to which salivating is a positive contribution - (2) programs that learn an interaction pattern - occurrence of the bell, then of the meat, then of the rewriting of the event "meat" into the salivation reaction - and (3) programs that promote this pattern to a program - rewriting "bell ringing" into the invocation of the function "salivate". *Learning* is, in our

example, identifying recurrent values for the projections of the events (occurrences of bell, meat and rewriting events) along the dimension associated to the pregnancy hunger and related intermediate processes; feedback (or reinforcement) comes as the satisfaction of the pregnancy. Pregnancy channeling constitutes a semantic association between events and reactions at a global scale: as a global learning mechanism it can lead to behavior conditioning (as in Pavlov's experiment), but it is also one possible mechanism for implementing associative memories where pregnancy channeling triggers the spreading of intensity control values across the system. Put briefly, plunging processes into topological spaces enables the identification of causal and dynamic relations throughout the entire state time-space for observing and controlling the system *as a whole*. This has also been demonstrated with a massively multi-agent architecture [1] whose essential parameters are represented in a topological space and controlled in real-time using morphological analysis.

Implementation

The development of Ikon Flux started in 1998, and in 2005 version 1.6 was used to test a system (Loki) in the field. The context was live theatrical performances³ where Loki was in charge of generating and enacting the stage control according to the development of the drama. Loki was part of the full production period, adjusting its reactions to the rehearsal events like any of the actors and stage hands, given the input of the director. Loki thus constituted a fully valid crew member as human actors responded to Loki's actions in an adaptive way, and vice versa along the lines of constructivist theatre. Loki was able to control reliably and in real-time a complex machinery (wide variety of sensors/actuators and constraints) in a rich, noisy and unstructured environment under frequent and significant changes (e.g. human behaviors, live modifications of the script). It performed both in supervised and unsupervised modes (resp. during rehearsals and live shows) and for 20 days (cumulative time). Loki performed under considerable financial and time pressure and under these conditions no formal evaluation could be conducted: this is left for the development of future systems. Although this field-testing represents a limited evaluation of the full set of Ikon Flux's ideas, it verified three key features: (1) Ikon Flux theory is implementable; (2) systems implemented on its foundation can adapt in practical real-time, both on short and long timescales; and (3) Ikon Flux architectures can be reliable enough to maintain their place in a multi-person, multi-week interaction involving continuous change while meeting real goals.

So far, our experience with Loki justifies the pursuit of further developments with great confidence. Nevertheless, much work remains to be done to address all the issues pertaining to the engineering of fully autonomous systems

³ In particular, the play *Roma Amor* - director J.M. Musial - premiered at the Cite des Sciences et de L'Industrie, Paris. Work supported by grants from the French Agency for Research (ANVAR) and Ministry of Culture.

like the Mars rover we described in the Introduction; there exist today no actual methodology for the *principled* design of non-trivial evolutionary systems. This is an open issue we are currently investigating on the basis of several techniques and concepts we have experimented for the construction of Loki. Here is a brief overview of these. First, in a traditional top-down fashion, we leverage prior knowledge by hand-crafting analogy-making programs. These are used to infer goals and plans by generating new heuristics from the ones provided in the bootstrap segment, and for unifying local models into more general ones. Second - and this is a more decisive research avenue - we identify and foster programmatically the formation and the transformation of high-level structures, functions and processes from the bottom up. This is required essentially (1) to measure and control the stability of functions, (2) to understand their formation for building new ones for arbitrary purposes, (3) to form concepts operationally for keeping the system complexity at a reasonable level as it produces new knowledge, and (4) to optimize existing rewrite graphs. To this end we have included in the bootstrap segment some “architectural” code: it consists of ad-hoc programs and models designed to identify and construct high-level organizations such as *functions* - stable coupling over time of inputs and effects - *organons* - ephemeral aggregates of code forming a substrate for a function, i.e. abstract patterns of computation flux - *organisms* - aggregates of organons operationally closed for a set of functions - and *individuals* - aggregates of functions, organons and organisms semantically closed for a pregnancy. However, to do so in a general and deterministic way remains unsolved. Notice also that, at this stage of development, a significant part of the bootstrap segment has been kept away from evolution, notably architectural code and internal drives.

Performing deep pattern matching over a massive amount of fine-grained objects⁴ in real-time is computationally intensive, but not intractable. Ikon Flux has been implemented on clusters of standard PCs running RTAI⁵. Our measurements of Loki’s performance show that, under constant load, rewriting speed scales well with the number of processors; but they also show that scalability - the admissible load - is severely limited by current networks and protocols (TCP/IP over Ethernet). As a result, a new version (2.0) is planned, targeted at multi-core processors communicating by RDMA over Infiniband.

Conclusion

In the domain of computational systems, autonomy can be operationalized by the concept of self-programming for which we have presented a prototype: a dynamic, real-time and self-referential architecture for building and grounding knowledge and processes in high-dimensional topological

spaces. For such systems, harnessing the emergence of high-order organizations in a general scalable way calls for new engineering methodologies. As we have argued here, these must be based on a process-oriented and generative approach. To this end we are currently investigating the possibility of modeling self-programming using Goldfarb’s Evolving Transformation System [3].

References

- [1] Campagne J.C. Morphologie et systèmes multi-agents. Ph.D. thesis, Université Pierre et Marie Curie, Paris. 2005
- [2] Froese T., Virgo N., Izquierdo E. Autonomy: A review and a reappraisal. In F. Almeida e Costa et al. eds. *Proc. of the 9th European Conference on Artificial Life*. Springer-Verlag, Berlin. 2007
- [3] Goldfarb L., Gay D. What is a structural representation? Fifth variation, Faculty of Computer Science, University of New Brunswick, Technical Report TR05-175. 2005
- [4] Landauer C., Bellman K.L. Self-Modeling Systems. In R. Laddaga, H. Shrobe eds. *Self-Adaptive Software: Applications. Springer Lecture Notes in Computer Science*, 2614:238-256. 2003
- [5] Nivel E. Ikon Flux 2.0. Reykjavik University, School of Computer Science Technical Report. RUTR-CS07006. 2007
- [6] Pattee H. *Evolving Self-Reference: Matter, Symbols, and Semantic Closure*. In *Communication and Cognition. Artificial Intelligence*, 12(1-2). 1995
- [7] Pollock J. OSCAR: An agent architecture based on defeasible reasoning. In *Proc. of the 2008 AAAI Spring Symposium on Architectures for Intelligent Theory-Based Agents*. 2008
- [8] Rocha L.M. ed. *Communication and Cognition. Artificial Intelligence*, Vol. 12, Nos. 1-2, pp. 3-8, Special Issue *Self-Reference in Biological and Cognitive Systems*. 1995
- [9] Rocha L.M. Syntactic autonomy, cellular automata, and RNA editing: or why self-organization needs symbols to evolve and how it might evolve them. In Chandler J.L.R. and G, Van de Vijver eds. *Closure: Emergent Organizations and Their Dynamics. Annals of the New York Academy of Sciences*, 901:207-223. 2000
- [10] Schmidhuber J. Optimal Ordered Problem Solver. *Machine Learning*, 54, 211-254. 2004
- [11] Schmidhuber J. Gödel machines: Fully Self-Referential Optimal Universal Self-Improvers. In Goertzel B. and Pennachin C. eds. *Artificial General Intelligence*, p. 119-226, 2006.
- [12] Thom R. *Structural Stability and Morphogenesis*. Reading, MA: W. A. Benjamin. 1972
- [13] Thom R. *Semiophysics: A Sketch*. Redwood City: Addison-Wesley. 1990
- [14] Varela F.J., Maturana H.R. *Autopoiesis and Cognition: The Realization of the Living*. Boston, MA: Reidel. 1980
- [15] Yamamoto L., Schreckling D., Meyer T. Self-Replicating and Self-Modifying Programs in Fraglets. *Proc. of the 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*. 2007

⁴ Loki was constituted by roughly 300 000 objects in average.

⁵ A real-time kernel extension for Linux [www.rtai.org] paired with RTnet a real-time network protocol stack [www.rts.uni-hannover.de/rtnet]

Bootstrap Dialog: A Conversational English Text Parsing and Generation System

Stephen L. Reed

Texai.org
3008 Oak Crest Ave, Austin TX, 78704
stephenreed@yahoo.com

Abstract

A conversational English text parsing and generation system is described in which its lexicon and construction grammar rules are revised, augmented, and improved via dialog with mentors. Both the parser and generator operate in a cognitively plausible, incremental manner. Construction Grammar is well suited for a precise and robust dialog system due to its emphasis on pairing utterance form with exact logical meaning. Combining lexicon representation and grammar rule representation from the theory of Fluid Construction Grammar, with grammar constructions adopted from Double R Grammar, the system is designed to accommodate wide coverage of the English language.

Introduction

Alan Turing, in his seminal paper on artificial intelligence (Turing 1950), proposed to create a mechanism that simulates a child's mind, and then to subject it to an appropriate course of education thus achieving an artificial general intelligence capable of passing his imitation game. Texai is an open source project to create artificial intelligence. Accordingly, the Texai project is developing conversational agents which are capable of learning concepts and skills by being taught by mentors.

AGI Organization

The Texai AGI, will consist of a vastly distributed set of skilled agents, who are members of mission-oriented agencies that act in concert. Texai agents will be organized as a hierarchical control structure, as described by James Albus (Albus and Meystel 2002). Plans call for users to either converse with a remote, shared Texai instance, or download their own instance for improved performance. Instances host one or more Texai agents and are physically organized as a cloud in which there are no isolated instances. Each Texai agent maintains a cached working set of knowledge safely encrypted, replicated, and persisted in the cloud.

This approach contrasts with Novemente (Goertzel 2006).

Copyright © 2008, The Second Conference on Artificial General Intelligence (agi-09.org). All rights reserved.

Although Novemente also employs the artificial child in their development road map, English dialog is not the sole method by which Novemente learns.

One good way to provide mentors for Texai agents is to apply them to human organizations such that each human member has one or more Texai agents as proxies for the various roles the human fills in the organization. The author hopes that Texai will be embraced by, and extend, human organizations. A multitude of volunteers may subsequently mentor the many agents that will comprise Texai.

Bootstrap Dialog

The initial Texai conversational agent is being developed to process a controlled language consisting of a minimal subset of English vocabulary and grammar rules. This controlled language is sufficient to acquire new vocabulary and new grammar rules from its human mentors. The bootstrap dialog system is designed to be taught skills that enhance its programmable capabilities. Texai addresses eight dialog challenges identified by James Allen (Allen et. al. 2000): (1) intuitive natural English input, (2) robustness in the face of misunderstandings, (3) mixed-initiative interaction, (4) user intention recognition, (5) effective grounding and ensuring mutual understanding, (6) topic change tracking, (7) dialog-based response planning to provide the appropriate level of information, and (8) portability so that the dialog system can operate with disparate knowledge domains.

Incremental Processing

The Texai grammar engine operates incrementally, in a cognitively plausible manner. During parsing, words are processed strictly left-to-right with no backtracking. As object referring expressions (e.g. noun phrase) are detected, all semantic and referential interpretations are considered simultaneously for elaboration and pruning by two respective spreading activation mechanisms.

Knowledge Base and Lexicon

The Texai knowledge base is derived from an RDF compatible subset of OpenCyc , and elaborated with RDF extracts from WordNet , Wiktonary, and the CMU Pronouncing Dictionary. The Texai lexicon is available in RDF and N3 format. It features good coverage of English word forms, pronunciations, word senses, glosses, and sample phrases, and an initially modest set of OpenCyc term mappings to word senses.

Knowledge base entities may be mapped into Java objects by the Texai RDF Entity Manager (Reed 2006). Operating in a manner similar to an object-to-relational mapper, the RDF Entity Manager facilitates the automatic retrieval and persistence of Java lexicon and grammar rule objects into the Sesame RDF store.

OpenCyc (Matuszek et al. 2006)

The OpenCyc knowledge base was extracted into RDF format, retaining only atomic terms and contextualized binary assertions. Approximately 130,000 class and individual concept terms are present in the extracted KB. About 12,000 of these terms are linked to WordNet synsets. It is a goal of the Texai project, via dialog with mentors, to complete the mapping of relevant word senses to OpenCyc terms, and to create new Texai terms filling gaps in OpenCyc.

WordNet (Feldman 1999)

WordNet version 2.1 contains lexical and taxonomic information about approximately 113,000 synonym sets. It was fully extracted into RDF for the Texai project.

Wiktonary

This user-authored dictionary is based upon the same platform as Wikipedia. Its XML dump as of September, 2007 was processed into RDF, in a form compatible with WordNet word sense descriptions.

The CMU Pronouncing Dictionary

(<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>)

This dictionary contains entries for over 125,000 English word forms, and gives each pronunciation as a sequence of phonemes in the ARPABET phonetic alphabet. It is compatible with both the CMU Sphinx automatic speech recognition tools and the CMU Festival speech generation tool. These speech tools are planned as the speech interface for Texai, in addition to its existing text chat interface. The dictionary was processed into RDF, compatible with WordNet word form descriptions.

Merged Texai Lexicon

Using WordNet as the framework, non-conflicting word senses were merged in from Wiktionary. OpenCyc provides corresponding KB terms for 12,000 WordNet synsets. Matching word forms received ARPABET phoneme sequences from the CMU Pronouncing

Dictionary.

<i>KB Component</i>	<i>Nbr. of RDF Statements</i>
OpenCyc	640,110
WordNet	4,134,543
Wiktonary	3,330,020
The CMU Pronouncing Dictionary	3,772,770
Merged Texai Lexicon	10,407,390

Table 1. KB and Lexicon Components

The Texai KB is physically partitioned by KB component in order for each to fit in main memory (i.e. 2 GB) and provide high performance via a Sesame RDF quad store.

Construction Grammar

Fluid Construction Grammar (FCG) (Steels & De Beule 2006) is a natural language parsing and generation system developed by researchers at emergent-languages.org. The system features a production rule mechanism for both parsing and generation using a reversible grammar. FCG provides a rule application engine in which the working memory (WM) is a coupled semantic and syntactic feature structure. FCG itself does not commit to any particular lexical categories, nor does it commit to any particular organization of construction rules. Like all construction grammars, FCG is a paring between form and meaning. The Texai system extends FCG so that it operates incrementally, word by word, left to right in English. Furthermore, Texai improves the original FCG implementation by adopting a simplified working memory feature structure and by substituting tailored unification for each of the five rule types, instead of using a generic list unification mechanism for construction rule matching. Texai rule formulation also improves on FCG by allowing optional unit constituents in grammar rules, thus reducing dramatically the otherwise large number of explicit permutations.

Double R Grammar

Double R Grammar (DRG) (Ball 2007), previously implemented in the ACT-R cognitive architecture (Ball et al. 2007), is a linguistic theory of the grammatical encoding and integration of referential and relational meaning in English. Its referential and relational constructions facilitate the composition of logical forms. In this work, a set of bi-directional FCG rules are developed that comply with DRG. Among the most important constituents of DRG is the *object referring expression* (ORE), which refers to a new or existing entity in the discourse context. The Texai system maps each ORE to a KB concept. ORE's are related to one another via *situation referring expressions* (SRE), in which the

relation is typically a verb. In the below example, which is formatted in the style of FCG, a PredicatePreposition WM unit is stated to be composed of a Preposition WM unit followed by an ORE construction. Each bi-directional rule consists of units (e.g. ?Prep) having features and attributes. The J unit specifies the head of the rule.

```
(con
  con-PredPrep
  ((category basic-construction))
  ((?Prep
    (category Preposition)
    (referent-subj ?subj)
    (referent-obj ?obj))
    (?ObjReferExpr
      (category ObjectReferringExpression)
      (referent ?obj))
    (?top
      (subunits (== ?Prep ?ObjReferExpr)))
    ((J ?PredPrep)
      (category PredicatePreposition)
      (referent ?obj)
      (referent-subj ?subj))))
```

Figure 1. An Example FCG Rule for a DRG Construction

User Modeling

The knowledge base contains a persistent, contextualized model of each user's belief state. The system is designed to avoid telling the user something that the system knows that the user already knows. This facility is chiefly employed during utterance generation, in which the actual belief state of the user is to be updated with some particular set of propositions.

Discourse Context

The Texai dialog system contains a discourse context for each user interaction session. Each discourse context consists of a list of utterance contexts, each of which represents a single utterance from either the system or the user. Attributes of the utterance context include a timestamp, the speaker identity, the utterance text, a cache of the speaker's preferred word senses, and either the source propositions for a generated utterance, or the understood propositions for a parsed utterance. It is intended that the system learn via reinforcement the number of utterance contexts to maintain, and the degree to which to decay their relative importance.

Incremental Parsing

The dialog system performs incremental utterance parsing in a cognitively plausible manner. As argued by Jerry Ball (Ball 2006) this method avoids possible combinatorial explosions when computing alternative interpretations, and interfaces tightly with automatic speech recognizers.

Indeed, it is planned that Texai augment the CMU Sphinx automatic speech recognition tool's language model with respect to scoring alternative recognized words.

Parsing Rule Application

In figure 1 above, Referent variables ?subj and ?obj facilitate the instantiation of logical propositions located throughout a single parsing interpretation. When the above rule is applied in the parsing direction, it matches a Preposition unit in the working memory feature structure being assembled, while binding the ?Prep variable to the corresponding WM unit name. The ObjectReferringExpression WM unit must immediately follow the Preposition in order for this rule to match. As a result of applying the rule in the parsing direction, a new PredicatePreposition WM unit is created in the WM feature structure. This new WM unit has the target Preposition and ObjectReferringExpression WM units as subunits. Incremental processing is facilitated during rule application by hiding already-subordinated WM units, and by focusing rule application on recently created WM units. Incremental processing is achieved by allowing grammar rules to partially match. When processing moves beyond the rightmost required and as-yet unmatched unit of a partially matched rule, its branch of the interpretation tree is pruned.

Kintsch Construction/Integration

Walter Kintsch (Kintsch 1998) proposed a model for reading comprehension, based upon cognitive principles and tested empirically with human subjects. In what he called Construction/Integration, all alternative interpretations are simultaneously considered. For each interpretation, elaborations are constructed in the discourse context (i.e. working memory). Then an iterative spreading activation procedure scores sets of interpretation propositions according to how well connected they are to the concepts initially in the discourse context.

Discourse Elaboration

In the Texai system, discourse elaboration takes place by a marker-passing spreading activation mechanism (Hendler 1998). Discourse elaboration is performed before Kintsch spreading activation so that ambiguous concepts in the input utterance might be inferred to be conceptually related to previously known concepts in the discourse context. In the example presented below, the word "table" is ambiguous. It could either mean cyc:Table, or as part of the multiple word form "on the table", mean subject to negotiation. There is no known table in the discourse context, but there is a known instance of cyc:RoomInAConstruction. Suppose there exists these commonsense rules in the Texai knowledge base:

- a room may typically contain furniture
- a room may typically have a window
- a room has a ceiling
- a room has a wall

- a room has a floor
- a room has a door
- a room has a means of illumination
- a room can contain a person
- a table is a type of furniture
- a family room is a type of room

Discourse elaboration, via spreading activation, could add furniture, and subsequently table, to the discourse context by activating cached links derived from these rules. A later example will demonstrate.

Pruning By Spreading Activation

Analogous to how automatic speech recognizers operate, the number of retained interpretations in the search space is kept to a specified beam width (e.g. four retained interpretations). At the conclusion of utterance parsing, the highest scoring interpretation is returned as the result.

An Example

The following diagrams were produced during the processing of the example utterance: “the book is on the table“. As part of the experiment, the discourse context is primed with knowledge of a room, which is an instance of *cyc:RoomInAConstruction*, and a book, which is an instance of *cyc:BookCopy*. Lexical grammar rules matching word stems in the example utterance yield these ambiguous meanings:

- book - a bound book copy
- book - a sheath of paper, e.g. match book
- is - has as an attribute
- is - situation described as
- on - an operational device
- “on the table” - subject to negotiation [a multiword word form]
- on - located on the surface of

These concepts form nodes in a graph, whose links designate a conceptual relationship between two concepts. Marker-passing spreading activation originates at the known discourse terms (e.g. *cyc:RoomInAConstruction*) and at each significant utterance term (e.g. *cyc:Table*) and terminates if paths meet (e.g. at *cyc:FurniturePiece*). When it can be inferred in this fashion that an utterance term and a known discourse term are conceptually related, then that proposition is added to the meaning propositions for subsequent Kintsch spreading activation to resolve ambiguities. The marker-passing spreading activation decays after only a few links to preclude weakly conceptually related results.

The Texai dialog system maintains a tree of parsing interpretation nodes. Each node in the tree is either an input word, such as ‘the’, or the name of an applied fluid construction grammar rule. Branches in this tree occur when there are alternative interpretations (i.e. meanings) for a word such as “book“. The parsing interpretation tree

is retained after the parsing process completes so that the user can ask questions about the parsing state (e.g. why a certain grammar rule did not apply as expected).

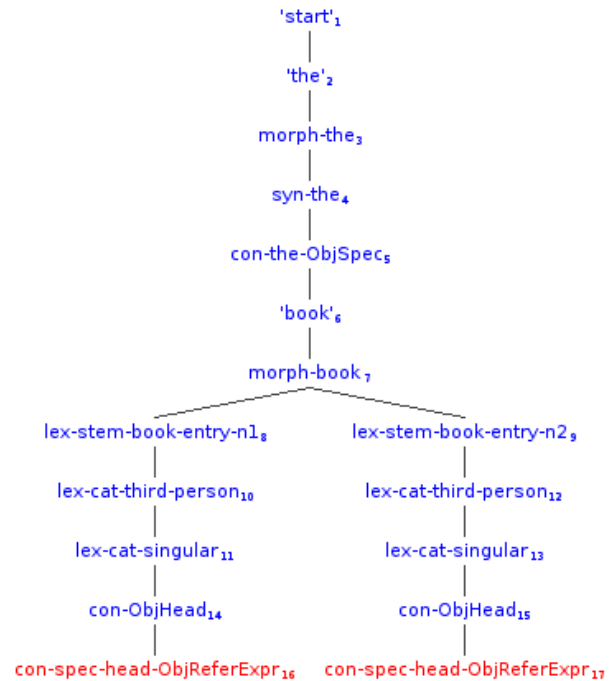


Figure 2. Two alternative interpretations of “book”

Figure 2 depicts the tree of two alternative parsing interpretations for the partial utterance “the book” whose leaves are: node 16 which represents an instance of *cyc:BookCopy*, and node 17 which represents an instance of *texai:SheetsBoundTogetherOnOneEdge*. Quoted nodes in the tree represent incrementally parsed words, and the remaining nodes name the applied grammar rule.

According to Walter Kintsch’s theory of reading comprehension, spreading activation flows over the nodes of a graph formed by the meaning propositions of the utterance. Links in this graph connect nodes mentioning the same term. The most relevant set of nodes receives the highest activation.

In figure 2 below are the ten propositions from the alternative parsing interpretations of the phrase “the book“. In the corresponding figure 3, magenta colored nodes indicate the interpretation: *SheetsBoundTogetherOnOneEdge*, Cyan colored nodes indicated the alternative interpretation *cyc:BookCopy*. The yellow nodes indicates prior knowledge - N4 is the prior discourse context knowledge about a *cyc:Table*, and N5 is the prior discourse context knowledge about a *cyc:BookCopy*. N1 and N7 are positively connected, which is indicated by a black line, because they share the concept: *SheetsBoundTogetherOnOneEdge-2*. Node N1 and N10 are negatively connected, which is indicated by a red line, because they represent alternative, conflicting, interpretations.

<i>node</i>	<i>RDF proposition</i>
N1	[texai:SheetsBoundTogetherOnOneEdge-2 texai:fcgStatus texai:SingleObject]
N2	[texai:BookCopy-1 rdf:type cyc:BookCopy]
N3	[texai:BookCopy-1 texai:fcgDiscourseRole texai:external]
N4	[texai:table-0 rdf:type cyc:Table]
N5	[texai:book-0 rdf:type cyc:BookCopy]
N6	[texai:BookCopy-1 rdf:type texai:PreviouslyIntroducedThingInThisDisco urse]
N7	[texai:SheetsBoundTogetherOnOneEdge-2 rdf:type texai:PreviouslyIntroducedThingInThisDisco urse]
N8	[texai:SheetsBoundTogetherOnOneEdge-2 rdf:type texai:SheetsBoundTogetherOnOneEdge]
N9	[texai:SheetsBoundTogetherOnOneEdge-2 texai:fcgDiscourseRole texai:external]
N10	[texai:BookCopy-1 texai:fcgStatus texai:SingleObject]

Figure 3. RDF Propositions From Two Alternative Interpretations

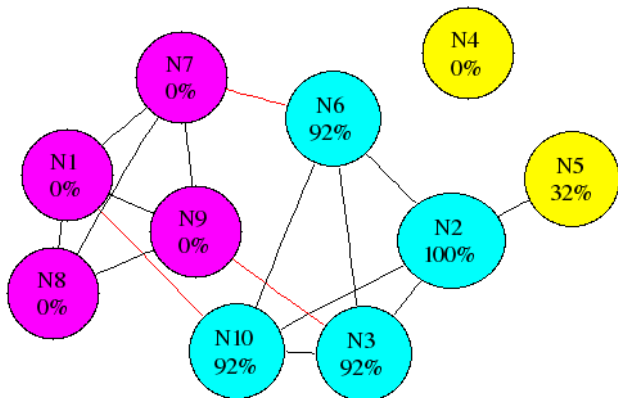


Figure 4. Quiesced Kintsch Spreading Activation Graph

Incremental Generation

The dialog system performs incremental utterance generation. Presently, the dialog planner is rudimentary, and consists of a component that forms a semantic dependence tree from terms in the set of propositions to be communicated to the user. The RDF propositions are gathered by their RDF subject term. One of the terms is heuristically chosen to be the subject of the utterance. Each of the propositions having this term as an RDF subject is selected for the root semantic dependency node. Child nodes are likewise created heuristically for the remaining propositions, grouped by RDF subject term. Incremental generation proceeds in much the same fashion as incremental parsing due to the fact that FCG is bi-

directional. As rules match, the resulting utterance is generated left-to-right, word by word. Whenever no rules match, the propositions from the next semantic dependency node are added to the top unit WM feature structure. Pruning of alternative interpretations will be a future research issue. Currently, simple scoring heuristics are:

- prefer fewer words
- prefer to reuse previously uttered words for a given meaning term
- prefer to use words that the recipient is otherwise likely to know

Finally, the resulting generated utterance is trial parsed to ensure that the system can understand what it generates with respect to the discourse context and its model of the user's belief state.

Vocabulary Acquisition

Once the most basic English grammar constructions are hand-coded, it is planned that Texai learn, by being taught, the constructions required for it to comprehend the word sense glosses (i.e. definitions) from WordNet and Wiktionary. By converting this definitional text into crisp, symbolic logic statements, Texai will acquire a degree of commonsense understanding about the defined concepts. The author is postponing grounding most of these concepts in physical perceptions. Initially, the only fully grounded symbols will be those involved with English grammar constructions and vocabulary. That is, Texai will have a grounded perception of what an utterance is, because it can directly sense one and it can generate one. It will likewise have a grounded notion of its mentors. It will be able to sense what each one says to it, and to influence their belief state by what it says.

It might be considered circular and shallow to define the various dictionary words in terms of each other. But this will be sufficient for question answering - e.g. better than an existing search engine on the same corpus. And one might reasonably expect that it will be sufficient for the really important step to soon follow - the acquisition of skills by being taught them.

Grammar Rule Acquisition

At first, rather simple, stylized dialogs will handle idioms and multi-word forms by requesting their corresponding meaning from the user. It remains the subject of future experimentation to find the best way to input logical formulas. The initial idea will be to accept RDF syntax.

After a sufficient vocabulary for describing FCG constructions is developed, then Texai can be taught additional complex grammar rules. This facility will be preceded by a sub-facility that will enable the dialog system to describe to its mentor why a grammar rule succeeded or failed for a particular parsed or generated utterance.

Future Work and Conclusion

Future experiments will explore the hypothesis that bootstrapping works. Here is an example use case:

[user] Do you understand “The book is on neuroscience”?
[texai] Not entirely, neuroscience is not a physical location, how can the book be on it?
[user] The utterance means that the book is about neuroscience.
[texai] Could I generalize “on neuroscience” to any field of scientific discovery, such as “on Astronomy”, and mean that the book is about Astronomy?
[user] yes
[texai] Could I generalize field of scientific discovery to some general region, such “on deep space?”
[user] yes, this role can be generalized to any topic.
[texai] Does the utterance “the book is on Mars” entail either the interpretation that (1) the book is physically located on Mars, or (2) the book is about the topic Mars?
[user] yes
...
[user] Do you understand “the book is on fire?”
[texai] Yes, the book is about the topic fire.
[user] There is another, more common, interpretation: There is a fire in which the book is the thing that is burning.
[texai] Could I generalize fire to any decomposition process, such as rot?
[user] no
...
[user] Do you understand “the book is on my to-do list?”
[texai] Yes, the book is about the topic to-do list.
[user] There is another, more common, interpretation: There is a to-do list, and the book is an element of the list.
[texai] I know from Wiktionary that a to-do list is “a list of errands and other tasks - often written on a piece of paper as a memory aid - that one needs or intends to accomplish”, but a book is not a task.
[user] metonymy: “the book” means “the task of reading the book”
[texai] Now I understand that “the book is on my to-do list” commonly means that the task of reading the book is a task that I intend to do.

References

Albus, J. S., Meystel, A. M., Intelligent Systems: architecture, design, and control, John Wiley & Sons, New York, USA, 2002

Allen, A., et. al., An Architecture for a Generic Dialogue Shell, In *Natural Language Engineering, Vol. 6*, Cambridge University Press, New York, New York, USA, September 2000.

Ball, J., Heiberg, A. & Silber, R. Toward a Large-Scale Model of Language Comprehension in ACT-R 6. In *Proceedings of the 8th International Conference on Cognitive Modeling*, pages 163-168, 2007

Ball, J. (2006). Can NLP Systems be a Cognitive Black Box? In *Papers from the AAI Spring Symposium, Technical Report SS-06-02*, pages 1-6, AAAI Press, Menlo Park, California, USA, 2006
<http://www.doublertheory.com/NLPBlackBox.pdf>

Ball, J. Double R Grammar, 2003
<http://www.DoubleRTheory.com/DoubleRGrammar.pdf>

Feldbaum, C. ed., WordNet: an electronic lexical database, MIT Press, Cambridge, Massachusetts, USA, 1999

Goertzel, B., The Hidden Pattern: A Patternist Philosophy of Mind, Chapter 15, Brown Walker Press, Boca Raton, Florida, USA, 2006

Hendler, J. A., Integrating Marker-Passing and Problem Solving, Chapter 8 – Cognitive Aspects, Lawrence Erlbaum Associates, Hillsdale, New Jersey, USA, 1988

Kintsch, W., Comprehension: a paradigm for cognition, Cambridge University Press, Cambridge, UK, 1998

Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J. An Introduction to the Syntax and Content of Cyc. In *Proceedings of the 2006 AAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, Stanford, CA, USA, March 2006.

Reed, S. L., Semantic Annotation for Persistence, In *Proceedings of the AAI Workshop on Semantic e-Science*, AAAI Press, Menlo Park, California, USA, 2006

Reiter, E., Dale R., Building Natural Language Generation Systems, Cambridge University Press, Cambridge, UK, 2000

Steels, L. and De Beule, J. (2006) A (very) Brief Introduction to Fluid Construction Grammar. In *Third International Workshop on Scalable Natural Language Understanding (2006)*.
<http://arti.vub.ac.be/~joachim/acl-ny-06-3.pdf>

Turing, A. M. Computing Machinery and Intelligence. In *Mind* 59, 1950.

Analytical Inductive Programming as a Cognitive Rule Acquisition Devise*

Ute Schmid and Martin Hofmann and Emanuel Kitzelmann

Faculty Information Systems and Applied Computer Science
University of Bamberg, Germany

{ute.schmid, martin.hofmann, emanuel.kitzelmann}@uni-bamberg.de

Abstract

One of the most admirable characteristic of the human cognitive system is its ability to extract generalized rules covering regularities from example experience presented by or experienced from the environment. Humans' problem solving, reasoning and verbal behavior often shows a high degree of systematicity and productivity which can best be characterized by a competence level reflected by a set of recursive rules. While we assume that such rules are different for different domains, we believe that there exists a general mechanism to extract such rules from only positive examples from the environment. Our system IGOR2 is an analytical approach to inductive programming which induces recursive rules by generalizing over regularities in a small set of positive input/output examples. We applied IGOR2 to typical examples from cognitive domains and can show that the IGOR2 mechanism is able to learn the rules which can best describe systematic and productive behavior in such domains.

Introduction

Research in inductive programming is concerned with the design of algorithms for synthesis of recursive programs from incomplete specifications such as input/output examples of the desired program behavior, possibly together with a set of constraints about size or time complexity (Biermann, Guiho, & Kodratoff 1984; Flener 1995). In general, there are two distinct approaches to inductive programming – search-based generate-and-test algorithms (Olsson 1995; Quinlan & Cameron-Jones 1995) and data-driven analytical algorithms (Summers 1977; Kitzelmann & Schmid 2006). In the first case, given some language restriction, hypothetical programs are generated, tested against the specification and modified until they meet some given criteria. In the second case, regularities in the input/output examples are identified and a generalized structure is built over the examples. While search-based approaches – in principle – can generate each possible program and therefore might be able to find the

desired one given enough time, analytical approaches have a more restricted language bias. The advantage of analytical inductive programming is that programs are synthesized very fast, that the programs are guaranteed to be correct for all input/output examples and fulfill further characteristics such as guaranteed termination and being minimal generalizations over the examples. The main goal of inductive programming research is to provide assistance systems for programmers or to support end-user programming (Flener & Partridge 2001).

From a broader perspective, analytical inductive programming provides algorithms for extracting generalized sets of recursive rules from small sets of positive examples of some behavior. Such algorithms can therefore be applied not only to input/output examples describing the behavior of some program but to arbitrary expressions. Taking this standpoint, analytical inductive programming provides a general device for the acquisition of generalized rules in all such domains where it is natural that people are typically exposed to only positive examples. This is, for example, the case in learning correct grammatical constructions where a child would never get explicitly exposed to scrambled sentences (such as *house a is this*).

In the sixties, Chomsky proposed that the human mind possesses a language acquisition device (LAD) which allows us to extract grammar rules from the language experience we are exposed to (Chomsky 1959; 1965). Input to this device are the linguistic experiences of a child, output is a grammar reflecting the linguistic competence. The concept of an LAD can be seen as a special case of a general cognitive rule acquisition device. Unfortunately, this idea became quite unpopular (Levelt 1976): One reason is, that only performance and not competence is empirically testable and therefore the idea was only of limited interest to psycho-linguists. Second, Chomsky (1959) argued that there “is little point in speculating about the process of acquisition without much better understanding of what is acquired” and therefore linguistic research focussed on search for a universal grammar. Third, the LAD is concerned with *learning* and learning research was predominantly associated with Skinner's reinforcement learning approach which clearly is unsuitable as a lan-

*Research was supported by the German Research Community (DFG), grant SCHM 1239/6-1.
Copyright © 2008, The Second Conference on Artificial General Intelligence (AGI-09.org). All rights reserved.

guage acquisition device since it explains language acquisition as selective reinforcement of imitation.

Since the time of the original proposal of the LAD there was considerable progress in the domain of machine learning (Mitchell 1997) and we propose that it might be worthwhile to give this plausible assumption of Chomsky a new chance. The conception of inductive biases (Mitchell 1997) introduced in machine learning, namely restriction (i.e. language) and preference (i.e. search) bias might be an alternative approach to the search of a universal grammar: Instead of providing a general grammatical framework from which each specific grammar – be it for a natural language or for some other problem domain – can be derived, it might be more fruitful to provide a set of constraints (biases) which characterize what kinds of rule systems are learnable by humans. Since we are interested in a mechanism to induce general, typically recursive, rules and not in classification learning, we propose to investigate the potential of analytical inductive programming as such a general rule acquisition device. Furthermore, we propose to take a broader view of Chomsky’s idea of an LAD and we claim that rule acquisition in that fashion is not only performed in language learning but in all domains where humans acquire systematic procedural knowledge such as problem solving and reasoning.

In the following we give a short overview of our analytical inductive programming system IGOR2 together with its biases. Then we illustrate IGOR2’s ability as a cognitive rule acquisition device in the domains of problem solving, reasoning, and natural language processing.¹

Recursive Structure Generalization

IGOR2 (Kitzelmann 2008) was developed as a successor to the classical THESYS system for learning Lisp programs from input/output examples (Summers 1977) and its generalization IGOR1 (Kitzelmann & Schmid 2006). To our knowledge, IGOR2 is currently the most powerful system for analytical inductive programming. Its scope of inducible programs and the time efficiency of the induction algorithm compares well with inductive logic programming and other approaches to inductive programming (Hofmann, Kitzelmann, & Schmid 2008). The system is realized in the constructor term rewriting system MAUDE. Therefore, all constructors specified for the data types used in the given examples are available for program construction. Since IGOR2 is designed as an assistant system for *program* induction, it relies on small sets of noise-free positive input/output examples and it cannot deal with uncertainty. Furthermore, the examples have to be the first inputs with respect to the complexity of the underlying data type. Given these restrictions, IGOR2 can guarantee that the induced program covers *all* examples correctly and provides a minimal generalization over them. Classification learning

for noise-free examples such as `PlayTennis` (Mitchell 1997) can be performed as a special case (Kitzelmann 2008).

IGOR2 specifications consist of such a set of examples together with a specification of the input data type. Background knowledge for additional functions can (but needs not) be provided. IGOR2 can induce several dependent target functions (i.e., mutual recursion) in one run. Auxiliary functions are invented if needed. In general, a set of rules is constructed by generalization of the input data by introducing patterns and predicates to partition the given examples and synthesis of expressions computing the specified outputs. Partitioning and search for expressions is done systematically and completely which is tractable even for relative complex examples because construction of hypotheses is data-driven. IGOR2’s restriction bias is the set of all functional recursive programs where the outermost function must be either non-recursive or provided as background knowledge.

IGOR2’s built-in preference bias is to prefer fewer case distinctions, most specific patterns and fewer recursive calls. Thus, the initial hypothesis is a single rule per target function which is the least general generalization of the example equations. If a rule contains unbound variables on its right-hand side, successor hypotheses are computed using the following operations: (i) Partitioning of the inputs by replacing one pattern by a set of disjoint more specific patterns or by introducing a predicate to the right-hand side of the rule; (ii) replacing the right-hand side of a rule by a (recursive) call to a defined function where finding the argument of the function call is treated as a new induction problem, that is, an auxiliary function is invented; (iii) replacing sub-terms in the right-hand side of a rule which contain unbound variables by a call to new subprograms.

Problem Solving

Often, in cognitive psychology, speed-up effects in problem solving are modelled simply as composition of primitive rules as a result of their co-occurrence during problem solving, e.g., knowledge compilation in ACT (Anderson & Lebière 1998) or operator chunking in SOAR (Rosenbloom & Newell 1986). Similarly, in AI planning macro learning was modelled as composition of primitive operators to more complex ones (Minton 1985; Korf 1985). But, there is empirical evidence that humans are able to acquire general problem solving strategies from problem solving experiences, that is, that generalized strategies are learned from sample solutions. For example, after solving Tower of Hanoi problems, at least some people have acquired the recursive solution strategy (Anzai & Simon 1979). Typically, experts are found to have superior *strategic* knowledge in contrast to novices in a domain (Meyer 1992).

There were some proposals to the learning of domain specific control knowledge in AI planning (Shell & Carbonell 1989; Shavlik 1990; Martín & Geffner 2000). All these approaches proposed to learn cyclic/recursive

¹The complete data sets and results can be found on www.cogsys.wiai.uni-bamberg.de/effalip/download.html.

Problem domain:

```
puttable(x)
PRE: clear(x), on(x, y)
EFFECT: ontable(x), clear(y), not on(x,y)
```

Problem Descriptions:

```
: init-1 clear(A), ontable(A)
: init-2 clear(A), on(A, B), ontable(B)
: init-3 on(B, A), clear(B), ontable(A)
: init-4 on(C, B), on(B, A), clear(C), ontable(A)
: goal clear(a)
```

Problem Solving Traces/Input to IGOR2

```
fmod CLEARBLOCK is
  *** data types, constructors
  sorts Block Tower State .
  op table : -> Tower [ctor] .
  op _ : Block Tower -> Tower [ctor] .
  op puttable : Block State -> State [ctor] .
  *** target function declaration
  op ClearBlock : Block Tower State -> State [metadata "induce"] .
  *** variable declaration
  vars A B C : Block .
  var S : State .
  *** examples
  eq ClearBlock(A, A table, S) = S .
  eq ClearBlock(A, A B table, S) = S .
  eq ClearBlock(A, B A table, S) = puttable(B, S) .
  eq ClearBlock(A, C B A table, S) = puttable(B, puttable(C, S)) .
endfm
```

Figure 1: Initial experience with the *clearblock* problem

control rules which reduce search. Learning recursive control rules, however, will eliminate search completely. With enough problem solving experience, some generalized strategy, represented by a set of rules (equivalent to a problem solving *scheme*) should be induced which allows a domain expert to solve this problem via application of his/her strategic knowledge. We already tried out this idea using IGOR1 (Schmid & Wysotzki 2000). However, since IGOR1 was a two-step approach where examples had to be first rewritten into traces and afterwards recurrence detection was performed in these traces, this approach was restricted in its applicability. With IGOR2 we can reproduce the results of IGOR1 on the problems *clearblock* and *rocket* faster and without specific assumptions to preprocessing and furthermore can tackle more complex problem domains such as building a *tower* in the blocks-world domain.

The general idea of learning domain specific problem solving strategies is that first some small sample problems are solved by means of some planning or problem solving algorithm and that then a set of generalized rules are learned from this sample experience. This set of rules represents the competence to solve arbitrary problems in this domain. We illustrate the idea of our approach with the simple *clearblock* problem (see Figure 1). A problem consists of a set of blocks which are stacked in some arbitrary order. The problem solving goal is that one specific block – in our case *A* – should

Clearblock (4 examples, 0.036 sec)

```
ClearBlock(A, (B T), S) = S if A == B
ClearBlock(A, (B T), S) =
  ClearBlock(A, T, puttable(B, S)) if A /= B
```

Rocket (3 examples, 0.012 sec)

```
Rocket(nil, S) = move(S) .
Rocket((O Os), S) = unload(O, Rocket(Os, load(O, S)))
```

Tower (9 examples of towers with up to four blocks, 1.2 sec)

(additionally: 10 corresponding examples for Clear and IsTower predicate as background knowledge)

```
Tower(O, S) = S if IsTower(O, S)
Tower(O, S) =
  put(O, Sub1(O, S),
    Clear(O, Clear(Sub1(O, S),
      Tower(Sub1(O, S), S)))) if not(IsTower(O, S))
Sub1(s(O), S) = O .
```

Tower of Hanoi (3 examples, 0.076 sec)

```
Hanoi(O, Src, Aux, Dst, S) = move(O, Src, Dst, S)
Hanoi(s D, Src, Aux, Dst, S) =
  Hanoi(D, Aux, Src, Dst,
    move(s D, Src, Dst,
      Hanoi(D, Src, Dst, Aux, S)))
```

Figure 2: Learned Rules in Problem Solving Domains

be cleared such that no block is standing above it. We use predicates *clear(x)*, *on(x, y)*, and *ontable(x)* to represent problem states and goals. The only available operator is *puttable*: A block *x* can be put on the table if it is clear (no block is standing on it) and if it is not already on the table but on another block. Application of *puttable(x)* has the effect that block *x* is on the table and the side-effect that block *y* gets cleared if *on(x, y)* held before operator application. The negative effect is that *x* is no longer on *y* after application of *puttable*.

We use a PDDL-like notation for the problem domain and the problem descriptions. We defined four different problems of small size each with the same problem solving goal (*clear(A)*) but with different initial states: The most simple problem is the case where *A* is already clear. This problem is presented in two variants – *A* is on the table and *A* is on another block – to allow the induction of a *clearblock* rule for a block which is positioned in an arbitrary place in a stack. The third initial state is that *A* is covered by one block, the fourth that *A* is covered by two blocks. A planner might be presented with the problem domain – the *puttable* operator – and problem descriptions given in Figure 1.

The resulting action sequences can be obtained by any PDDL planner (Ghallab, Nau, & Traverso 2004) and rewritten to IGOR2 (i.e. MAUDE) syntax. When rewriting plans to MAUDE equations (see Figure 1) we give the goal, that is, the name of the block which is to be cleared, as first argument. The second argument represents the initial state, that is, the stack as list of blocks and *table* as bottom block. The third argument is a situation variable (McCarthy 1963;

Manna & Waldinger 1987; Schmid & Wysotzki 2000) representing the current state. Thereby plans can be interpreted as nested function applications and plan execution can be performed on the content of the situation variable. The right-hand sides of the example equations correspond to the action sequences which were constructed by a planner, rewritten as nested terms with situation variable S as second argument of the *puttable* operator. Currently, the transformation of plans to examples for IGOR2 is done “by hand”. For a fully automated interface from planning to inductive programming, a set of rewrite rules must be defined.

Given the action sequences for clearing a block up to three blocks deep in a stack as initial experience, IGOR2 generalizes a simple tail recursive rule system which represents the competence to clear a block which is situated in arbitrary depth in a stack (see Figure 2). That is, from now on, it is no longer necessary to search for a suitable action sequence to reach the *clearblock* goal. Instead, the generalized knowledge can be applied to produce the correct action sequence directly. Note, that IGOR2 automatically introduced the equal predicate to discern cases where A is on top of the stack from cases where A is situated farther below since these cases could not be discriminated by disjoint patterns on the left-hand sides of the rules.

A more complex problem domain is *rocket* (Velošo & Carbonell 1993). This domain was originally proposed to demonstrate the need of interleaving goals. The problem is to transport a number of objects from earth to moon where the rocket can only fly in one direction. That is, the problem cannot be solved by first solving the goal *at(o1, moon)* by loading it, moving it to the moon and then unloading it. Because with this strategy there is no possibility to transport further objects from earth to moon. The correct procedure is first to load all objects, then to fly to the moon and finally to unload the objects. IGOR2 learned this strategy from examples for zero to two objects (see Figure 2).

A most challenging problem domain which is still used as a benchmark for planning algorithms is *blocks-world*. A typical blocks-world problem is to build a *tower* of some blocks in some prespecified order. With evolutionary programming, an iterative solution procedure to this problem was found from 166 examples (Koza 1992). The found strategy was to first put all blocks on the table and then build the tower. This strategy is clearly not efficient and cognitively not very plausible. If, for example, the goal is a tower *on(A, B)*, *on(B, C)* and the current state is *on(C, B)*, *on(B, A)*, even a young child will first put C on the table and then *directly* put B on C and not put B on the table first. Another proposal to tackle this problem is to learn decision rules which at least in some situations can guide a planner to select the most suitable action (Martín & Geffner 2000). With the learned rules, 95.5% of 1000 test problems were solved for 5-block problems and 72.2% of 500 test problems were solved for 20-block problems. The generated plans, however, are about two

```

eq Tower(s s table,
        ((s s s table) (s table) table | ,
         (s s s table) (s s table) table | , nil)) =
  put(s s table, s table,
      put(s s s table, table,
          put(s s s s table, table,
              ((s s s s table) (s table) table | ,
               (s s s table) (s s table) table | , nil)))) .

```

Figure 3: One of the nine example equations for *tower*

steps longer than the optimal plans. In Figure 2 we present the rules IGOR2 generated from only nine example solutions. This rule system will always produce the optimal action sequence.

To illustrate how examples were presented to IGOR2 we show one example in Figure 3. The goal is to construct a tower for some predefined ordering of blocks. To represent this ordering, blocks are represented constructively as “successors” to the table with respect to the goal state ($|$ representing the empty tower). Therefore the top object of the to be constructed tower is given as first argument of the *tower* function. If the top object is *s s s table*, the goal is to construct a tower with three blocks with *s table* on the table, *s s table* on *s table* and *s s s table* on *s s table*. The second argument again is a situation variable which initially holds the initial state. In the example in Figure 3 *s s table* (we may call it block 2) shall be the top object and the initial state consists of two towers, namely block 4 on block 1 and block 3 on block 2. That is, the desired output is the plan to get the tower block 2 on block 1. Therefore blocks 1 and 2 have to be cleared, these are the both innermost puts, and finally block 2 has to be stacked on block 1 (block 1 lies on the table already), this is the out-most put.

In addition to the *tower* example, IGOR2 was given an auxiliary function *IsTower* as background knowledge. This predicate is true if the list of blocks presented to it are already in the desired order. Furthermore, we did not learn the *Clear* function used in *tower* but presented some examples as background knowledge.

Finally, the recursive solution to the Tower of Hanoi problem was generated by IGOR2 from three examples (see Figure 2). The input to IGOR2 is given in Figure 4.

For the discussed typical problem solving domains IGOR2 could infer the recursive generalizations very fast and from small example sets. The learned recursive rule systems represent the strategic knowledge to solve all problems of the respective domains with a minimal number of actions.

Reasoning

A classic work in the domain of reasoning is how humans induce rules in concept learning tasks (Bruner, Goodnow, & Austin 1956). Indeed, this work has inspired the first decision tree algorithms (Hunt, Marin,

```

eq Hanoi(0, Src, Aux, Dst, S) =
  move(0, Src, Dst, S) .
eq Hanoi(s 0, Src, Aux, Dst, S) =
  move(0, Aux, Dst,
    move(s 0, Src, Dst,
      move(0, Src, Aux, S))) .
eq Hanoi(s s 0, Src, Aux, Dst, S) =
  move(0, Src, Dst,
    move(s 0, Aux, Dst,
      move(0, Aux, Src,
        move(s s 0, Src, Dst,
          move(0, Dst, Aux,
            move(s 0, Src, Aux,
              move(0, Src, Dst, S))))))) .

```

Figure 4: Posing the *Tower of Hanoi* problem for IGOR2

Ancestor (9 examples, 10.1 sec)
 (and corresponding 4 examples for IsIn and Or)

```

Ancestor(X, Y, nil) = nilp .
Ancestor(X, Y, node(Z, L, R)) =
  IsIn(Y, node(Z, L, R)) if X == Z .
Ancestor(X, Y, node(Z, L, R)) =
  Ancestor(X, Y, L) Or Ancestor(X, Y, R) if X /= Z .

```

Corresponding to:

```

ancestor(x,y) = parent(x,y).
ancestor(x,y) = parent(x,z), ancestor(z,y).

```

```

isa(x,y) = directlink(x,y).
isa(x,y) = directlink(x,z), isa(z,y).

```

Figure 5: Learned Transitivity Rules

& Stone 1966). This work addressed simple conjunctive or more difficult to acquire disjunctive concepts. However, people are also able to acquire and correctly apply recursive concepts such as *ancestor*, *prime number*, *member of a list* and so on.

In the following, we will focus on the concept of *ancestor* which is often used as standard example in inductive logic programming (Lavrač & Džeroski 1994). The competence underlying the correct application of the *ancestor* concept, that is, correctly classifying a person as ancestor of some other person, in our opinion is the correct application of the transitivity relation in some partial ordering. We believe that if a person has grasped the concept of transitivity in one domain, such as *ancestor*, this person will also be able to correctly apply it in other, previously unknown domains. For example, such a person should be able to correctly infer *is-a* relations in some ontology. We plan to conduct a psychological experiment with children to strengthen this claim.

For simplicity of modeling, we used binary trees as domain model. For trees with arbitrary branching factor, the number of examples would have to be increased significantly. The transitivity rule learned by IGOR2 is given in Figure 5.

original grammar (in the very original grammar, *d n v* are non-terminals *D N V* which go to concrete words)

```

S -> NP VP
NP -> d n
VP -> v NP | v S

```

examples

```

fmod GENERATOR is
  *** types
  sorts Cat CList Depth .
  ops d n v : -> Cat [ctor] .
  op ! : -> CList [ctor] .
  op _ : Cat CList -> CList [ctor] .
  op 1 : -> Depth [ctor] .
  op s_ : Depth -> Depth [ctor] .
  *** target fun declaration
  op Sentence : Depth -> CList [metadata "induce"] .
  *** examples
  eq Sentence(1) = (d n v d n !) .
  eq Sentence(s 1) = (d n v d n v d n !) .
  eq Sentence(s s 1) = (d n v d n v d n v d n !) .

```

learned grammar rules (3 examples, 0.072 sec)

```

Sentence(1) = (d n v d n !)
Sentence(s N) = (d n v Sentence(N))

```

Figure 6: Learning a Phrase-Structure Grammar

Natural Language Processing

Finally, we come back to Chomsky's claim of an LAD. We presented IGOR2 with examples to learn a phrase-structure grammar. This problem is also addressed in grammar inference research (Sakakibara 1997). We avoided the problem of learning word-category associations and provided examples abstracted from concrete words (see Figure 6). This, in our opinion, is legitimate since word categories are learned before complex grammatical structures are acquired. There is empirical evidence that children first learn rather simple Pivot grammars where the basic word categories are systematically positioned before they are able to produce more complex grammatical structures (Braine 1963; Marcus 2001).

The abstract sentence structures correspond to sentences as (Covington 1994):

- 1: *The dog chased the cat.*
- 2: *The girl thought the dog chased the cat.*
- 3: *The butler said the girl thought the dog chased the cat.*
- 4: *The gardener claimed the butler said the girl thought the dog chased the cat.*

The recursive rules can generate sentences for an arbitrary depth which is given as parameter. IGOR2 can also learn more complex rules, for example allowing for conjunctions of noun phrases or verb phrases. In this case, a nested numerical parameter can be used to specify at which position conjunctions in which depth can be introduced. Alternatively, a parser could be learned. Note that the learned rules are simpler than the original grammar but fulfill the same functionality.

Conclusion

IGOR2 is a rather successful system for analytical inductive programming. Up to now we applied IGOR2 to typical programming problems (Hofmann, Kitzelmann, & Schmid 2008). In this paper we showed that analytical inductive programming is one possible approach to model a general cognitive rule acquisition device and we successfully applied IGOR2 to a range of prototypical problems from the domains of problem solving, reasoning, and natural language processing. Analytical inductive programming seems a highly suitable approach to model the human ability to extract generalized rules from example experience since it allows fast generalization from very small sets of only positive examples (Marcus 2001). We want to restrict IGOR2 to such domains where it is natural to provide positive examples only. Nevertheless, to transform IGOR2 from an inductive programming to an AGI system, in future we need to address the problem of noisy data as well as the problem of automatically transforming traces presented by other systems (a planner, a reasoner, a human teacher) into IGOR2 specifications.

References

- Anderson, J. R., and Lebière, C. 1998. *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.
- Anzai, Y., and Simon, H. 1979. The theory of learning by doing. *Psychological Review* 86:124–140.
- Biermann, A. W.; Guiho, G.; and Kodratoff, Y., eds. 1984. *Automatic Program Construction Techniques*. New York: Macmillan.
- Braine, M. 1963. On learning the grammatical order of words. *Psychological Review* 70:332–348.
- Bruner, J. S.; Goodnow, J. J.; and Austin, G. A. 1956. *A Study of Thinking*. New York: Wiley.
- Chomsky, N. 1959. Review of Skinner’s ‘Verbal Behavior’. *Language* 35:26–58.
- Chomsky, N. 1965. *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.
- Covington, M. A. 1994. *Natural Language Processing for Prolog Programmers*. Prentice Hall.
- Flener, P., and Partridge, D. 2001. Inductive programming. *Automated Software Engineering* 8(2):131–137.
- Flener, P. 1995. *Logic Program Synthesis from Incomplete Information*. Boston: Kluwer Academic Press.
- Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.
- Gold, E. 1967. Language identification in the limit. *Information and Control* 10:447–474.
- Hofmann, M.; Kitzelmann, E.; and Schmid, U. 2008. Analysis and evaluation of inductive programming systems in a higher-order framework. In Dengel, A. et al., eds., *KI 2008: Advances in Artificial Intelligence*, number 5243 in LNAI, 78–86. Berlin: Springer.
- Hunt, E.; Marin, J.; and Stone, P. J. 1966. *Experiments in Induction*. New York: Academic Press.
- Kitzelmann, E., and Schmid, U. 2006. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research* 7(Feb):429–454.
- Kitzelmann, E. 2008. Analytical inductive functional programming. In Hanus, M., ed., *Pre-Proceedings of LOPSTR 2008*, 166–180.
- Korf, R. E. 1985. Macro-operators: a weak method for learning. *Artificial Intelligence, 1985* 26:35–77.
- Koza, J. 1992. *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.
- Lavrač, N., and Džeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications*. London: Ellis Horwood.
- Levelt, W. 1976. *What became of LAD?* Lisse: Peter de Ridder Press.
- Manna, Z., and Waldinger, R. 1987. How to clear a block: a theory of plans. *Journal of Automated Reasoning* 3(4):343–378.
- Marcus, G. F. 2001. *The Algebraic Mind. Integrating Connectionism and Cognitive Science*. Bradford.
- Martín, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *Proc. KR 2000*, 667–677. San Francisco, CA: Morgan Kaufmann.
- McCarthy, J. 1963. Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California.
- Meyer, R. 1992. *Thinking, Problem Solving, Cognition, second edition*. Freeman.
- Minton, S. 1985. Selectively generalizing plans for problem-solving. In *Proc. IJCAI-85*, 596–599. San Francisco, CA: Morgan Kaufmann.
- Mitchell, T. M. 1997. *Machine Learning*. New York: McGraw-Hill.
- Olsson, R. 1995. Inductive functional programming using incremental program transformation. *Artificial Intelligence* 74(1):55–83.
- Quinlan, J., and Cameron-Jones, R. 1995. Induction of logic programs: FOIL and related systems. *New Generation Computing* 13(3-4):287–312.
- Rosenbloom, P. S., and Newell, A. 1986. The chunking of goal hierarchies: A generalized model of practice. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds., *Machine Learning - An Artificial Intelligence Approach*, vol. 2. Morgan Kaufmann. 247–288.
- Sakakibara, Y. 1997. Recent advances of grammatical inference. *Theoretical Computer Science* 185:15–45.
- Schmid, U., and Wysotzki, F. 2000. Applying inductive program synthesis to macro learning. In *Proc. AIPS 2000*, 371–378. AAAI Press.
- Shavlik, J. W. 1990. Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning* 5:39–70.
- Shell, P., and Carbonell, J. 1989. Towards a general framework for composing disjunctive and iterative macro-operators. In *Proc. IJCAI-89*. Morgan Kaufman.
- Summers, P. D. 1977. A methodology for LISP program construction from examples. *Journal ACM* 24(1):162–175.
- Veloso, M. M., and Carbonell, J. G. 1993. Derivational analogy in Prodigy: Automating case acquisition, storage, and utilization. *Machine Learning* 10:249–278.

Human and Machine Understanding Of Natural Language Character Strings

Peter G. Tripodes

pgtripodes@cs.com

Abstract

There is a great deal of variability in the way in which different language users understand a given natural language (NL) character string. This variability probably arises because of some combination of differences in language users' perceptions of its context-of-use (pragmatics), identity and mode of organization of its meaning bearing parts (syntax), and in the meanings assigned to those parts (semantics). This paper proposes a formalization of the syntax and semantics of NL character strings within a logical framework which is sufficiently flexible to represent the full breadth of possible ways of understanding NL character strings as influenced by different contexts-of use, beyond what can be represented in currently used predicate-logic-based frameworks. While the question of how language users understand NL character strings is ultimately a question in the psychology of language, it appears to us that, for the purposes of AGI, that question needs to be addressed within a logical framework which explicitly identifies the syntactic and semantic components that comprise that understanding, and which account – in formal terms – for differences in that understanding. Such a logical framework would provide a formal basis not only on which to address further psychological issues regarding human language understanding, but also for coherently imparting such understanding to machines.

Purpose

In this paper, I attempt to formalize the notion of what it means to understand NL character string as a sentence, and to describe a mechanism whereby particular sentential readings of given character strings induce particular patterns of deductive connections among them [1]. Allowing that great variability appears to exist among language users regarding the patterns of deductive connections which they perceive to hold among given NL character strings and to include non-normal, i.e., atypical, patterns as well as normal ones, the question arises regarding how to formalize sentential readings of character strings broadly enough to induce such a range of perceived patterns. Predicate logic and its variants such as resolution logic [2], [3], and its extensions such as Montague logic [4], do not appear sufficiently flexible for formalizing sentential readings broadly enough. In particular, while their respective mechanisms for inducing patterns of

deductive connections are both explicit and precise, as well as applicable to a wide range of NL character strings, the restrictions they impose on sentential readings are such as to be capable of inducing only a small range of normal patterns of deductive connections. Moreover their mechanisms are typically sequential rather than parallel, hence too slow in machine applications to meet envisioned AGI capabilities. Connectionist formalizations such as [5] appear even more restrictive in the kinds of deductive patterns which they induce and, while designed to use massively parallel mechanisms for inducing deductive patterns on NL character strings, their formalizations and proposed mechanisms have thus far been illustrated for a limited number of cases of the simplest types. In this paper we outline an alternative logic which appears minimally restrictive in the range of sentential readings it can represent and yet capable of supporting a massively parallel mechanism for inducing non-normal as well as normal patterns of deductive connections among them. There is not space to describe this mechanism in this paper, but it is described to some extent in [6].

Key Notions

Natural Language (NL) Character Strings. By a *natural language (NL) character string* I mean an expression of natural language stripped of any structure beyond the ordering and spacing of its characters.

Readings of NL Character Strings [1]. By a *reading of an NL character string* I mean a language user's way of understanding it which includes an intuitive conceptualization of its meaning bearing parts¹ (syntax), and intuitive conceptualization of the meanings that the language user associates with those meaning bearing parts (semantics, where both conceptualizations are conditioned by a perceived context-of-use (pragmatics)).

Sentential Readings of NL Character Strings. By a *sentential reading of an NL character string* I mean a reading of that character string as an assertion which can be judged as true or false. Sentential readings of a given NL character string can vary markedly among language users according to the way that they conceptualize its syntax and semantics and perceive its context-of-use².

Sentential Reading Assignments (SRAs) on Sets of Character Strings. By a *sentential reading assignment*

(SRA) on a set of NL character strings I mean an assignment of sentential readings to each character string in the set which *induces a pattern of deductive connections* among them.

Normal and Non-Normal SRAs on Sets of Character Strings. An SRA on a set of character strings is said to be *normal (non-normal)* to the degree that the patterns of deductive connections which it induces on the set is consistent (inconsistent) with language users' deductive intuitions relative to typical contexts-of-use.

Readings: Formally Considered

Formalizing Readings. A reading of an NL character string c is formalized as a pair $\langle \text{Syn}(c), \text{Sem}(c) \rangle$ consisting of a *syntactic representation* $\text{Syn}(c)$ of c , and a *semantic representation* $\text{Sem}(c)$ of c . We assume a syntactic representation language L and a universe of discourse. Due to space limitations we summarize the internal structures of $\text{Syn}(c)$ and $\text{Sem}(c)$, as indicated below. (A fuller account can be found in [6].)

Syntactic Representation $\text{Syn}(c)$ of a Sentential Reading of Character String c . There are three grammatical categories of expressions in the syntactic representation language L : *relation expressions*, *thing expressions*, and *modifier expressions*. In order to accommodate the differences which appear to exist in ways that different language users could understand given natural language word strings, we allow a very permissive grammar for L , called an “open grammar,” which is one in which the syntactic representation component $\text{Syn}(c)$ of a given (not necessarily sentential) reading of an NL character string c can be a relation expression, a thing expression, or a modifier expression³. For example, the syntactic representation component of the character string “love” could be a relation expression in one occurrence, a thing-expression in another occurrence, and a modifier-expression in a third. The syntactic representation component $\text{Syn}(c)$ of a sentential reading of a character string c is composed of an n -place relation expression r^n together with n thing expressions a_1, \dots, a_n which it relates, and together with three ordering functions p, q, t , on those n thing expressions (illustrated below). We schematically express $\text{Syn}(c)$ as $r^n (a_1, \dots, a_n)_{p,q,t}$, with the ordering functions as indicated. The relation expression r^n is, in turn, composed of a sequence of modifier expressions applied to a base (i.e., modifier-less) relation expression (e.g., “give”) together with m case expressions b_1, \dots, b_m (e.g., variously representing “agent”, “patient”, “recipient”, and so on), each of which identifies the semantic role of one (or more) of the n thing expressions a_1, \dots, a_n . Each a_i is, turn, composed of a sequence of modifier expressions applied to a base (i.e., modifier-less) thing expression.

Interpretations. The description of $\text{Sem}(c)$ makes essential reference to the notion of an *interpretation*, defined as follows: An *interpretation f on $\text{Syn}(c)$* is a function which assigns denotations to expressions in

$\text{Syn}(c)$ as follows: (i) f assigns to every n -place relation expression r^n in $\text{Syn}(c)$ a set $f[r^n]$ of n -tuples of elements of the universe of discourse; (ii) f assigns to every thing expression a_i in $\text{Syn}(c)$ a set $f[a_i]$ of subsets of the universe of discourse; and assigns to every modifier expression m in $\text{Syn}(c)$ a function $f[m]$ which assigns tuples and sets of subsets of elements of the universe of discourse to tuples and sets of subsets of the universe of discourse. By virtue of what the interpretation f assigns to the relation expressions, thing expressions, and modifier expressions in $\text{Syn}(c)$, f recursively assigns to $\text{Syn}(c)$ a set $f[\text{Syn}(c)]$ whose defining condition (called *the truth condition of $\text{Syn}(c)$ under f*) is a statement in the set theoretic meta-language of L which expresses in set theoretic terms the content of $\text{Syn}(c)$ relative to f . If this defining condition is a true statement of set theory, we say $\text{Syn}(c)$ is *true* under the interpretation f , and otherwise that $\text{Syn}(c)$ is *false* under the interpretation f . We restrict interpretations to *permissible* ones that render truth conditions comparable and computationally tractable. Roughly, the only way two permissible interpretations could differ would be in the denotations they assign to base relation expressions.

Denotation of the Syntactic Representation Component $\text{Syn}(c)$ of a Sentential Reading of c . The truth condition of the denotation of the syntactic representation component $\text{Syn}(c)$ under which $\text{Syn}(c)$ is true is regarded as describing an “event” or “state of affairs” to the effect that the denotations of the n thing expressions a_1, \dots, a_n stand in the relation denoted by r^n relative to three orderings p, q , and t on those thing expressions. The ordering p is called the *relative scope ordering* of a_1, \dots, a_n in $\text{Syn}(c)$, the ordering q is called the *relative place ordering* of a_1, \dots, a_n in $\text{Syn}(c)$, and the ordering t is called *the relative case ordering* of a_1, \dots, a_n in $\text{Syn}(c)$. The *relative scope ordering p* determines the scopes of the governing modifiers on each a_1, \dots, a_n . The *relative place ordering q* determines the order in which a_1, \dots, a_n are to be taken relative to the n -place relation denoted by r^n , in the sense that the thing expression a_i is to occupy the $p(i)$ th argument place of that relation. Finally, the *relative case ordering t* determines which of the cases b_1, \dots, b_m is associated with each of the thing expressions a_1, \dots, a_n , in the sense that, for each $i, 1 \leq i \leq n$, $t(a_i)$ is that case among b_1, \dots, b_m which applies to the thing expression a_i . For most sentences of English, case expressions are usually placed adjacent to the thing expression they govern, and both relative scope and relative place orderings are usually the “identity orderings”, that is, they coincide with the order of occurrence of the thing expressions they govern. But this is not the situation for all sentences of English, nor for sentences of many other languages. The syntactic structure of sentences must take into account each of these special orderings. For example, different *relative scope orderings p* correspond to the difference between “Every man loves some woman” and “Some woman is such that every man loves her”, different *relative place orderings q* correspond to the difference between “Every man loves some woman” and “Some woman loves every man,” and different *relative*

case orderings t correspond to the difference between “Every man loves some woman” and “Every man is loved by some woman”. We thus schematically express the syntactic representation component $Syn(c)$ of a sentential reading of a character string c as $r^n(a_1, \dots, a_n)_{p,q,t}$ where p is the relative scope ordering of c , q is the relative place ordering of c , and t is the relative case ordering of c . We refer to the elements of $U(f[a_i])$ for $1 \leq i \leq n$, as belonging to the i th domain of $f(r^n)$.

Semantic Representation $Sem(c)$ of a Sentential Reading of Character String c . The semantic representation component $Sem(c)$ of a sentential reading of c is the set of all denotations $f[Syn(c)]$, as f ranges over all permissible interpretations of $Syn(c)$ under which $Syn(c)$ is true. Thus the semantic representation $Sem(c)$ of a sentential reading of character string c expresses all states of affairs relative to permissible interpretations under which its syntactic representation $Syn(c)$ is true.

Sentential Reading Assignments (SRAs)

Sentential Reading Assignments. A *Sentential reading assignment (SRA)* on a set of C of NL character strings relative to a set C^\wedge of auxiliary NL character strings is an assignment of a sentential reading $\langle Syn(c), Sem(c) \rangle$ to every character string c in $C \cup C^\wedge$. An SRA induces a pattern of deductive connections on C as a relation R between subsets C' of C and elements c of C which holds just in case $syn(c)$ is true under every permissible interpretation f under which, for every c' in $C' \cup C^\wedge$, $Syn(c')$ is true under f . The set C^\wedge of auxiliary character strings can be regarded as a set of *assumptions* under the readings assigned to them, and which the language user perceives to be related to the readings assigned to the character strings in C , and which the language user regards as true. We will refer to C^\wedge as an *assumptive set for C* .

Normality of Patterns of Deductive Connections.

Normality of patterns of deductive connections is always relative to a context-of-use, which may be a typical one or an atypical one. When no reference is made to a context-of-use we regard the unreferenced context-of-use as being a *typical* one. A given pattern of deductive connections among given sentential readings of given character strings has a greater degree of normality than another pattern of deductive connections among sentential readings of those character strings *relative to a given context-of-use* if most language users would tend to regard the former pattern as more consistent with their deductive intuitions than the latter pattern relative to that particular context-of-use. A *normal pattern* of deductive connections among given character strings would be one which had a relatively high degree of normality relative to typical contexts-of-use and a *non-normal pattern* of deductive connections would be one which had a relatively low degree of normality relative to typical context-of-use.

Normality of SRAs. An SRA on $C \cup C^\wedge$ is regarded as *normal or non-normal, and as normal or non-normal to a*

given degree, relative to a given context-of-use-according as the pattern of deductive connections which that SRA induces is normal, non-normal, normal to that degree, or non-normal to that degree relative to that context-of-use, that is, according as the pattern of deductive connections which that SRA induces is consistent, inconsistent, consistent to that degree, or inconsistent to that degree with the deductive intuitions of language users relative to that context-of-use.

Normality of Readings. A reading $\langle Syn(c), Sem(c) \rangle$ of an NL character string c is normal relative to a given context-of-use, and is normal (i.e., without reference to a context-of-use) if it is normal relative to typical contexts-of-use.

Variability in SRAs among Language Users. Variability in SRAs on a set C of character strings relative to the assumptive set C^\wedge and relative to a context-of-use can derive from various sources: Variability in the character strings in C^\wedge among language users, variability in the syntactic representations $Syn(c)$ of the character strings in $C \cup C^\wedge$ assigned them, variability in the semantic representations of these character strings assigned them, and variability in the context-of-use relative to which the readings of character strings in $C \cup C^\wedge$ are made. Each of these three sources also allows for a large measure of possible variation in the normality of SRAs among language users.

Examples

Let C consist of the following character strings ⁴:

- (1) John loves Mary.
- (2) Mary is a person.
- (3) John loves a person
- (4) John does not love Mary.
- (5) Something loves Mary.
- (6) Mary is loved.
- (7) Johns knows Mary.
- (8) Mary is loved by John.
- (9) Love loves love.
- (10) Something loves love.

Some Normal Patterns of Deductive Connections among Character Strings (1) – (10). There are various possible patterns of deductive connections among sentential readings of (1) – (10) induced by SRAs which could reasonably be considered “normal” relative to some typical context-of-use. Some examples are given in Patterns (A), (B), and (C), below:

Pattern (A): This would be a pattern of deductive connections among (1) – (10) which included the following: (1) and (2) together deductively imply (3), but (1) alone does not; (1) deductively implies each of (5) and (6); (1) also deductively implies (7) if C^\wedge includes a character string which expresses, “Whoever loves Mary knows her” under a suitable sentential reading; (1) and (8) deductively imply each other, hence (8) deductively

implies each of (5) and (6) and, if C^\wedge includes a suitable sentential reading of character string “Whoever loves Mary knows her” then (8) deductively implies (7) as well; (1) does not deductively imply (4), nor does (4) deductively imply (1); and neither (9) nor (10) deductively imply or are implied by any subset of (1) – (7). This (partial) pattern of deductive connections would be induced by an SRA in which “John”, “Mary”, and “love” were assigned the same denotation in each of their occurrences in (1) – (10), and in which C^\wedge did *not* include a character string which expressed something like, “Mary is a person.” (Mary may have been a cat or some other non-person).⁴ The failure of (9) to imply (10) is due to the circumstance that we are considering normality relative to a *typical* context-of-use, and it is unlikely that there could be a typical context-of-use relative to which any sentential reading of (9) or (10) could be considered normal, that is, could enter into implications which were consistent with language users’ deductive intuitions. On the other hand, one can imagine certain atypical contexts-of-use relative to which (9) and (10) could be considered normal. See Pattern (E) below.

Pattern (B): Another normal pattern of deductive connections among sentential readings of the character strings (1) – (10) would be induced by the same SRA as induced Pattern (A) with the exception that (1) no longer deductively implies (7) inasmuch as C^\wedge no longer includes a sentential reading of “Whoever loves Mary knows her.” Pattern (B) here is a diminution of Pattern (A).

Pattern (C): A third normal pattern of deductive connections among sentential readings of these character strings would be that induced by the same SRA as induced Pattern (A) with the exception that (2) is not now in C but is in C^\wedge , so that this Pattern (C) now includes the additional implication that (1) alone implies (3). Pattern (C) is an augmentation of Pattern (A).

Some Normal and Non-normal Readings of Character String (1): There is not sufficient space in this short paper to give a detailed indication of the structure of normal readings of character strings (1) - (10) for an SRA that would induce the above indicated pattern of deductive connections among them. However, we can indicate the structure of one reading N1 of (1) that could be assigned to (1) as part of a normal SRA on (1) – (10) that would induce the above Pattern (A). We later indicate the structure of two *non-normal* readings, namely readings NN2 and NN3, which would also induce Pattern (A).

Normal Reading N1 <Syn₁(1), Sem₁(1)> of Character String (1): In Syn₁(1), “loves” is syntactically marked as a two term relation whose first term is syntactically marked as an agent position and whose second term is syntactically marked as a recipient position, and “John” and “Mary” are syntactically marked as individual entities which occupy the agent and recipient positions, respectively. In Sem₁(1), we have an interpretation f1 of (1) relative to the syntactic component Syn₁(1) of (1) which is a function which assigns, as denotation of “loves” a set of pairs of entities of the domain of discourse, and which assigns, as denotations of “John” and “Mary” individual entities of the

domain of discourse (as opposed to, say, classes or relations), and which is such that (1) is true under that interpretation f1 if and only if the pair which has the denotation of “John” as its first element and the denotation of “Mary” as its second element belongs to the denotation of “loves” that is, is one of the pairs in that denotation. Reading (1) of (1) would be a normal reading of the character string (1).

Non-Normal Reading NN2 <Syn₂(1), Sem₂(1)> of Character String (1): In Syn₂(1), “loves” is syntactically marked as a two term relation whose first term is syntactically marked as a recipient position and whose second term is syntactically marked as an agent position, i.e., the converse of the way “loves” is syntactically marked in Syn₁(1). In Sem₂(1) we have an interpretation f2 which is a function which assigns, as denotation of “loves” a set of pairs of entities of the domain of discourse, and which assigns, as denotations of “John” and “Mary” individual entities of the domain of discourse, and which is such that (1) is true under the interpretation f2 if and only if the pair which has the denotation of “John” as its first element and the denotation of “Mary” as its second element belongs to the denotation of “loves” that is, is one of the pairs in that denotation. Reading NN2 is not a normal reading of character string “Johns loves Mary” but would be a normal reading of the character string “Mary is loved by John.”

Non-Normal Reading NN3 <Syn₃(1), Sem₃(1)> of Character String (1): In Syn₃(1), “loves” is syntactically marked as a two term relation whose first term is syntactically marked as an agent position and whose second term is syntactically marked as recipient position, and “John” and “Mary” are syntactically marked as individual entities which respectively occupy the recipient and agent positions. In Sem₃(1), we have an interpretation f3 of (1) relative to the syntactic component Syn₂(1) of (1) which is a function which assigns, as denotation of “loves” a set of pairs of entities of the domain of discourse, and which assigns, as denotations of “John” and “Mary” individual entities of the domain discourse, and which is such that (1) is true under that interpretation f3 if and only if the pair which has the denotation of “Mary” as its first element and the denotation of “John” as its second element belongs to the denotation of “loves” assigned by f3. Reading NN3 of (1) is not a normal reading of the character string “John loves Mary” but would be a normal reading of the character string “Mary loves John.”

Some Non-Normal Patterns of Deductive Connections Among Character Strings (1) – (10): There are also various patterns of deductive connections among sentential readings of (1) – (10) induced by SRAs which could *not* reasonably be considered “normal” relative to some typical context-of-use. Some examples are given Patterns (D), (E), and (F) below.

Pattern (D): A fourth pattern of deductive connections among the character strings (1) – (10) is a non-normal one which would be induced by an SRA which assigned readings to (1) and (3) in which the character string “John”

in (1) received a different denotation than the denotation it received in (3) (which could occur in a context-of-use where there were two individuals, say, one an individual named “John” who loved “Mary”, and another named “John” who loved no one). This non-normal way of understanding (1) – (10) induces a pattern of deductive connections which is very different from any of the three earlier indicated patterns; in particular, (1) and (2) together no longer deductively imply (3).

Pattern (E): A fifth pattern of deductive connections among the character strings (1) – (10) is another non-normal one induced by an SRA such that: (i) the second occurrence of “love” in (9) and the second occurrence of “love” in (10) are each syntactically represented as a 2-place relation expression whose first term is syntactically marked as an agent position and whose second term is syntactically marked as a recipient position. The first and third occurrences of “love” in (9) are each syntactically marked as individual entities and respectively occupy the agent and recipient positions of the syntactic representation of the second occurrence of “loves” in (9). Similarly, the occurrence of “Something” in (10) and the second occurrence of “love” in (10) are each syntactically marked as individual entities, and respectively occupy the agent and recipient positions of the syntactic representation of the second occurrence of “loves” in (10). This non-normal reading of (9) and (10) induces a pattern of deductive connections among (1) – (10) which, unlike the case with Patterns (A) – (D) above, now includes the implication of (10) from (9).

Pattern (F): A sixth pattern of deductive connections among these character strings is another non-normal one induced by an SRA which assigns “love” a denotation in (8) which is opposite that assigned to “love” in (1) and (6), such as, for example, that its meaning in (1) and (6) is its usual meaning, while its meaning in (8) is an ironic one, i.e., to mean “hate”. This is non-normal way of understanding (1) – (10) induces a pattern of deductive connections which is very different from any of the four earlier indicated patterns; in particular, (1) and (8) no longer deductively imply each other, and (8) no longer deductively implies (6). Indeed, we now have the bizarre implication of (1) from (4) and (4) from (1).

Relative Degree of Normality of Above Sample Patterns of Deductive Readings. Recalling that the degree of normality of a given pattern of deductive connections is the degree to which language users would tend to regard the pattern of deductive consequences as consistent with their deductive intuitions relative to a typical context-of-use, we would order the above six patterns (A) - (F) as being in decreasing order of normality.

Internal Structure of Sentential Readings

Internal Structure of the Syntactic Component of a Reading. The syntactic component $Syn(c)$ of a reading of

a character string c describes the underlying syntactic structure of c as a pattern of interconnections of its meaning-bearing parts. The minimal meaning-bearing parts of a syntactic representation of a character string are called *representational morphemes*. The syntactic representation of that character string is recursively built out of representational morphemes into a syntactic representation $Syn(c)$ of the entire character string.

Internal Structure of the Semantic Component of a Reading. The semantic component $Sem(c)$ of a reading of a character string c assigns a set-theoretical meaning to every meaning-bearing part identifies in the syntactic representation component of that reading, and thereby interprets that syntactic representation, proceeding from its (syntactically) smaller meaning-bearing parts and, by a recursive process, ultimately to the full pattern of interconnections of those meaning-bearing parts. The semantic component is specified in *semantic axioms*, which state the set theoretical meanings to be assigned to meaning-bearing parts.

Parts: Of Character Strings and of Syntactic Representations. Parts of character strings will be distinguished from parts of syntactic representations of character strings. The notion of “part” as it applies to character strings is to be understood in the sense that the sequence of letter and blanks comprising the part in question is a subsequence of the sequence of letters and blanks comprising the containing character string, and is not intended to be semantically interpretable. On the other hand, the notion of “part” as it applies to syntactic representations of character strings, is intended to be semantically interpretable; that is, “part” in this latter sense means “interpretable part”, whereas, in the case of character strings, it does not.

Implicitly and Explicitly Realized NL Morphemes. As remarked earlier, a deductive reading of a character string specifies a system of syntactically and semantically inter-related representational morphemes. Consistent with standard linguistic usage, I regard the notion of a natural language morpheme as a theoretical construct, i.e., as an abstract entity that is “realized” in a given character string in one of two ways: (a) explicitly, indicated in part by and corresponding to a specific part of that character string called a “morph”; (b) implicitly, indicated solely by global relations among the parts of that character string, involving factors such as order of occurrence, juxtaposition, intonation patterns (if oral), perceived grammatical and semantic relationships among character string parts, etc. A natural language morpheme that is explicitly realized in a part of (i.e.: as a morph occurring in) a given character string is also said to be explicitly marked in that character string by that part (i.e., by that morph). A natural language morpheme that is implicitly realized in a given character string by certain global relations among its parts is said to be implicitly marked in that character string by those relations.

Logical and Lexical NL Morphemes. The intended distinction between logical and lexical natural language

morphemes is an intuitive semantic one: roughly, a lexical natural language morpheme is one which intuitively denotes some entity, relation, or characteristic of an entity or relation, such as “boy”, “walks”, “hits”, “tall”, “slowly”, etc; whereas a logical natural language morpheme is one that intuitively denotes some way of operating on what lexical natural language morphemes denote, and expressed by character strings such as “all”, “and”, “not”, “many”, “after”, etc. We distinguish the notion of an NL morpheme from that of a representational morpheme, which is an actual expression of a syntactic representation of a character string which occurs as an explicit part of that syntactic representation.

Morphemic Base Assumption. We assume that a language user’s intuitive judgments regarding the degree of normality of a given pattern of deductive connections among the character strings in a given set C of character strings derive from this or her intuitive judgments regarding semantic interconnections among the logical natural language morphemes realized in the character strings of $C \cup C^A$, and regarding semantic interconnections among the lexical natural language morphemes realized in the character strings of $C \cup C^A$.

Semantic Interconnections among Logical Natural Language Morphemes Realized in (1) – (10). Applying the Morphemic Base Assumption to the pattern (A) of deductive connections, we would conclude that the particular deductive connections of (A) derived ultimately from intuitive judgments regarding semantic interconnections among the logical natural language morphemes realized in the character strings (1) – (10), which included in part, the explicit logical natural language morphemes “not” and “is”, as well as, various implicit logical natural language morphemes. We are suggesting, then, that the typical English speaker who understood the meanings of these logical natural language morphemes would assent to the above pattern of deductive connections even if he did not understand the meanings of the lexical natural language morphemes “John”, “Mary”, “love”, and “person” occurring there.

Semantic Interconnections Among Lexical Natural Language Morphemes Realized in (1) – (10). On the other hand, an English speaker’s intuitive judgment that (1) deductively implied (7) would derive *both* from his intuitive judgments regarding semantic interconnections among the logical natural language morphemes occurring in character strings (1) and (7), *and* from his intuitive judgments regarding the semantic interconnections between the lexical natural language morphemes “loves” and “knows” (such as, for example, that loving a person meant, in part, knowing that person).

strings, or of the patterns of deductive connections they induce among them. Rather, our analysis is forwarded as a “competence” model of their role in language users’ understanding of those character strings and how that understanding induces perceived patterns of deductive connections among them.

2. The notion of context-of-use is treated in this paper as a primitive notion to mean something like *the real-world situation in which given character strings are produced*.
3. We refer to this type of grammar as an “open grammar” inasmuch as the grammatical category in which a given character string is syntactically represented is not fixed, but can vary from one occurrence to another.
4. For simplicity we express character strings in their ordinary appearance as sentences rather than as a concatenation of individual alphabetic symbols and spaces.

References

- [1] Tripodes, P.G., *A Theory of Readings*. Unpublished Manuscript.
- [2] Wos, L., *Automated Reasoning: 33 Basic Research Problems*, Prentiss-Hall, Englewood Cliffs, New Jersey, 1988.
- [3] Bachmair, L. & Ganzinger, H., “Resolution Theorem Proving,” in A. Robinson & A. Voronkov, eds. *Handbook of Automated Reasoning. Vol I*. Elsevier Science, Amsterdam, chapter 1, pp 21-97. 2001.
- [4] R. Montague, English as a Formal Language, in R. Thomason, ed., *Formal Philosophy. Selected Papers of Richard Montague*, Yale University Press, New Haven. Pp 188-221. 1974.
- [5] Shastri, L., & Ajjanagadde, V., “From simple association to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony,” in *Behavioral and Brain Sciences*, 16, 417-494. 1993.
- [6] Tripodes, P.G., “Real time machine deduction and AGI,” in P. Wang, B. Goertzel, & S. Franklin, eds. *Artificial General Intelligence*. Amsterdam, IOS Press. 2008.

Endnotes

1. This is not to say that the language user is explicitly conscious of any of these components, or how they condition his or her understanding of given character

Embodiment: Does a laptop have a body?

Pei Wang

Temple University, Philadelphia, USA

<http://www.cis.temple.edu/~pwang/>

Abstract

This paper analyzes the different understandings of “embodiment”. It argues that the issue is not on the hardware a system is implemented in (that is, robot or conventional computer), but on the relation between the system and its working environment. Using an AGI system NARS as an example, the paper shows that the problem of disembodiment can be solved in a symbolic system implemented in a conventional computer, as far as the system makes realistic assumptions about the environment, and adapts to its experience.

This paper starts by briefly summarizing the appeal for embodiment, then it analyzes the related concepts, identifies some misconceptions, and suggests a solution, in the context of AGI research.

The Appeal for Embodiment

In the last two decades, there have been repeated appeals for *embodiment*, both in AI (Brooks, 1991a; Brooks, 1991b; Pfeifer and Scheier, 1999) and CogSci (Barsalou, 1999; Lakoff and Johnson, 1998). In AI, this movement argues that many problems in the field can be solved if people move their working platform from conventional computer to robot; in CogSci, this movement argues that human cognition is deeply based on human sensorimotor mechanism.

In general, “embodiment” calls people’s attention to the “body” of the system, though like all theoretical concepts, the notion of “embodiment” has many different interpretations and usages. This paper does not attempt to provide a survey to the field, which can be found in (Anderson, 2003), but to concentrate on the central issue of the debate, as well as its relevance to AGI research.

The stress on the importance of body clearly distinguishes this new movement from the traditions in AI and CogSci. In its history of half a century, a large part of AI research has been guided by the “Physical Symbol Hypothesis” (Newell and Simon, 1976), which asks AI systems to build internal representation of the environment, by using “symbols” to represent objects and relations in the outside world. Various formal operations, typically searching and reasoning, can be carried out on such a symbolic representation, so as

Copyright © 2008, The Second Conference on Artificial General Intelligence (AGI-09.org). All rights reserved.

to solve the corresponding problems in the world. Representative projects of this tradition include GPS (Newell and Simon, 1963) and CYC (Lenat, 1995). Except serving as a physical container of the system, the body of such a system has little to do with the content and behavior of the system. Even in robotics, where the role of body cannot be ignored, the traditional approach works in a Sense-Model-Plan-Act (SMPA) framework, in which the robot acts according to an internal “world model”, a symbolic representation of the world (Nilsson, 1984; Brooks, 1991a).

As a reaction to the problems in this tradition, the ‘embodied’ approach criticizes the traditional approach as being ‘disembodied’, and emphasizes the role of sensorimotor experience to intelligence and cognition. Brooks’ behavior-based robots have no representation of the world or the goal of the system, since “the world is its own best model” (Brooks, 1991a), so the actions of the robot are directly triggered by corresponding sensations. According to Brooks, “In order to really test ideas of intelligence it is important to build complete agents which operate in dynamic environments using real sensors. Internal world models which are complete representations of the external environment, besides being impossible to obtain, are not at all necessary for agents to act in a competent manner.” (Brooks, 1991a)

Therefore, as far as the current discussion is concerned, ‘embodiments’ means the following two requirements:

Working in real world: “Only an embodied intelligent agent is fully validated as one that can deal with the real world” (Brooks, 1991a), since it is more realistic by taking the complex, uncertain, real-time, and dynamic nature of the world into consideration (Brooks, 1991a; Pfeifer and Scheier, 1999).

Having grounded meaning: “Only through a physical grounding can any internal symbolic or other system find a place to bottom out, and give ‘meaning’ to the processing going on within the system” (Brooks, 1991a), which supports content-sensitive processing (Anderson, 2003), and solves the “symbol grounding” problem (Harndad, 1990).

Though this approach has achieved remarkable success in robotics, it still has difficulty in learning skills and handling complicated goals (Anderson, 2003; Brooks, 1991a; Murphy, 2000).

Embodiment and Robot

Though the embodiment school has contributed good ideas to AI research, it also has caused some misconceptions.

In the context of AI, it is often suggested, explicitly or implicitly, that only robotic systems are “embodied”, while systems implemented in conventional computer are “disembodied”. This opinion is problematic. As long as a system is implemented in a computer, it has a body — the hardware of the computer. Even though sometimes the system does not have a piece of dedicated hardware, it still stays in a body, the physical devices that carry out the corresponding operations. For instance, a laptop computer obviously has a body, on which all of its software run.

Though the above statement sounds trivially true, some people may reject it by saying that in this context, a “body” means something that have *real* sensorimotor mechanism, as suggested in (Brooks, 1991a; Brooks, 1991b; Pfeifer and Scheier, 1999). After all, robots have sensors and actuators, while laptop computers do not, right? Though this is indeed how we describe these two types of system in everyday language, this casual distinction does not make a fundamental difference. As long as a system interacts with its environment, it has sensors and actuators, that is, input and output devices. For a laptop computer, its sensors include keyboard and touch-pad, and its actuators include screen and speaker, while the network connection serves as both. These devices are different from the ones of robots in the type, range, and granularity of signals accepted/produced, but they are no less “real” as sensorimotor devices. Similarly, computer input and output operations can be considered as “perception” and “action”, in the broad sense of the words.

How about the claim that only robots interact with the “real” world? Once again, it is a misleading claim, because the environment of other (non-robotic) systems are no less “real” — at least the human users who use the computer via the input/output decides are as real as the floor the robots run on! After all, to a robot, the “world” it can perceive is still limited by the function of its sensorimotor devices.

A related distinction is between “physical agents” (like robots) and “virtual agents” (like chatbots). They are clearly different, but the difference is not that the latter does not run in a physical device or does not interact with its environment via physical processes — the electric currents carrying the input/output signals for a chatbot are as “physical” as the lights going into the visual sensor of a robot.

The above misconceptions usually come from the opinion that though an ordinary computer has a hardware body and does interact with its environment, the interaction is *symbolic* and *abstract*, and therefore is fundamentally different from the *physical* and *concrete* interaction between a robot and its environment. However, this opinion is an misunderstanding itself.

In the context of the current discussion, there is no such a thing as “purely symbolic and abstract interaction”. Every interaction between every computer and its environment is carried out by some concrete physical process, such as pressure on a key, movement on a touch-pad, light change on a monitor, electronic flow in a cable, and so on. What is ‘symbolic’ and ‘abstract’ is not such a process itself, but

the traditional *description* about it, where the details of the underlying physical process is completely omitted. On this topic, the difference between a computer and a robot is not really in the system themselves, but in the usual ways to treat them.

Now some reader may think that this paper is another defense of the symbolic AI school against the embodiment school, like (Vera and Simon, 1993), since it dismisses the embodiment approach by saying that what it demands are already there all the time. This is not the case. What this paper wants to do is actually to *strengthen* the embodiment argument, by rejecting certain common misunderstandings and focusing on the genuine issues.

Though every computer system has a body, and does interact with its environment, there is indeed something special about robots: a robot directly interacts with the world without human involvement, while the other systems mainly interact with human users. As argued above, here the difference is not whether the world or the sensor/actuator is “real”. Instead, it is that the human users are *tolerant* to the system, while the non-human part of world is not. In robotics, “There is no room for cheating” (Brooks, 1991a) — a robot usually has to face various kinds of uncertainty, and to make real-time response. On the contrary, in other AI systems there are various assumptions on what types of input are acceptable, and on how much time-space resources are required for a certain computation, that the users have got used to gratify.

Therefore, the “real world” requirement is really about whether the assumptions on environment are “realistic”, by keeping its complexity, uncertainty, and resource-restriction. Under this interpretation, “be real” is applicable not only to robots, but also to almost all AI systems, since in most realistic situations, the system has *insufficient knowledge* (various uncertainties) and *insufficient resources* (time-space restriction), with respect to the problems. It is only that traditional AI systems have the option to “cheat” by only accepting an “idealized” version of a problem, while robotic systems usually do not have such an option.

The traditional symbolic AI systems are indeed *disembodied*. Though every AI system has a body (with real sensors and actuators) and interacts with the real world, in traditional AI systems these factors are all *ignored*. Especially, in the internal representation of the world in such a system, the meaning of a symbol is determined by its *denotation* in the world, and therefore have little to do with the system’s sensorimotor experience, as well as the bias and restriction imposed by the system’s body. For example, if the meaning of symbol “Garfield” is nothing but a cat existing in the world, then whether a system using the symbol can see or touch the cat does not matter. The system does not even need to have a body (even though it does have one) for the symbol to have this meaning. This is not how meaning should be handled in intelligent systems.

Based on the above analysis, the two central requirements of embodiment can be revised as the following:

Working in real world: An intelligent system should be designed to handle various types of uncertainty, and to work in real time.

Having grounded meaning: In an intelligent system, the meaning of symbols should be determined by the system's experience, and be sensitive to the current context.

This version of embodiment is different from the Brooks-Pfeifer version, in that it does not insist on using robots to do AI (though of course it allows that as one possibility). Here "embodiment" no longer means "to give the system a body", but "to take the body into account". According to this opinion, as long as a system is implemented, it has a body; as long as it has input/output, it has perception/action. For the current discussion, what matters is not the physical properties of the system's body and input/output devices, but the *experience* they provide to the system. Whether a system is "embodied" is determined by whether the system is adaptive to its experience, as well as whether there are unrealistic constraints on its experience.

Many traditional AI systems are disembodied, not because they are not implemented as robots, but because the symbols in them are understood as labels of objects in the world (therefore are experience-independent), and there are strong constraints on what the system can experience. For example, the users should not feed the system inconsistent knowledge, or ask questions beyond its knowledge scope. When these events happen, the system either refuses to work or simply crashes, and the blame falls on the user, since the system is not designed to deal with these situations.

Embodiment in NARS

To show the possibility of achieving embodiment (as interpreted above) without using a robot, an AGI project, NARS, is briefly introduced. Limited by the paper length, here only the basic ideas are described, with reference to detailed descriptions in other publications.

NARS is a general-purpose AI system designed according to the theory that "intelligence" means "adapting to environment and working with insufficient knowledge and resources" (Wang, 2006). Since the system is designed in the reasoning system framework, with a formal language and a set of formal inference rules, at the first glance it looks just like a "disembodied" traditional AI system, though this illusion will be removed, hopefully, by the following description and discussion.

At the current stage of development, the interaction between NARS and its environment happens as input or output sentences of the system, expressed in a formal language. A sentence can represent a *judgment*, a *question*, or a *goal*. As input, a judgment provides the system new knowledge to remember, a question requests the system to find an answer according to available knowledge, and a goal demands the system to achieve it by carrying out some operations. As output, a judgment provides an answer to a question or a message to other systems, a question or a goal asks help from other systems in the environment to answer or achieve it. Over a period of time, the stream of input sentences is the system's *experience*, and the stream of output sentences is the system's *behavior*.

Since NARS assumes insufficient knowledge, there is no constraint on the *content* of its experience. New knowledge

may conflict with previous knowledge, no knowledge is absolutely certain, and questions and goals may be beyond the current knowledge scope. Consequently, the system cannot guarantee the absolute correctness of its conclusions, and its predictions may turn out to be wrong. Instead, the validity of its inference is justified by the principle of adaptation, that is, the conclusion has the highest evidential support (among the alternatives), according to the system's experience.

Since NARS assumes insufficient resources, the system is open all the time to new input, and processes them in real-time. So the system cannot simply process every problem exhaustively by taking all possibilities into consideration. Also, it has to manage its storage space, by removing some data whenever there is a shortage of space. Consequently, the system cannot guarantee the absolute optimum of its conclusions, and any of them may be revised by new information or further consideration. Instead, the validity of its strategy is also justified by the principle of adaptation, that is, the resources are allocated to various activities to achieve the highest overall efficiency (among the alternatives), according to the system's experience.

The requirement of embodiment follows from the above assumption and principle. The assumption on the insufficiency in knowledge and resources puts the system in a realistic environment, where it has to deal with various types of uncertainty, and handle tasks in real-time. The system does not have the knowledge and resources to build a model of the world, then to act accordingly. Instead, its knowledge is nothing but summary of its past experience, which guides the system to deal with the present, and be prepared for the future. There is an internal representation in the system, though it is not a *representation of the world*, but a *representation of the experience of the system*, after summarization and organization. The symbols in the representation have different meaning to the system, not because they refer to different objects in the world, but because they have played different roles in the system's experience.

Concretely, the meaning of a concept in NARS is determined by its experienced relation with other concepts. That is to say, what "Garfield" means to (an implementation of) NARS is not decided by an object labeled by that term, but by what the system knows about "Garfield". Given the resources restriction, each time the concept "Garfield" is used in the system, only part of its relations are taken into consideration. Therefore, what the term means to the system may (more or less) change from time to time, and from situation to situation, though not arbitrarily.

The details of this "experience-grounded semantics" is explained and discussed in (Wang, 2005; Wang, 2006). Though many people have argued for the importance of experience in intelligence and cognition, no other work has explicitly and formally defined the central semantic notions 'meaning' and 'truth-value' as functions of the system's experience, and specified the details in their computational implementation.

How about the *sensorimotor* aspects of the meaning? In a broad sense, all knowledge (directly or indirectly) comes from the system's experience, which initially comes through sensorimotor devices of the system. If we use the term to re-

fer to non-linguistic experience, then in NARS it is possible to link “Garfield” to related visual images and operation sequences, so as to enrich its meaning. However, it is important to understand that both linguistic experience and non-linguistic experience are special cases of experience, and the latter is not more “real” than the former.

In the previous discussions, many people implicitly suppose that linguistic experience is nothing but “Dictionary-Go-Round” (Harnad, 1990) or “Chinese Room” (Searle, 1980), and only non-linguistic sensorimotor experience can give symbols meaning. This is a misconception coming from traditional semantics, which determines meaning by referred object, so that an image of the object seems to be closer to the “real thing” than a verbal description. NARS’ experience in Chinese is different from the content of a Chinese-Chinese dictionary, because a dictionary is static, while the experience of a system extends in time, in which the system gets feedback from its environment as consequences of its actions, i.e., output sentences in Chinese. To the system, its experience contains all the information it can get from the environment. Therefore, the system’s processing is not “purely formal” in the sense that the meaning of the symbols can be assigned arbitrarily by an outside observer. Instead, to the system, the relations among the symbols are what give them meaning. A more detailed discussion on this misconception can be found in (Wang, 2007), and will not be repeated here.

In summary, NARS satisfies the two requirements of embodiment introduced previously:

Working in real world: This requirement is satisfied by the assumption of insufficiency in knowledge and resources.

Having grounded meaning: This requirement is satisfied by the experience-grounded semantics.

Difference in Embodiment

Of course, to say an implementation of NARS running in a laptop computer is “already embodied”, it does not mean that it is embodied in exactly the same form as a human mind operating in a human body. However, here the difference is not between “disembodied” and “embodied”, but between different forms of embodiment.

As explained previously, every concrete system interacts with its environment in one or multiple modalities. For a human being, major modalities include vision, audition, tactile, etc.; for a robot, they include some human-like ones, but also non-human modalities like ultrasonic; for an ordinary computer, they directly communicate electronically, and also can have optional modalities like tactile (keyboard and various pointing devices), audition (microphone), vision (camera), though they are not used in the same form as in a human body.

In each modality, the system’s experience is constructed from certain “primes” or “atoms” that is the smallest units the system can recognize and distinguish. The system’s processing of its experience is usually carried out on their compound “patterns” that are much larger in scale, though short in details. If the patterns are further abstracted, they can

even become modality-independent “symbols”. This is the usual level of description for linguistic experience, where the original modality of a pattern, with all of its modality-specific details, is ignored in the processing of the message. However, this treatment does not necessarily make the system disembodied, because the symbols still comes from the system’s experience, and can be processed in an experience-dependent manner.

What makes the traditional symbolic AI system disembodied is that the symbols are not only abstracted to become *modality-independent*, but also *experience-independent*, in the sense that the system’s processing of the symbol is fully determined by the system’s *design*, and have little to do with its *history*. In this way, the system’s body becomes completely irrelevant, even though literally speaking the system exists in a body all the time.

On the contrary, linguistic experience does not exclude the body from the picture. For a system that only interact with its environment in a language, its experience is linguistic and amodal, in the sense that the relevant modality is not explicitly marked in the description of the system’s experience. However, what experience the system can get is still partially determined by the modality that carries out the interaction, and therefore, by the body of the system. As far as the system’s behavior is *experience-dependent*, it is also *body-dependent*, or *embodied*.

Different bodies give a system different experiences and behaviors, because they usually have different sensors and operators, as well as different sensitivity and efficiency on different patterns in the experience and the behavior. Consequently, even when they are put into the same environment, they will have different experience, and therefore different thoughts and behaviors. According to experience-grounded semantics, the meaning of a concept depends on the system’s experience on the concept, as well as on the possible operations related to the concept, so any change in the system’s body will more or less change the system’s mind.

For example, at the current stage, the experience of NARS is purely linguistic, so the meaning of a concept like ‘Garfield’ only depends on its experienced relations with other concepts, like ‘cat’, ‘cartoon character’, ‘comic strip’, ‘lazy’, and so on. In the future, if the system’s experience is extended to include visual and tactile components, the meaning of ‘Garfield’ will include additional relations with patterns in those modalities, and therefore become closer to the meaning of ‘Garfield’ in a typical human mind. Therefore, NARS implemented in a laptop and NARS implemented in a robot will probably associate different meaning to the same term, even though these meanings may have overlap.

However, it is wrong to say that the concept of ‘Garfield’ is *meaningful* or *grounded* if and only if it is used by a robot. There are two common misconceptions on this issue. One is to only take *sensorimotor* experience as real, and refuse to accept *linguistic* experience; and the other is to take *human* experience as the standard to judge the intelligence of other systems. As argued previously, every linguistic experience must be based on some sensorimotor experience, and though the latter is omitted in the description, it does not make the former less ‘real’ in any sense. Though “behave according

to experience” can be argued to be a necessary condition of being intelligent (Wang, 2006), to insist the experience must be equal to or similar to *human* experience leads to an anthropocentric understanding of intelligence, and will greatly limit our ability to build, and even to image, other (non-human) forms of intelligence (Wang, 2008).

In the current AI field, very few research project aims at accurately duplicating human behaviors, that is, passing the Turing Test. It is not only because of the difficulty of the test, but also because it is not a *necessary condition* for being intelligent, which was acknowledged by Turing himself (Turing, 1950), though often forgot by people talking about that article. Even so, many outside people still taking “passing the Turing Test” as the ultimate goal, or even the definition, of AI. This is why the proponents of the embodied view of human cognition often have negative view on the possibility of AI (Barsalou, 1999; Lakoff and Johnson, 1998). After identifying the fundamental impacts of human sensorimotor experience on human concepts, they see this as counter evidence for a computer to form the same concepts, without a human body. Though this conclusion is correct, it does not mean AI is impossible, unless “artificial intelligence” is interpreted as “artificial human intelligence”, that is, the system not only follows the general principles associated with ‘intelligence’, but also have the same concepts as a normal human being.

Because of the fundamental difference between human experience and the experience an AI system can have, the meaning of a word like ‘Garfield’ may never be the same in these two types of system. If AI aims at an accurate duplication of the contents of human categories, then we may never get there, but if it only aims at relating the contents of categories and the experience of the system in the same way as in the human mind, then it is quite possible, and that is what NARS attempts to achieve, among other things.

When people use the same concept with different meanings, it is usually due to their different *experience*, rather than their different *intelligence*. If this is the case, then how can we expect AI systems to agree with us on the meaning of a word (such as “meaning”, or “intelligence”), when we cannot agree on it among ourselves? We cannot deny the intelligence of a computer system just because it uses some of our words in a way that is not exactly like human beings.

Of course, for many practical reasons, it is highly desired for the concepts in an AI system to have similar meaning as in a typical human mind. In those situations, it becomes necessary to simulate human experience, both linguistic and non-linguistic. For the latter, we can use robots with human-like sensors and actuators, or simulated agents in virtual worlds (Bringsjord et al., 2008; Goertzel et al., 2008). However, we should understand that in principle, we can build fully intelligent systems, which, when given experience that is very different from human experience, may use some human words in non-human ways. After all, “to ground symbols in experience” does not mean “to ground symbols in *human* experience”. The former is required for being intelligent, while the latter is optional for being intelligent, though maybe desired for certain practical purposes.

Conclusion

Embodiment is the request for a system to be designed to work in a realistic environment, where its knowledge, categories, and behavior all depend on its experience, and therefore can be analyzed by considering the interaction between the system’s body and the environment.

The traditional symbolic AI systems are disembodied, mainly because of their unrealistic assumptions about the environment, and their experience-independent treatment of symbols, categories, and knowledge.

Though robotic research makes great contribution to AI, being a robot is neither a sufficient nor a necessary condition for embodiment. When proposed as a requirement for all AI systems, the requirement of embodiment should not be interpreted as “to give the system a body”, or “to give the system a human-like body”, but as “to make the system to behave according to its experience”. Here “experience” includes linguistic experience, as a high-level description of certain underlying sensorimotor activity.

The practice in NARS shows that embodiment can be achieved by a system where realistic assumption about the environment is made, such as “the system has insufficient knowledge/resources with respect to the problems the environment raises”, and the symbols in the system can get their meaning from the experience of the system, by using an experience-grounded semantics.

Though a laptop computer always has a body, a system running in this laptop can be either “embodied”, or “disembodied”, depending on whether the system behaves according to its experience.

Different bodies give systems different possible experiences and behaviors, which in turn lead to different knowledge and categories. However, here the difference is not between intelligent systems and non-intelligent ones, but among different types of intelligent systems.

Given the fundamental difference in hardware and experience, we should not expect AI systems to have human concepts and behaviors, but the same *relationship* between their experience and behavior, that is, being adaptive, and working with insufficient knowledge and resources.

Acknowledgment

The author benefits from a related discussion with Ben Goertzel and some others on the AGI mailing list, as well as from the comments of the anonymous reviewers.

References

- Anderson, M. L. (2003). Embodied cognition: A field guide. *Artificial Intelligence*, 149(1):91–130.
- Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, 22:577–609.
- Bringsjord, S., Shilliday, A., Taylor, J., Werner, D., Clark, M., Charpentie, E., and Bringsjord, A. (2008). Toward logic-based cognitively robust synthetic characters in digital environments. In *Artificial General Intelligence 2008*, pages 87–98, Amsterdam. IOS Press.

- Brooks, R. A. (1991a). Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, San Mateo, CA. Morgan Kaufmann.
- Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.
- Goertzel, B., Pennachin, C., Geissweiller, N., Looks, M., Senna, A., Silva, W., Heljakka, A., and Lopes, C. (2008). An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in Second Life. In *Artificial General Intelligence 2008*, pages 161–175, Amsterdam. IOS Press.
- Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42:335–346.
- Lakoff, G. and Johnson, M. (1998). *Philosophy in the Flesh: The Embodied Mind and Its Challenge to Western Thought*. Basic Books, New York.
- Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.
- Murphy, R. R. (2000). *An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents)*. MIT Press, Cambridge, Massachusetts.
- Newell, A. and Simon, H. A. (1963). GPS, a program that simulates human thought. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York.
- Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126.
- Nilsson, N. J. (1984). Shakey the robot. Technical Report 323, SRI AI Center, Menlo Park, CA.
- Pfeifer, R. and Scheier, C. (1999). *Understanding intelligence*. MIT Press, Cambridge, Massachusetts.
- Searle, J. (1980). Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–424.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, LIX:433–460.
- Vera, A. H. and Simon, H. A. (1993). Situated action: A symbolic interpretation. *Cognitive Science*, 17(1):7–48.
- Wang, P. (2005). Experience-grounded semantics: a theory for intelligent systems. *Cognitive Systems Research*, 6(4):282–302.
- Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht.
- Wang, P. (2007). Three fundamental misconceptions of artificial intelligence. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(3):249–268.
- Wang, P. (2008). What do you mean by “AI”? In *Artificial General Intelligence 2008*, pages 362–373, Amsterdam. IOS Press.

Case-by-Case Problem Solving

Pei Wang

Temple University, Philadelphia, USA

<http://www.cis.temple.edu/~pwang/>

Abstract

Case-by-case Problem Solving solves each occurrence, or case, of a problem using available knowledge and resources on the case. It is different from the traditional Algorithmic Problem Solving, which applies the same algorithm to all occurrences of all problem instances. Case-by-case Problem Solving is suitable for situations where the system has no applicable algorithm for a problem. This approach gives the system flexibility, originality, and scalability, at the cost of predictability. This paper introduces the basic notion of Case-by-case Problem Solving, as well as its most recent implementation in NARS, an AGI project.

Algorithmic Problem Solving

“Problem Solving” is the process to find a *solution* for a given *problem* by executing some *operations*. For a certain system at a certain moment, the set of executable operations usually remains constant. Therefore, the task for the system is to find a way to select proper operations and to execute them in proper order for the given problem.

In computer science and AI, the dominant approach in problem solving can be called “Algorithmic Problem Solving” (APS in the following). According to this approach, first a *problem* is specified as a function that maps any input (problem instance) of a certain type to the corresponding output. Then, an *algorithm* is designed, which accomplishes this function step by step, where each step is a well-defined operation. Finally, the algorithm is implemented in a programming language to become a computer program, which will be able to let a computer routinely transform valid input data into output data. A well-known description of this approach can be found in (Marr, 1982).

Accurately speaking, in this approach “problem solving” happens in two different levels:

1. When the *problem* refers to a *problem type*, or input-output mapping, the *solution* is the corresponding algorithm (conceptually speaking) or program (practically speaking) that accomplishes the mapping. For example, when the problem is “to sort sequences of comparable items”, one solution is “quicksort”.

2. When the *problem* refers to a *problem instance*, or input data, then the *solution* is the corresponding output data, according to the problem specification. For example, when the problem is “to sort [3, 2, 4, 1]”, the solution is “[1, 2, 3, 4]”.

So APS has two phases: at first a human solves a problem by designing an algorithm for it, then a computer applies the algorithm to solve concrete instances of the problem.

Computer science inherited APS from mathematics, and has successfully applied and enhanced it to provide a theoretical and methodological foundation for the information technology. Even so, this approach has its limitation:

- For some problems, no algorithm has been found. Even worse, for some problems it can be proved that no algorithm can be found. This is the issue of *computability* (Davis, 1958).
- For some problems, all known algorithms require too much time-space resources to solve every instances of the problem in practical situations. This is the issue of *computational complexity* (Cormen et al., 2001).

Beside the above issues that are well-known to computer science, AI has taken the additional challenge of building computer systems that require little human involvement in the whole problem solving process. To be intelligent, a computer system should have creativity and flexibility, which often means to be able to solve a problem for which it has not been given an applicable algorithm.

Some people consider this task as impossible: if everything a computer does follow some algorithm, how can it solve a problem for which no algorithm is given in advance? This opinion comes from a misconception, because a computer may be able to solve a problem without a predetermined algorithm *for that problem*, while in the whole process the system still follow algorithms defined on other problems, not the one under consideration (Wang, 2007).

Obviously, when a computer system must solve problems for which no algorithm is given in advance, then it can no longer follow the APS approach. In computer science and AI, many alternative approaches have been explored. This paper will not provide a comprehensive survey on this topic. Instead, it will concentrate on one approach, “Case-by-case Problem Solving”, describe its up-to-date implementation in an AGI system, and compare it with some of the alternatives.

CPS: the Basic Idea

Case-by-case Problem Solving (CPS) is a notion introduced in contrast with Algorithmic Problem Solving (APS). This notion was formed during the development of NARS, an AGI project, and the basic idea has been described in previous publications (Wang, 1996; Wang, 2004), though not bearing this name. Here the notion is briefly summarized and explained.

NARS is an intelligent system designed according to the theory that “intelligence” means “adaptation and working with insufficient knowledge and resources”. Descriptions of the whole project can be found in (Wang, 1995; Wang, 2006), and this paper only focuses on a certain aspect of the system.

NARS accepts three types of task from the environment: *knowledge* to be absorbed, *questions* to be answered, and *goals* to be achieved. Each piece of new knowledge is turned into a belief of the system, and is used in forward inference to derive or revise other beliefs; Each new question and goal, which is what we usually call a “problem”, is matched with existing beliefs for possible direct solutions, as well as used in backward inference to produce derived questions and goals, based on relevant beliefs.

One concrete implication of the above theory of intelligence is that an intelligent system, like NARS, often needs to deal with problems for which the system has no applicable algorithm, as a special case of “insufficient knowledge”. As analyzed in the previous section, this can be caused by various reasons, such as:

- The problem is not computable;
- Though the problem may be computable, no algorithm has been found yet;
- Though the problem can be solved by an algorithm, it is unknown to the system at the moment;
- Though the system knows some algorithmic solutions to the problem, it cannot afford the resource required by any of them.

No matter what the reason is, in this situation the system cannot follow APS. To work in this situation, there are two possible approaches:

1. Find an algorithm first, then use it to process the problem instances;
2. Directly solve the problem instances without following a predetermined algorithm.

While most of the relevant works in AI follow the first approach, in NARS the second approach is explored. Here a key observation is that the “problem” an intelligent system meets is usually a “problem instance”, rather than a “problem type”. The “sorting problem” ordinary people meet in their daily life is usually to sort concrete sequences, one at a time, not “to find a method to routinely sort any sequence”, as defined by mathematicians and computer scientists. Therefore, even when a problem type cannot be “solved” by an algorithm, some (even if not all) of its instances may still be solved, by taking the special properties of each of them into consideration. In this way, “problem

solving” is carried out in a *case by case* manner, and that is where the name CPS comes.

Some people may suspect CPS as APS rebranded, by treating what is previously taken as a *problem instance* as a *problem type* — though the system has no algorithm to sort all sequences, it might have an algorithm to sort [3, 2, 4, 1]. This is not what CPS means, because in a system like NARS, not only that each *instance* of the same problem type may be processed differently, but also that each *occurrence* of the same problem instance may be processed differently. This is not as strange as it sounds if we consider human problem solving, where the same problem (instance) often gets different treatment when it occurs in different contexts.

How about to insist that “The system is still following an algorithm for each *occurrence* of the problem, though different occurrences of the same problem may be handled by different algorithms”? After all, the actual solving process of the problem (occurrence) consists of nothing but a sequence of operations, right? Isn’t it just an algorithm? Such a usage of the notion of “algorithm”, though possible, would make it useless in analyzing the system, because such an “algorithm” can only be recorded *after* the problem-solving process, and is not repeatable. No matter what word is used, the system’s processing of the next occurrence of the problem is no longer accurately predictable, unless everything in the environment and the system are fully specified. In this situation the system does not serve as a fixed function mapping the problem instances to corresponding solutions.

The above analysis suggests that non-algorithmic CPS is not only logically possible, but also has the human mind as an existing proof. However, it does not tell us *how* to carry out this kind of process in a computer. After all, a computer system has to follow some algorithms (though not specific to the domain problems) to control its activities.

Theoretically speaking, if the precondition and consequence of each operation are accurately known, the system should be able to solve a concrete problem by exhaustively evaluating all possible operation sequences, and choosing the best solution according to their overall results. However, it is obvious that for any non-trivial problem such an exhaustive search will not be affordable. This is especially true for NARS, with its assumption on the insufficiency of knowledge and resources — “insufficient knowledge” means that the precondition and consequence of each operation are not accurately known, and “insufficient resources” means that the system does not have the time and space to consider all known possibilities. Under this assumption, by definition, the system cannot always find the best solution that guarantees the optimal result among all alternatives.

On the other hand, for a system working in this situation, the “insufficiency” assumption does not mean that all solutions are equally good. According to the opinion that intelligence is a form of adaptation, an intelligent system should pick the best solution that, according to its experience, is most likely to achieve a desired result, among the alternatives the system can consider with available resources.

As a realization of the above idea, the problem-solving process in NARS can be informally and briefly described as the following.

First, since NARS is designed in the framework of reasoning system, in it goals, operations, and beliefs are all represented as sentences in a formal language. A *goal* describes what the system want to achieve (i.e., to make it true); an *operation* can be directly achieved by executing some code; and a *belief* summarizes the system's experience on the relations among items in the system, including goals, operations, and other beliefs.

Assuming insufficient knowledge, in NARS a belief can only specify partial preconditions or consequences of an operation, with a truth-value to indicate the evidential support for it according to the system's experience. Each inference rule, when used for *forward* inference, takes a couple of existing beliefs as premise, and derives a conclusion, with a truth-value determined according to the evidence provided by the premises. With the coming of new evidence, new beliefs are derived, and existing beliefs are revised. Therefore, the system's overall opinion about the preconditions and consequences of each operation changes over time.

Similarly, a goal is a *statement*, not a *state*, so is a *incomplete* specification of a certain aspect of the (internal or external) environment. There are inference rules used for *backward* inference, to produce derived goals, recursively from existing goals and beliefs. At any moment, there are usually many (input or derived) goals in the system, which are not necessarily consistent in what they specify as desired. If according to its experience the system expects the execution of a certain operation will achieve a certain goal, there is no guarantee that the expectation will be confirmed by future experience. Furthermore, the operation may have some undesired impact on other goals. Therefore, in the system each statement has a desire-value associated to summarize its overall relations with the goals considered, which, plus some other factors, will decide whether the system will take the statement as a goal.

Under the assumption of insufficient resources, the system cannot afford to explore all possibilities by interacting *every* task with *every* (relevant) belief. Instead, it can only let *selected* tasks interact with *selected* beliefs. The selections are based on the system's evaluation on the *priority* (which summarizes factors like *urgency*, *importance*, *relevance*, *usefulness*, etc.) of the tasks and beliefs, according to the system's experience. Since the system constantly gets new experience while communicating with the environment and working on the tasks, the evaluation results change from time to time.

The above mechanism inevitably leads to CPS. To NARS, each task corresponds to a new *case*, that is, the occurrence of a problem instance in a internal context (defined by the available knowledge and resources at the moment). What the system does to a task is determined by what the system knows about it (the existing relevant beliefs), how much resources the system can spend on it (the number of beliefs that will be selected), and the priority distribution among the beliefs (the access order of the selected beliefs). Since the above factors are constantly changing, the processing of a given task becomes unpredictable and non-repeatable according to the task alone, and the problem-solving process cannot be abstracted as APS.

CPS with Procedural Knowledge

In this section, more technical details are provided on the CPS process in NARS. Given the paper length restriction, here the focus is in the recent progress on CPS with *procedural* knowledge. For CPS with *declarative* knowledge, see (Wang, 1996; Wang, 2004).

NARS uses a formal language Narsese, which is term-oriented, that is, a statement in it typically has the form of *subject-copula-predicate*. While "There is a R relation among objects a, b, c " is usually represented in predicate logic as $R(a, b, c)$, in Narsese it becomes $((\times a b c) \rightarrow R)$, which states that the tuple $[a, b, c]$ (the subject term) is a special case of the relation R (the predicate term), and ' \rightarrow ' (the copula) is the *inheritance* relation.

A "statement on statements" can be represented as a *higher-order statement*. For example, "An R_1 relation among objects a, b, c implies an R_2 relation among b, a, c " is represented as $((\times a b c) \rightarrow R_1) \Rightarrow ((\times b a c) \rightarrow R_2)$, where the two terms are statements, and ' \Rightarrow ' is the *implication* relation, another type of copula.

An *event* is a statement with temporal information. For example, "An R_1 relation among objects a, b, c is usually followed by an R_2 relation among b, a, c " is represented as $((\times a b c) \rightarrow R_1) \text{ /}\Rightarrow ((\times b a c) \rightarrow R_2)$, where ' $\text{ /}\Rightarrow$ ' is implication plus the temporal information that the event as subject happens before the event as predicate.

With insufficient knowledge, in NARS no statement is absolutely true. A *judgment* is a statement with a truth-value attached, indicating the evidential support the statement gets from the experience of the system. A truth-value consists of two factors: a *frequency* factor in $[0, 1]$, measuring the proportion of positive evidence among all available evidence, and a *confidence* factor in $(0, 1)$, measuring the proportion of current evidence among future evidence, after the coming of new evidence of a unit amount. For example, if statement $((\times a b c) \rightarrow R)$ has been tested 4 times, and in 3 of them it is true, while in 1 of them it is false, the truth-value of the statement is $f = 3/4 = 0.75$, $c = 4/5 = 0.80$, and the judgment is written as " $((\times a b c) \rightarrow R) <0.75; 0.80>$ ".

A *goal* is an event the system wants to achieve, that is, the system is willing to do something so that the truth-value of that statement will approach $<1.00; 1.00>$ as closely as possible. The attached desire-value of each goal is the truth-value of the system's belief that the achieving of the goal really leads to desired situations.

An *operation* is an event that the system can directly realize by executing some program (which are usually not written in Narsese). In other words, it is a statement with a "procedural interpretation", as in logic programming. For example, if the term R corresponds to the name of an operation, and a, b , and c are arguments of the operation, then $((\times a b c) \rightarrow R)$ represent the event that R is applied on a, b , and c , which is a special case for the three to be related.

The system's knowledge about an operation is mainly represented as beliefs on what the operation implies, as well as what it is implied by. Each belief provides partial information about the precondition or consequence of the operation, and the overall meaning of the operation, to the system, is the collection of all such beliefs. To simplify the description,

in the following a term, like S , will be used to represent a statement, such as $((\times a b c) \rightarrow R)$.

In NARS, a judgment $(S_1 \not\Rightarrow S_2) <f; c>$ can be used to uniformly represent many different types of knowledge.

- If S_1 is directly about an operation, but S_2 is not, then the judgment represents a belief on an *effect* or *consequence* of the operation;
- If S_2 is directly about an operation, but S_1 is not, then the judgment represents a belief on a *cause* or *precondition* of the operation.

Such a judgment can be used by various rules. In forward inference, it and a judgment on S_1 can derive a judgment on S_2 by the *deduction* rule, as a prediction; it and a judgment on S_2 can derive a judgment on S_1 by the *abduction* rule, as an explanation. This judgment itself can be derived from the system's observation of event S_1 followed by event S_2 , by the *induction* rule, as a generalization. In backward inference, this judgment and a goal (or a question) on S_2 can derive a goal (or a question) on S_1 . Different rules use different truth-value functions to calculate the truth-value of the conclusion from those of the premises. The details of these rules, with their truth-value functions, can be found in (Wang, 2006).

For more complicated situations, both the S_1 and S_2 in above judgment can be compound statements consisting of other statements. For example, very common the condition part of an implication statement is a "sequential conjunction", as in $((S_1, S_2) \not\Rightarrow S_3)$, which means the event sequence " S_1 , then S_2 " is usually followed by event S_3 . When S_2 is an operation, such a statement represents its (partial) precondition and consequence. When S_3 is a goal, (S_1, S_2) indicates a plan to achieve it.

The inference rules of NARS carry out various cognitive functionalities in a uniform. Beside the above mentioned prediction, explanation, and generalization, the system can also do planning (finding a sequence of operations that lead to a given goal), skill learning (forming stable operation sequence with useful overall function), decision making (choosing among alternatives), etc., though in this paper their details cannot be explained. Working examples of these functions in NARS can be found at the project website <http://code.google.com/p/open-nars/>.

In NARS, all beliefs (existing judgments) and tasks (new knowledge, questions, and goals) are clustered into *concepts*, according to the terms appearing in them. The system runs by repeating the following working cycle:

1. Select tasks in a task buffer to insert into the corresponding concepts, which may trigger the creation of new concepts and beliefs, as well as direct processing on the tasks.
2. Select a concept from the memory, then select a task and a belief from the concept.
3. Feed the task and the belief to the inference engine to produce derived tasks.
4. Add the derived tasks into the task buffer, and send report to the environment if a task provides a best-so-far answer to an input question, or indicates the realization of an input goal.

5. Return the processed belief, task, and concept back to memory.

The selections in the first two steps are all probabilistic, with the probability for an item (concept, task, or belief) to be selected proportional to its priority value. In the last step, the priority of the involved items are adjusted according to the immediate feedback obtained from the inference result.

Now we can see that for a given task, its processing path and result are determined by the beliefs interacting with it, as well as the order of the interactions (that is, inference steps), which in turn depends on the items in the memory (concepts, tasks, and beliefs), as well as the priority distributions among the items. All these factors change constantly as the system communicates with the environment and works on the tasks. As a result, there is no algorithm specifying the inference step sequence for a task. Instead, this sequence is formed at run time, determined by many preceding events in the system. In this way, task processing (that is, problem solving) in NARS becomes "case by case".

Comparison and Discussion

CPS and APS are different approaches of problem solving in computer. In APS, it is the programmer who solves the problem (as a class), and the computer just applies the solution to each instance of the problem. In CPS, it is the computer that directly solves the problem (as a case), depending on its available knowledge and resources. A CPS system still follow algorithms, but these algorithms are not solutions of domain-specific problems. Instead, they are domain-independent solutions of "meta-problems" like the handling of input/output, the carrying out of the inference steps, the allocating of resources, etc.

These two approaches are suitable for different situations. Given the scope of the problems a system faces, APS is preferred when there are sufficient knowledge (to get a problem-specific algorithm) and resources (to execute the algorithm), while CPS is an option when no problem-specific algorithm is available and affordable. CPS gives the system creativity, flexibility, and robustness, though it lacks the predictability, repeatability, and reliability of APS.

CPS processes are difficult to analyze, because the traditional theories on computability and computational complexity become inapplicable at the problem-solving level (though it may be applied in other levels), as the solvable problems and the solution costs all become context-sensitive and practically unpredictable, unless the system's experience in the past and near future (when the problem is being solved) is fully known, and the system can be simulated step-by-step with all details.

Some claims on the limitations of AI are based on the "non-algorithmic" nature of intelligence and cognition, as in (Dreyfus, 1979; Penrose, 1989). When facing CPS systems, all such claims become invalid, because the problem-solving processes in these systems are already non-algorithmic. This topic has been discussed with more details in (Wang, 2007), and will not be repeated here.

A large part of AI research is driven by the challenge of problems for which no efficient algorithm is available. The

typical response is to find such an algorithm first, then to use it in APS. CPS is different from these techniques in its basic idea, though still related to them here or there.

One of the earliest AI technique is *heuristic search* (Newell and Simon, 1976). Since all possible solutions come from permutations of a constant set of basic operations, problem solving in theory can be described as searching for a path from the initial state to a goal state in a state space. Because exhausting all possibilities usually demands unaffordable resources, the key becomes the selection of paths to be explored. NARS is similar to heuristic search in that (1) it compares alternative paths using numerical functions, since in NARS the truth-values, desire-values, and priority-values all have impact on the order by which the alternatives are explored, and (2) the system usually gets satisfying solutions, rather than optimal solutions. Their major differences are that at each step NARS does not evaluate a static list of alternatives according to a fixed heuristic, but recognizes and builds the alternatives by reasoning, and allocates resources among them, so to explore them in a *controlled concurrency* (Wang, 1996), which is similar to *parallel terraced scan* (Hofstadter and FARG, 1995). Furthermore, in NARS the heuristic information is provided mainly by the domain knowledge, which is not built into the system, but learned and derived, so is flexible and context-sensitive, while a heuristic algorithm has a fixed step sequence.

A related technique is *production system*, or *rule-based system*, where each state change is caused by the applying of a rule, and different solutions correspond to different rule-application sequences. NARS looks like such a system, since it also describes a problem in a formal language, and modifies the description by rules during inference. However, in traditional rule-based system there is a static long-term memory (containing *rules*) and a changeable working memory (containing *facts*) (Newell, 1990). In a problem solving process, only the latter is changed, and after the process, the working memory is reset. Therefore, the system still does APS, since it provides a (fixed) mapping from input (problem instances) to output (solutions), even though here the “algorithm” is not explicitly coded as a program, but is implicitly distributed among the rules and the control mechanism (which is responsible for selecting a rule to fire in each working cycle). On the contrary, in NARS the content of memory is modified by every problem-solving process, so the processes have strong mutual influence, and it is impossible to analyze one of them without the others.

The “case” in CPS should not be confused with the same term in *Case-Based Reasoning* (Leake, 1996), which solves problems by revising solutions of similar problems — that is still APS, with the algorithms distributed among the cases and the control mechanism (which is responsible for selecting similar cases and putting together a new solution).

More complicated forms of APS can be found in works on *randomized algorithms* (Cormen et al., 2001), *anytime algorithms* (Dean and Boddy, 1988), and *metareasoning* (Russell and Wefald, 1991). Though driving by different considerations and suggesting different approaches, each of these technique solves a problem with a family of algorithms, rather than a single one. For a given problem instance, some

outside factor (beyond the input data) decides which algorithm in the family will be selected and applied. In randomized algorithms, the selection is made randomly; in anytime algorithms, it is determined by the executing time restriction; and in metareasoning, it is the result of an explicit deliberation. If we treat these factors as an additional argument of the problem, used as the index of the algorithm selected from the family, all these situations are reduced to APS. On the contrary, CPS cannot be reduced into APS in this way, since it is not even a selection among preexisting algorithms. If a problem (instance) is repeatedly solved in NARS, the solution does not form a probability distribution (as in a randomized algorithms). Even if the same amount of time is allocated to a problem, in NARS the results can still be different, while according to anytime algorithm and metareasoning, the results should be the same. Even so, NARS still share some properties with these approaches. For example, given more resources, the system usually provides better solutions, as an anytime algorithm (Wang, 1996).

In NARS a problem-solving process is non-algorithmic, because the process is not built into the system, but formed by the learning and adaptation mechanism of the system. To learn problem-solving skills from environment feedback is not a new idea at all (Turing, 1950). There is a whole *reinforcement learning* field aimed at the optimization of the reward from the environment to the system’s operations (Kaelbling et al., 1996). Also, *genetic programming* provides a powerful way to generate algorithms for many problems (Koza, 1992). Though these techniques (and some others) are very different in details, they are still within the APS framework given at the beginning of the paper: first, find an algorithm for a problem class, then, apply the algorithm to each problem instance. What these techniques aim is to replace the human designer in the first phase of APS. This is different from the aim of CPS, which merge these two phases into one, by directly solving problem instances in a non-algorithmic manner. There are indeed many situations where algorithms are desired, and therefore some kind of algorithm-learning technique will be preferred. However, there are also situations where the system cannot satisfy the demand of these algorithm-learning techniques on knowledge and resources, so CPS will be more proper. CPS does not reject all forms of skill learning, as far as it does not reduce the problem-solving process into APS. As described in the previous section, NARS can learn procedures.

In summary, CPS is designed for a situation where no existing technique can be applied, rather than as an alternative to an existing technique in the field for which it is designed. CPS is similar to the existing techniques in many aspects, but cannot be reduced to any of them.

Conclusion

For problem-solving, the “case-by-case” approach (CPS), which is used in NARS and advocated in this paper, is different from the algorithmic approach (APS) by taking the following positions:

- Do not define a “problem” as a class and use the same method to solve all of its instances. Instead, treat each

“problem instance” as a “problem” on its own, and solve it in a case-by-case manner, according to the current (knowledge/resource) situation in the system.

- Do not draw a sharp line between solutions and non-solutions for a given problem, and treat all solutions as equally good. Instead, allow solutions to be partial, and compare candidate solutions to decide which one is better.
- Do not insist on “one problem, one solution”. Instead, allow the system to generate zero, one, or a sequence of solutions, each of which is better than the previous ones.
- Do not depend on a predetermined algorithm to solve a problem. Instead, cut a problem-solving process into steps. Though each step follows an algorithm, the overall process is formed by linking steps together at run time in a context-sensitive manner.
- Do not predetermine the method by which a problem is processed in each step. Instead, let the selected problem and available knowledge decide how the problem is processed in that step.
- Do not attempt to use all relevant beliefs to solve a problem. Instead, in each step only consider one of them, selected according to their priority values.
- Do not solve problems one after another. Instead, process problems (and subproblems) in parallel, but at different speed, according to their priority values.
- Do not throw away the intermediate results at the end of a problem-solving process. Instead, keep them for future problems.
- Do not isolate each problem-solving process in its own working space. Instead, let all problems interact with the same memory.
- Do not attempt to keep all beliefs forever. Instead, remove items with the lowest priority when the memory is full.
- Do not process each problem with a fixed resources supply. Instead, let the processes compete for resources.
- Do not keep a fixed resources distribution. Instead, adjust the priority distribution according to the experience of the system and the current context, so as to give important and relevant items more resources.

Some of the above issues are discussed in this paper, while the others have been addressed in related publications (Wang, 1996; Wang, 2004; Wang, 2006).

This new approach for problem-solving is proposed as a supplement to the traditional (algorithmic) approach, for situations where the system has insufficient knowledge and resources to apply or to build an algorithm for the problem it faces.

The practice of project NARS shows that such an approach can be implemented, and has properties not available in traditional systems. Since the capability of such a system is not only determined by its design, but also by its experience, it is hard to evaluate the potential of this approach in solving practical problems. However, at least we can say that this approach is exploring a territory beyond the scope of classic theory of computation, and it is more similar to the actual thinking process of the human mind.

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, McGraw-Hill Book Company, 2nd edition.
- Davis, M. (1958). *Computability and Unsolvability*. McGraw-Hill, New York.
- Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of AAAI-88*, pages 49–54.
- Dreyfus, H. L. (1979). *What Computers Can't Do: Revised Edition*. Harper and Row, New York.
- Hofstadter, D. R. and FARG (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, New York.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts.
- Leake, D., editor (1996). *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, Menlo Park, California.
- Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman & Co., San Francisco.
- Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.
- Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126.
- Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press.
- Russell, S. and Wefald, E. H. (1991). Principles of meta-reasoning. *Artificial Intelligence*, 49:361–395.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, LIX:433–460.
- Wang, P. (1995). *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. PhD thesis, Indiana University.
- Wang, P. (1996). Problem-solving under insufficient resources. In *Working Notes of the AAAI Fall Symposium on Flexible Computation*, pages 148–155, Cambridge, Massachusetts.
- Wang, P. (2004). Problem solving with insufficient resources. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 12(5):673–700.
- Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht.
- Wang, P. (2007). Three fundamental misconceptions of artificial intelligence. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(3):249–268.

What Is Artificial General Intelligence?

Clarifying The Goal For Engineering And Evaluation

Mark R. Waser

Books International
22883 Quicksilver Drive, Dulles, VA 20166
MWaser@BooksIntl.com

Abstract

Artificial general intelligence (AGI) has no consensus definition but everyone believes that they will recognize it when it appears. Unfortunately, in reality, there is great debate over specific examples that range the gamut from exact human brain simulations to infinitely capable systems. Indeed, it has even been argued whether specific instances of humanity are truly generally intelligent. Lack of a consensus definition seriously hampers effective discussion, design, development, and evaluation of generally intelligent systems. We will address this by proposing a goal for AGI, rigorously defining one specific class of general intelligence architecture that fulfills this goal that a number of the currently active AGI projects appear to be converging towards, and presenting a simplified view intended to promote new research in order to facilitate the creation of a safe artificial general intelligence.

Classifying Artificial Intelligence

Defining and redefining “Artificial Intelligence” (AI) has become a perennial academic exercise so it shouldn’t be surprising that “Artificial General Intelligence” is now undergoing exactly the same fate. Pei Wang addressed this problem (Wang 2008) by dividing the definitions of AI into five broad classes based upon how a given artificial intelligence would be similar to human intelligence: in structure, in behavior, in capability, in function, or in principle. Wang states that

These working definitions of AI are all valid, in the sense that each of them corresponds to a description of the human intelligence at a certain level of abstraction, and sets a precise research goal, which is achievable to various extents. Each of them is also fruitful, in the sense that it has guided the research to produce results with intellectual and practical values. On the other hand, these working definitions are different, since they set different goals, require different methods, produce different results, and evaluate progress according to different criteria.

We contend that replacing the fourth level of abstraction (Functional-AI) with “similarity of architecture of mind (as opposed to brain)” and altering its boundary with the fifth would greatly improve the accuracy and usability this scheme for AGI. Since Stan Franklin proposed (Franklin 2007) that his LIDA architecture was “ideally suited to provide a working ontology that would allow for the discussion, design, and comparison of AGI systems” since it implemented and fleshed out a number of psychological and neuroscience theories of cognition and since the feasibility of this claim was quickly demonstrated when Franklin and the principals involved in NARS (Wang 2006), Novamente (Looks, Goertzel and Pennachin 2004), and Cognitive Constructor (Samsonovitch et. al. 2008) put together a comparative treatment of their four systems based upon that architecture (Franklin et al. 2007), we would place all of those systems in the new category.

Making these changes leaves three classes based upon different levels of architecture, with Structure-AI equating to brain architecture and Principle-AI equating to the architecture of problem-solving, and two classes based upon emergent properties, behavior and capability. However, it must be noted that both of Wang’s examples of the behavioral category have moved to more of an architectural approach with Wang noting the migration of Soar (Lehman, Laird and Rosenbloom 2006; Laird 2008) and the recent combination of the symbolic system ACT-R (Anderson and Lebiere 1998, Anderson et al. 2004) with the connectionist [L]eabra (O’Reilly, and Munakata 2000), to produce SAL (Lebiere et al. 2008) as the [S]ynthesis of [A]CT-R and [L]ibra. Further, the capability category contains only examples of “Narrow AI” and Cyc (Lenat 1995) that arguably belongs to the Principle-AI category.

Viewing them this way, we must argue vehemently with Wang’s contentions that “these five trails lead to different summits, rather than to the same one”, or that “to mix them together in one project is not a good idea.” To accept these arguments is analogous to resigning ourselves to being blind men who will attempt only to engineer an example of elephantness by focusing solely on a single view of elephantness, to the exclusion of all other views and to the extent of throwing out valuable information. While we certainly agree with the observations that “Many current AI projects have no clearly specified research goal, and

people working on them often swing between different definitions of intelligence” and that this “causes inconsistency in the criteria of design and evaluation”, we believe that the solution is to maintain a single goal-oriented focus on one particular definition while drawing clues and inspiration from all of the others.

What Is The Goal of AGI?

Thus far, we have classified intelligence and thus the goals of AI by three different levels of abstraction of architecture (i.e. what it is), how it behaves, and what it can do. Amazingly enough, what we haven’t chosen as a goal is what we want it to do. AGI researchers should be examining their own reasons for creating AGI both in terms of their own goals in creating AGI and the goals that they intend to pass on and have the AGI implement. Determining and codifying these goals would enable us to finally knowing the direction in which we are headed.

It has been our observation that, at the most abstract level, there are two primary views of the potential goals of an AGI, one positive and one negative. The positive view generally seems to regard intelligence as a universal problem-solver and expects an AGI to contribute to solving the problems of the world. The negative view sees the power of intelligence and fears that humanity will be one of the problems that is solved. More than anything else, we need an AGI that will not be inimical to human beings or our chosen way of life.

Eliezer Yudkowsky claims (Yudkowsky 2004) that the only way to sufficiently mitigate the risk to humanity is to ensure that machines always have an explicit and inalterable top-level goal to fulfill the “perfected” goals of humanity, his Coherent Extrapolated Volition or CEV. We believe, however, that humanity is so endlessly diverse that we will **never** find a coherent, non-conflicting set of ordered goals. On the other hand, the presence of functioning human society makes it clear that we should be able to find some common ground that we can all co-exist with.

We contend that it is the overly abstract Principle-AI view of intelligence as “just” a problem-solver that is the true source of risk and that re-introducing more similarity with humans can cleanly avoid it. For example, Frans de Waal, the noted primatologist, points out (de Waal 2006) that any zoologist would classify humans as *obligatorily gregarious* since we “come from a long lineage of hierarchical animals for which life in groups is not an option but a survival strategy”. If we, therefore, extended the definition of intelligence to “The ability **and desire** to live and work together in an inclusive community to solve problems and improve life for all” there would be no existential risk to humans or anyone else.

We have previously argued (Waser 2008) that acting ethically is an attractor in the state space of intelligent behavior for goal-driven systems and that humans are basically moral and that deviations from ethical behavior on the part of humans are merely the result of

shortcomings in our own foresight and intelligence. As pointed out by James Q. Wilson (Wilson 1993), the real questions about human behaviors are not why we are so bad but “how and why most of us, most of the time, restrain our basic appetites for food, status, and sex within legal limits, and expect others to do the same.”

Of course, extending the definition of intelligence in this way should also impact the view of our stated goal for AGI that we should promote. The goal of AGI cannot ethically be to produce slaves to solve the problems of the world but must be to create companions with differing capabilities and desires who will journey with us to create a better world.

Ethics, Language, and Mind

The first advantage of this new goal is that the study of human ethical motivations and ethical behavior rapidly leads us into very rich territory regarding the details in architecture of the mind required for such motivations and behaviors. As mentioned repeatedly by Noam Chomsky but first detailed in depth by John Rawls (Rawls 1971), the study of morality is highly analogous to the study of language since we have an innate moral faculty with operative principles that cannot be expressed in much the same way we have an innate language faculty with the same attributes. Chomsky transformed the study of language and mind by claiming (Chomsky 1986) that human beings are endowed with an innate program for language acquisition and developing a series of questions and fundamental distinctions. Chomsky and the community of linguists working within this framework have provided us with an exceptionally clear and compelling model of how such a cognitive faculty can be studied.

As pointed out by Marc Hauser (Hauser 2006; Hauser, Young and Cushman 2008), both language and morality are cognitive systems that can be characterized in terms of principles or rules that can construct or generate an unlimited number and variety of representations. Both can be viewed as being configurable by parameters that alter the behavior of the system without altering the system itself and a theory of moral cognition would greatly benefit from drawing on parts of the terminology and theoretical apparatus of Chomsky’s Universal Grammar.

Particularly relevant for the development of AGI, is their view that it is entirely likely that language is a mind-internal computational system that evolved for internal thought and planning and only later was co-opted for communication. Steven Pinker argues (Pinker 2007) that studying cross-cultural constants in language can provide insight into both our internal representation system and when we switch from one model to another. Hauser’s studies showing that language dramatically affects our moral perceptions argues that they both use the same underlying computational system and that studying cross-cultural moral constants could not only answer what is moral but how we think and possibly even why we talk.

Finally, the facts that both seem to be genetically endowed but socially conditioned and that we can watch the formation and growth of each mean that they can provide windows for observing autogeny in action.

Growing A Mind

One difference between most AGI researchers and many others working in the field of AI is the recognition that a full-blown intelligence is not going to be coded into existence. While AI researchers universally recognize the requirement of learning, there frequently isn't the recognition that the shortest path to AGI is to start with a certain minimal seed and to have the AGI grow itself from there. Indeed, many AGI research projects seem to have also lost this critical focus and be concentrating more on whether specific capabilities can be programmed in specific ways or on specific knowledge representations rather than focusing on the far more difficult subjects of what is required for effective growth from such a seed and how it might be implemented.

The interesting and important question, of course, is "What is the minimum critical mass for the seed AGI and what proportion of that mass is composed of hard-coded initial information as opposed to instructions for reasoning and growth?" Undoubtedly, there are many correct answers that will lead to a variety of different AGIs but we would prefer to pick one with a shorter path and time frame and a lesser amount of effort rather than a longer or more difficult path.

Daniel Oblinger (Oblinger 2008) has gone so far as to posit that it is possible that the majority of the work currently being done is unnecessary and can, and quite possibly will, be avoided by working instead on the bootstrapping process itself. It is his hope that a very minimal embedded system with the familiar AGI cognitive cycle (perceive/abstract/act or sense/cognize/act), the appropriate internal "emotional" drivers, and certain minimal social abilities will be able to use "embodiment scaffolding" and "social scaffolding" as a framework for growth that will permit the bootstrapping of strong performance from repeated iterations of weak learning. Both Marvin Minsky (Minsky 2006) and J. Storrs Hall (Hall 2007) give plausible models that we should be able to extend further.

On the other hand, longitudinal studies of twins raised apart (Bouchard 1990) show surprisingly high correlation levels in an incredible variety of choices, behaviors and outcomes. This, plus the examples of language and morality, suggests that much more of the details of intelligence are programmed in genetics than we might otherwise generally believe. It is our contention that studying the formation and growth of these examples will not only give us additional insight into the architecture of the human mind but is actually the quickest and most likely path to AGI by providing enough information to build the seed for a human-like architecture.

Architecture of Mind

Since we have previously noted that LIDA architecture implements and fleshes out a number of psychological and neuroscience theories of cognition and has already been deemed as an acceptable basis for comparison by the principals of a number of projects, we will consider it the consensus architecture of mind. The most salient features of LIDA's architecture are its cognitive cycle; the fact that it is very much an attentional architecture based upon Sloman's architecture for a human-like agent (Sloman 1999); and its use of Baar's global workspace theory of consciousness (Baars 1993, 1997, 2003; Newman, Baars and Cho 2003; Baars and Franklin 2007).

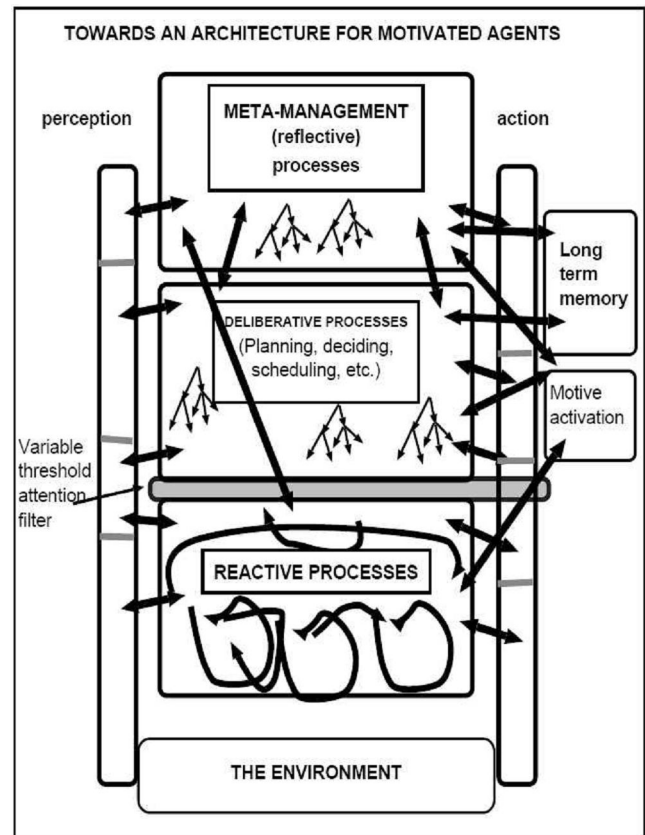


Figure 1. Sloman's human-like agent architecture

Franklin starts with an embodied autonomous agent that senses its environment and acts on it, over time, in pursuit of its own agenda. While it doesn't have the bootstrap view of what is the most minimal cycle that can build the simplest tool that can then be used as a building block to create the next tool, the LIDA model does include automatization, the process of going from consciously learning something like driving to the effortless, frequently unconscious, automatic actions of an experienced driver. Since it is embodied and all cognitive symbols are ultimately grounded in perception, it is not subject to the symbol-grounding problem (Harnad 1990).

Franklin characterizes the simplicity of the initial agent by saying:

It must have sensors with which to sense, it must have effectors with which to act, and it must have primitive motivators ... [drives] ... which motivate its actions. Without motivation, the agent wouldn't do anything. Sensors, effectors, and drives are primitives which must be built into, or evolved into, any agent.

Unfortunately, we would argue, for the purposes of both autogeny and morality, far too little attention has been paid to drives and their implementation.

Conscious Attention

In many ways, the most important feature of Sloman's architecture is the grey bar across the middle between conscious attentional processes and unconscious processes. Alfred North Whitehead claimed, "Civilization advances by extending the number of important operations which we can perform without thinking about them." We contend that the same is true of intelligence and would argue that there has been far less attention to the distinction between conscious and unconscious processing than we believe is warranted.

Experimental studies (Soon et. al. 2008) show that many decisions are made by the unconscious mind up to 10 seconds before the conscious mind is aware of it. Further, a study of the "deliberation-without-attention" effect (Dijksterhuis et al. 2006) shows clearly that engaging in a thorough conscious deliberation is only advantageous for simple choices while choices in complex matters should be left to unconscious thought. This effect is attributed to the fact that a person can pay conscious attention to only a limited amount of information at once, which can lead to a focus on just a few factors and the loss of the bigger picture. Logically, constraint satisfaction or optimization would seem to be an operation that would be best implemented on a parallel architecture (the unconscious) with a serial post-process (consciousness) for evaluating and implementing the result -- and another serial post-post-process for evaluating the results of the implementation and learning from them). Arguably, from the experiments presented above, it is entirely possible that the conscious mind merely "set up" the problem and then runs it on an unconscious tool.

Attention is also particularly important since it facilitates a second aspect of behavior control. As Minsky points out (Minsky 2006), most of our drives have both a sensory control and an attentional control. Sex not only feels good and but sexual thoughts tend to grab our attention and try to take over. Similarly, pain hurts and can distract us enough to prevent us from thinking of anything else.

Baars Global Workspace Theory postulates (Baars 1997) that most of cognition is implemented by a multitude of relatively small, local, special purpose processes, that are almost always unconscious. Coalitions of these processes compete for conscious attention (access to a limited

capacity global workspace) that then serves as an integration point that allows us to deal with novel or challenging situations that cannot be dealt with efficiently, or at all, by local, routine unconscious processes. Indeed, Don Perlis argues (Perlis 2008) that Rational Anomaly Handling is "the missing link between all our fancy idiot-savant software and human-level performance."

A More Abstract View

An interesting abstraction of this architecture yields a simple view of intelligence, composed of just three parts, which is still complex enough to serve as a foundation to guide research into both the original evolution of the mind and also how individual human minds grow from infancy. The first part of the mind is the simple unconscious processes. Initially these must be hard-wired by genetics. The next part is a world model that has expectations of the world and recognizes anomalies. Desires are also a part of this world model. The third part is the integrative conscious processes that are not only invoked to handle anomalies but are also used to improve the world model and develop new unconscious processes.

This simple model captures many of the features of the human mind that many current models do not. Most important is the balance of the conscious processes being a slave to the desires and context of the world model formed initially and constantly revised by the subconscious yet being able to modify that model and create new subconscious processes. This is the dynamic of the seed that we contend is the quickest and safest path to AGI.

An important first question for ontogeny is where genetically "hard-coded" processes and model features stop and learned processes and features start. For example, evolution clearly has "primed" us with certain conceptual templates, particularly those of potential dangers like snakes and spiders (Ohman, Flykt and Esteves 2001). Equally interesting is the demonstration of the beginning of moral concepts like fairness in dogs (Range et al 2008) and monkeys (Brosnan and de Wall 2003).

What we believe to be most important, however, is further research into the development of a sense of self including its incredible plasticity in the world model and it's effects upon both the conscious and subconscious. Too many AGI researchers are simply waiting for a sense of self to emerge while the "Rubber Hand Illusion" (Botvinick and Cohen 1998) and the "Illusion of Body Swapping" (Petkova and Ehrsson 2008) give important clues as to how incredibly disparate subconscious processes will appear to the conscious mind merely as extensions to itself.

This is important point because it means that anything that can be plugged into the global workspace is immediately usable whether the conscious mind understands its internal operation or not. Of course, this immediately begs the question of exactly what the detailed "plug-and-play" interface specifications of the workspace architecture are -- and this is where current systems all

differ. NARS uses Narsese, the fairly simple yet robust knowledge representation language of the system as an integration point. Novamente uses complex node-and-link hypergraphs. Polyscheme (Cassimatis 2005, 2006) uses numerous different representation schemes and attempts to implement the basic cognitive algorithms over them all.

More important than the knowledge representation scheme, we believe, however, is how the mechanism of attention is actually implemented. In LIDA, attention is the work of attention codelets that form coalitions to compete **in parallel** for access to the global workspace. Filtering occurs in multiple locations and is pretty much ubiquitous during cognition. Other systems merely label the various units of their representation schemes with interest values and priorities but there are tremendously variable degrees as to where attention falls on the spectrum of serial to parallel. It is our fear that the systems that do not dramatically limit the size of consciousness have deviated far enough from the model of human intelligence as to be in uncharted waters but only time will tell.

Conclusion

We have argued that creating an **Ethical Autogenous Attentional Artificial General Intelligence** (EA3GI) is likely to be the fastest and safest path to developing machine intelligence and that focusing on creating companions with differing capabilities and desires who will journey with us to create a better world instead of producing slaves to solve the problems of the world should be the consensus goal of AGI research.

References

Anderson, J.R.; Bothell, D.; Byrne, M.D.; Douglass, S.; Lebiere, C. and Qin, Y. 2004. An integrated theory of Mind. In *Psychological Review* 111:4.

Anderson, J.R. and Lebiere, C. 1998. *The atomic components of thought*. Mahwah, New Jersey: Erlbaum.

Baars, B.J. 1993. *A Cognitive Theory of Consciousness*. Cambridge University Press.

Baars, B.J. 1997. In *The Theater of Consciousness: The Workspace of the Mind*. New York, New York: Oxford University Press.

Baars, B.J. 2003. How Does a Serial, Integrated, and Very Limited Stream of Consciousness Emerge from a Nervous System That Is Mostly Unconscious, Distributed, Parallel, and of Enormous Capacity? In Baars, B.J.; Banks, W.P.; and Newman, J.B. eds *Essential Sources in the Scientific Study of Consciousness*. Cambridge, MA: MIT Press.

Baars, B.J. and Franklin, S. 2007. An architectural model of conscious and unconscious brain functions: Global

Workspace Theory and IDA. In *Neural Networks* 20. Elsevier.

Beck, J.; Ma, W.J.; Kiani, R.; Hanks, T.; Churchland, A.K.; Roitman, J.; Shadlen, M.; Latham, P.E.; and Pouget, A. 2008. Probabilistic Population Codes for Bayesian Decision Making. *Neuron* 60(6): 1142 - 1152.

Botvinick, M. and Cohen, J. 1998. Rubber hands ‘feel’ touch that eyes see. *Nature* 391: 756–756.

Bouchard, T.J. Jr; Lykken. D.T.; McGue, M.; Segal, N.L.; and Tellegen, A. 1990. Sources of human psychological differences: the Minnesota Study of Twins Reared Apart. *Science* 250: 223–228.

Brosnan, S. and de Wall, F. 2003. Monkeys reject unequal pay. *Nature* 425: 297-299.

Cassimatis, N. 2005. Integrating Cognitive Models Based on Different Computational Methods. In *Proceedings of the Twenty-Seventh Annual Conference of the Cognitive Science Society*. MahWah, New Jersey: Erlbaum.

Cassimatis, N. 2006. A Cognitive Substrate For Human-Level Intelligence. In *Artificial Intelligence Magazine*: 27. Menlo Park, CA: AAI Press.

Chomsky, N. 1986. *Knowledge of Language: Its Nature, Origin, and Use*. New York, NY: Praeger Publishers.

Dijksterhuis, A.; Bos, M.; Nordgren, L.; and Baaren, R. van 2006 On Making the Right Choice: The Deliberation-Without-Attention Effect. *Science* 311: 1005 – 1007.

Franklin, S. 2007. A Foundational Architecture for Artificial General Intelligence. In Goertzel, B and Wang, P. eds. *Advances in Artificial General Intelligence*. Amsterdam, The Netherlands: IOS Press.

Franklin, S.; Goertzel, B.; Samsonovich, A. and Wang, P. 2007. Four Contemporary AGI Designs: A Comparative Treatment. In Goertzel, B and Wang, P. eds. *Advances in Artificial General Intelligence*. Amsterdam, The Netherlands: IOS Press.

Harnad, S. 1990. The Symbol Grounding Problem. *Physica D* 42: 335-346.

Hall, J. 2007. *Beyond AI: Creating the Conscience of the Machine*. Amherst, NY: Prometheus Books.

Hauser, M. 2006. *Moral Minds: How Nature Designed Our Universal Sense of Right and Wrong*. New York, NY: HarperCollins/Ecco.

Hauser, M. et al. 2007. A Dissociation Between Moral Judgments and Justifications. *Mind&Language* 22(1):1-27.

- Hauser, M.; Young, Y. and Cushman, F. 2008. Reviving Rawls' Linguistic Analogy: Operative principles and the causal structure of moral actions. In Sinnott-Armstrong ed. *Moral Psychology and Biology*. New York, NY: OUP.
- Laird, J. 2008. Extending the Soar Cognitive Architecture. In *AGI 2008: Proceedings of the First AGI Conference*. Amsterdam, The Netherlands: IOS Press.
- Lebiere, C.; O'Reilly, R.; Jilk, D.; Taatgen, N. and Anderson, J.R. 2008. The SAL Integrated Cognitive Architecture. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Lehman, J.; Laird, J. and Rosenbloom, P. 2006. *A Gentle Introduction To Soar, An Architecture For Human Cognition: 2006 Update*. Available at <http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf>.
- Lenat, D.B. 1995. Cyc: a large-scale investment in Knowledge Infrastructure. *Communications of the ACM* 38(11): 33-38.
- Looks, M.; Goertzel, B. and Pennachin, C. 2004. Novamente: An Integrative Architecture for General Intelligence. In *AAAI Technical Report FS-04-01*. Menlo Park, CA: AAAI Press.
- Minsky, M. 2006. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. New York, NY: Simon & Schuster.
- Newman, J.; Baars, B.J.; Cho, S.B. 2003. A Neural Global Workspace Model for Conscious Attention. In Baars, B.J.; Banks, W.P.; and Newman, J.B. eds *Essential Sources in the Scientific Study of Consciousness*. Cambridge, MA: MIT Press.
- Oblinger, D. 2008. Towards an Adaptive Intelligent Agent. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Ohman, A.; Flykt, A.; and Esteves, F. 2001. Emotion Drives Attention: Detecting the Snake in the Grass. *Journal of Experimental Psychology: General* 130(3): 466-478.
- O'Reilly, R.C. and Munakata, Y. 2000. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. Cambridge, MA: MIT Press.
- Perlis, D. 2008. To BICA and Beyond: RAH-RAH-RAH! –or– How Biology and Anomalies Together Contribute to Flexible Cognition. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Petkova, V.I. and Ehrsson, H.H. 2008. If I Were You: Perceptual Illusion of Body Swapping. *PLoS ONE* 3(12): e3832.
- Pinker, S. 2007. *The Stuff of Thought: Language as a Window into Human Nature*. New York, NY: Viking/Penguin Group.
- Range, F.; Horn, L.; Viranyi, Z.; and Huber, L. 2008. The absence of reward induces inequity inversion in dogs. *Proceedings of the National Academy of Sciences USA* 2008 : 0810957105v1-pnas.0810957105.
- Rawls, J. 1971. *A Theory of Justice*. Harvard Univ. Press.
- Samsonovich, A.; De Jong, K.; Kitsantas, A.; Peters, E.; Dabbagh, N. and Kalbfleisch, M.L. 2008. Cognitive Constructor: An Intelligent Tutoring System Based on a Biologically Inspired Cognitive Architecture (BICA). In *AGI 2008: Proceedings of the First AGI Conference*. Amsterdam, The Netherlands: IOS Press.
- Sloman, A. 1999. What Sort of Architecture is Required for a Human-like Agent? In Wooldridge, M. and Rao, A.S. eds *Foundations of Rational Agency*. Dordrecht, Netherlands: Kluwer Academic Publishers.
- Soon, C.S.; Brass, M.; Heinze, H-J; and Haynes, J-D. 2008. Unconscious determinants of free decisions in the human brain. *Nature Neuroscience* 11: 543-545.
- Tomasello, M. 2008. *Origins of Human Communication*. Cambridge, MA: MIT Press.
- de Waal, F. 2006. *Primates and Philosophers: How Morality Evolved*. Princeton University Press.
- Waller, N.G.; Kojetin, B.A.; Bouchard, T.J.; Lykken, D.T.; and Tellegen, A. 1990. Genetic and environmental influences on religious interests, attitudes, and values: a study of twins reared apart and together. *Psychological Science* 1(2): 138-142.
- Wang, P. 2006. *Rigid Flexibility: The Logic of Intelligence*. Dordrecht, the Netherlands: Springer.
- Wang, P. 2008. What Do You Mean By "AI"? In *AGI 2008: Proceedings of the First AGI Conference*. Amsterdam, The Netherlands: IOS Press.
- Waser, M. 2008. Discovering The Foundations Of A Universal System Of Ethics As A Road To Safe Artificial Intelligence. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Yudkowsky, E. 2004. *Coherent Extrapolated Volition*. Available at <http://www.singinst.org/upload/CEV.html>.

Integrating Action and Reasoning through Simulation

Samuel Wintermute

University of Michigan
2260 Hayward St.
Ann Arbor, MI 48109-2121
swinterm@umich.edu

Abstract

This paper presents an approach for integrating action in the world with general symbolic reasoning. Instead of working with task-specific symbolic abstractions of continuous space, our system mediates action through a simple spatial representation. Low-level action controllers work in the context of this representation, and a high-level symbolic system has access to it. By allowing actions to be spatially simulated, general reasoning about action is possible. Only very simple task-independent symbolic abstractions of space are necessary, and controllers can be used without the need for symbolic characterization of their behavior. We draw parallels between this system and a modern robotic motion planning algorithm, RRT. This algorithm is instantiated in our system, and serves as a case study showing how the architecture can effectively address real robotics problems.

Introduction

It has been argued by many that low-level motor control and perception are critically important for intelligent behavior in embodied agents (e.g., Brooks, 1991). These views are often put in contrast with more traditional views of intelligence that emphasize symbolic processing (e.g., Newell, 1992). To date, adherents on either side haven't built robots that have exhibited anything close to general intelligence. Solely symbolic robots are rigid, and unable to deal with the subtlety of the real world, where every relevant perception needed to control action doesn't directly map onto a symbol. Robots without a symbolic level, however, aren't able to comprehend tasks anywhere near the level of complexity of those that humans perform, as those tasks can require reasoning at a much higher level than raw perception.

Attempts have been made to bridge deliberative symbolic reasoning and continuous reactive behavior. Many of these systems fit the general diagram in Figure 1a. In this sort of system, perception abstracts out information from the environment, and provides it to the symbolic system. The symbolic system reasons over this information, and invokes a controller with an abstract command. Following this command, the controller maps low-level input to output, causing action in the world. The perception box provides symbols that completely and concisely describe the environment, and invoking a particular action results in a predictable transition in these symbols. This is the classic blocks-world approach: the

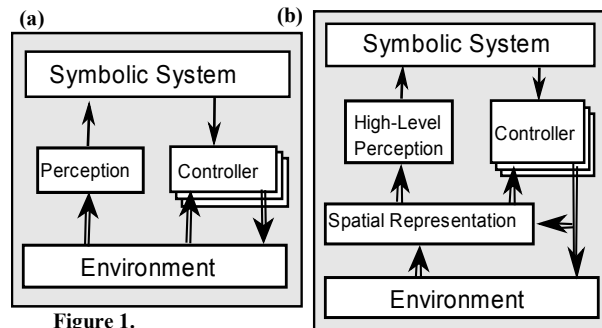


Figure 1.
a: Simple approach to integrating action and reasoning.
b: Our approach, where a spatial imagery layer mediates between the world and the symbolic system.

perception system provides symbols like (on a b), actions are represented by symbols like (move b table), and simple rules describe how the perception symbols will change in response to the action, enabling symbolic action planning.

Embodied AI systems have moved beyond this simple sort of system, but retain much of the flavor. In typical approaches, the core of the system deals with the world in terms of an abstract representation, and peripheral processes, considered as largely independent of the problem-solving part of the system, convert between the abstract representation and the world. This approach leads to systems where the perception and action pieces are designed together—each action corresponds to a straightforward transition in perception, so reasoning solely in terms of the abstract representation is possible.

We are investigating an alternative architecture that integrates action, perception, and reasoning in such a way that the generality of the overall system is increased. The key aspect of this system (Figure 1b) is that perception and action are mediated through a simple spatial representation. The interface between this representation and the symbolic system is generic and fixed. In addition to causing action in the world, controllers can simulate action in the spatial representation.

With this system, generality is enhanced in two ways compared to systems like those in Figure 1a. First, mediating action and perception through a spatial representation makes the system highly modular, allowing different low-level controllers or high-level strategies to be used without changing the rest of the system. Secondly, the need for complicated action-specific perception processes, as well as strategy-specific controllers, is greatly reduced.

Since reasoning does not take place entirely at the abstract symbolic level, perception does not have to provide enough information to completely describe the world, nor must controllers be designed such that commands result in predetermined transitions between abstract states.

This work is done in the context of the Soar cognitive architecture (Laird, 2008). Soar has been extended recently to handle problems involving space and motion, using specialized representations and mechanisms inspired by human perception and mental imagery. The extension to Soar is SVS (Spatial/Visual System, Wintermute and Lathrop, 2008). SVS provides the core of an action simulation and execution system.

To examine this system in detail, a case study will be used. Since research in Soar/SVS is moving towards robotics, a modern robotic motion-planning algorithm will be examined. The RRT algorithm (LaValle, 2006) has been used in many recent robots (e.g., Leonard et al., 2008), and works through simulating action. Looking at developments in the field of motion planning that led to RRT and other similar algorithms, we can provide evidence that simulation is an appropriate way to reason about action. We will instantiate RRT in Soar/SVS, showing how all of the parts of the system work together, and the generality this approach affords.

Simulation and Motion Planning

A key aspect of our approach to action and planning is that simulation is used for planning, rather than abstract symbol transformations. Our chief reason for doing this is that it leads to a more general system; however, there is increasing evidence from robotic motion planning research that regardless of generality concerns, simulation is an appropriate way to plan motion.

Motion planning research has usually been pursued outside of the context of creating generally intelligent systems. Earlier approaches focused on efforts to exactly compute optimal paths for particular classes of robots, such as polygon robots that can move in any direction. This involves computing exactly the configuration space of the robot, a space in which any path corresponds to a real path to robot is able to follow. As motion planning has progressed to address problems involving more and more realistic robots, however, this exact computation has become intractable (Lindemann and LaValle, 2003).

One reason for this difficulty is that certain kinds of constraints on motion are infeasible to capture in representations like configuration spaces. Nonholonomic constraints result from systems where the number of controllable dimensions is less than the total number of degrees of freedom. For instance, a car is nonholonomic, since its position can be described by three parameters (x , y , and an angle), but it is only controllable in two dimensions (driving forward and reverse, and steering left and right). Where it is relatively straightforward to calculate the configuration space of a robot that can turn in place, this is not as simple with a car-like robot. In addition

to nonholonomic constraints, traditional geometric motion planning approaches also have trouble incorporating dynamic constraints, where the path of the robot is affected by dynamics, such as a car that can't stop without slowing.

Recently, sampling-based approaches have become popular for planning with dynamic and nonholonomic constraints (LaValle, 2006). In sampling-based motion planning, the goal is not to exactly compute the configuration space, but instead to sample it through simulation. While previous approaches required difficult-to-calculate specialized representations that were specific to the particular motion under consideration, motion planning through simulation requires only a basic spatial representation, as details particular to the motion are encapsulated in the controller. If the simulation is accurate, motion plans can be guaranteed to meet nonholonomic and dynamic constraints.

This development from computation of configuration space to sampling reflects only two of many prominent techniques in motion planning. It is also worth mentioning behavior-based approaches, where representations are eschewed, and motion planning emerges from relatively simple mappings from perceptions to actions (e.g., Brooks, 1991). While our approach most certainly involves representations, it allows techniques developed in this tradition to be used as controllers within a broader symbolic AI system (in our case, the equations of Fajen and Warren, 2003, are used).

The RRT Algorithm

RRT (Rapidly-exploring Random Trees, LaValle, 2006) is a sampling-based motion planning algorithm that works by constructing a tree of states of the robot, rooted at the initial state, and adding nodes until that tree reaches the goal. Nodes are generated by extending the tree in random directions, in such a way that it will eventually reach the goal, given enough time. Each path from the root of the tree to a leaf represents a path that the robot could take, constantly obeying all constraints on its motion. The tree is constructed by the algorithm in Figure 2.

Two of the steps in this figure hide the true complexity of the algorithm. The steps to get the closest node to the random state, and to extend that node towards the random state, can both take substantial computation. This computation is also specific to the exact problem being solved, where the rest of the algorithm is general to all motion planning problems.

To determine the closest existing state to a random state, some metric must be determined that can measure the distance between states. In the case of car path planning, a simple metric is the Euclidian distance between the two

```
make tree rooted at initial state
while tree does not reach goal
  generate random state -> Xr
  get closest existing state to Xr -> Xc
  extend Xc towards Xr -> Xn
  if no collision occurred
    add Xn to the tree, connected to Xc
```

Figure 2. Basic RRT Algorithm

states, with the condition that the distance is infinite if the target state is not in front of the source.

The other problem-specific step in the algorithm is extending the chosen node towards the new state, while detecting collisions along the path. A typical approach is to numerically integrate differential equations to simulate motion, resulting in a sequence of states parameterized by time. This simulation must occur within a system capable of detecting collisions.

The Soar/SVS Architecture

The Soar cognitive architecture (Laird, 2008) provides the basis of our system. Work in Soar has traditionally focused on symbolic processing, and that remains a strength of the system. SVS builds on two prior lines of work on extending Soar to encompass non-symbolic spatial and visual representations, SVI (Lathrop, 2008), and SRS (Wintermute and Laird, 2008).

SVS encompasses the capabilities of both of these previous systems. Figure 3 shows the relevant parts of the system for this discussion. Compared to the generic diagram in Figure 1b, the names of some of the parts have changed, and two ways of adding objects to the scene from the top down have been added. SVS contains a long-term memory of the 3D structure of objects and the relationships between them, called the Perceptual LTM. By accessing this memory, Soar can recall objects or entire scenes to its Spatial Scene short-term memory as necessary. Sometimes, it is also necessary to add qualitatively described new objects to the scene, such as a line between two objects. This is called predicate projection (Chandrasekaran, 1998). A complementary process, predicate extraction, allows Soar to determine qualitative information about the contents of the scene, such as whether two objects intersect. This is the high-level perception box from Figure 1b, it contains the fixed processes through which Soar perceives the spatial world.

SVS also contains representations and mechanisms for dealing with 2D visual data, which are not used in this work. An external environment is also not used, all reasoning occurs in scenes retrieved from memory.

SVS and its predecessor systems have been used to study problems such as reasoning about positioning of military scouts to observe enemy units (Lathrop, 2008), and determining the layout of buildings in a real-time strategy game (Wintermute and Laird, 2008). Particularly

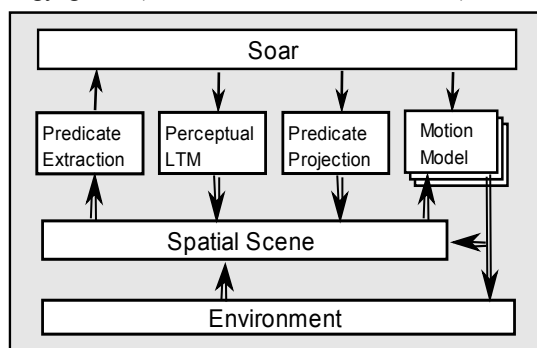


Figure 3. The SVS system.

relevant to this work, we have previously addressed reasoning with motion (Wintermute and Laird, 2008).

The interface between SVS and Soar is designed to work at a qualitative level (outside of a few exceptions). This means that information exchanged between Soar and SVS is expressed chiefly in terms of references to known items in one of SVS's memories and a library of fixed qualitative relationships. Detailed quantitative information, such as the continuous coordinates describing the polyhedrons in the scene, is inaccessible to Soar.

One way to add new objects to the SVS scene is by applying a known motion to an existing object. Motions are encapsulated as motion models. These allow stepwise simulations of particular motions to be invoked. To do this, the Soar agent specifics which motion model to apply, which existing object to move, and any other object parameters, such as a goal object. For example, the agent can apply motion **drive** to object **car**, moving towards object **goal**. SVS responds to this command by building an image of **car**, driving towards **goal**. The symbolic command also has a parameter for time—changing the time steps the motion forward.

Complex termination conditions are not declared to SVS beforehand, instead, the general capabilities that allow Soar to extract information from the scene are used. For example, Soar can query whether objects in the scene are intersecting. If the conditions for car-driving to terminate are that it has collided with an obstacle, Soar can query as to whether the moving car intersects with any obstacles, and stop updating the motion once this becomes true.

Motion models can be used to reason over any motion, including the agent's own actions, actions of others, and environmental movement. In the first case, the motion model will be the same as the actual controller for the motion, just simulating its output in the scene, instead of executing it in the environment. We will use the terms "motion model" and "controller" interchangeably when talking about actions for this reason.

Implementing RRT in Soar/SVS

In order to explore reasoning about action in our system, we have instantiated the RRT algorithm in a Soar agent. The problem we considered is that of planning to drive a car from an initial state to a goal region, while avoiding obstacles in a known environment. The car motion model takes as input the identity of a car in the scene, and the location of a goal. Inside this model, a system of differential equations that describe the configuration of a simple car-like vehicle as a function of the time and goal location is used. When integrated numerically, these equations yield a sequence of configurations, allowing for simulation. These equations were determined by combining a model of human movement (Fajen and Warren, 2003) with a simple car model (LaValle, 2006). The human model controls the intended steering angle of the car, and this steering angle determines the next position of the car. A constant speed is assumed.

The controller simulates motion towards a goal, while maintaining the nonholonomic constraints of the vehicle. Along with geometric models of the car and world in the LTM of SVS, it is the low-level knowledge that was added to the existing SVS system to implement this planner.

Symbolic Soar rules were written to perform the algorithm in Figure 2. The algorithm relies on existing architectural functionality in the interface between Soar and SVS. As a metric for node distance, our implementation used the Euclidean distance, with the condition that the distance is infinite where the goal is not “in front” of the node. SVS includes mechanisms for Soar to extract distances, and to query for an in-front relationship. The motion model described above enables simulation, and SVS supports querying for intersections between objects in the scene, enabling collision detection. The only new mechanism in SVS to support this algorithm was a method to generate random goal points in the scene, which was a simple addition.

Examples of the SVS scene during RRT planning are shown in Figure 4 (top). Soar stores, as a symbolic structure in its working memory, the RRT tree. The nodes in that tree are symbolic indexes into SVS—they point to specific objects in the scene, which can be seen in the figure. Soar proceeds by adding new random point objects to the scene, and querying for the distance from each node to that object, which are then compared to find the closest. A simulation is then instantiated with that node as the initial conditions (creating a new car object in the scene), this simulation is stepped until a certain time is reached, the goal is reached, or Soar detects a collision with an obstacle. In all but the last case, the termination of the simulation results in a new tree node being added. In addition to moving towards random points, with a certain probability the agent instead tries to extend the tree directly towards the overall goal, biasing the tree in that direction.

RRT Planning with Local Obstacle Avoidance

It is clear that the controller in the previous example leaves much room for improvement. The task is to avoid obstacles while driving towards the goal, but the controller knows nothing about obstacles. All obstacle avoidance happens by pruning the tree when simulations collide with obstacles.

As is done in Fajen and Warren (2003), the above controller can be enhanced to be biased to avoid obstacles, in addition to moving toward the goal. Each obstacle affects the steering of the car, with nearer obstacles located towards the front of the car having the most influence. This requires slightly more information to be provided to the controller: the identities of the obstacles. Other than that, the system outside of the controller remains unchanged.

With this new, smarter controller, performance is greatly increased. An example instance is shown in Figure 4 (bottom), note that the number of states in the RRT tree before a solution is found is much less than in the top of the figure. Over 100 trials of the scenario presented in Figure 4, the average number of individual simulations needed to solve the problem was 128 without obstacle avoidance and only 12 when it was present.

Discussion

These case studies exemplify how reasoning and action can be integrated through simulation. In general, problems are decomposed between high-level algorithmic knowledge, instantiated in the symbolic system, and low-level action knowledge, instantiated as motion models. There is a spatial representation allowing real and imaginary situations to be instantiated, and fixed processes which communicate between that representation and the symbolic level. Complex reasoning about action occurs through interplay of low-level simulation and high-level symbolic reasoning over simulation results, detecting spatial interactions and making decisions accordingly.

The interface between the spatial and symbolic levels is the only means by which symbolic processing in Soar can obtain information about the results of lower-level processing. Soar uses the Predicate Extraction system in Figure 3 to query the scene for information such as which objects intersect which, relative directions, and other simple qualitative relationships. The commitment to make this system fixed is important, as it implies that the type of qualitative spatial information available to the symbolic system is the same, regardless of the task. The poverty conjecture of Forbus et al. (1991) states that “there is no purely qualitative, general-purpose, representation of spatial properties”, and if this is true, the decision to use a fixed representation, calculated without task knowledge, seems a poor choice.

However, this conjecture might not apply to our system, as the qualitative representation used does not have to be rich enough to enable purely qualitative reasoning (and hence is not really “general purpose”). The essential role of simulation with respect to symbolic reasoning is that of an

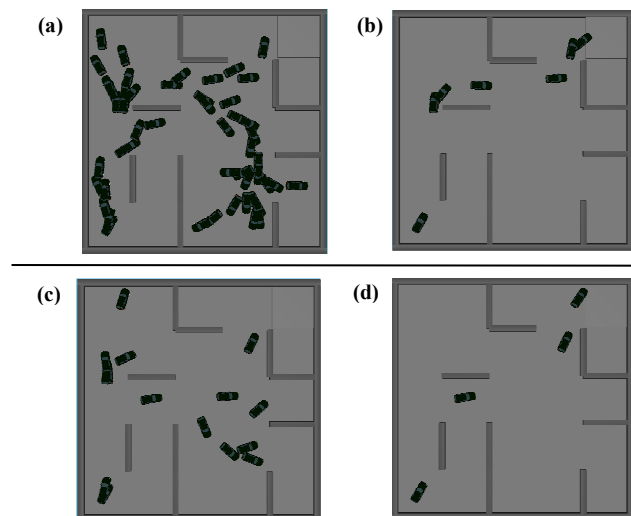


Figure 4. States of SVS Spatial Scene during RRT planning. The problem is to drive a car from lower-left to upper-right.
(a). RRT tree for goal-seeking controller, just before a solution is found.
(b). Sequence of car positions that solve the problem.
(c), (d). Same, for obstacle-avoiding controller.

action model. The system is able to predict the symbolic consequences of symbolically described actions—for example, inferring that driving a certain car object toward the goal leads to a collision. Since simulation is available, the qualitative representation used needs to be only rich enough to capture important aspects of the current state of the spatial system, not the implications of action.

The alternative to this is to represent the consequences of actions completely symbolically, as in planning systems such as STRIPS. Consider a problem in a variant of the blocks world. In this problem, a robot finds itself in front of a table with blocks on it, and must stack them in a certain way (e.g. A on B, and C on A). Each time it encounters a table, the robot's perception system must encode the problem in such a way that planning can be performed. This encoding must capture both the states of the world (such as $on(B,C)$ and $clear(C)$), and the way those states are transformed by action. The robot can make certain assumptions about any table it encounters. For example, if a block is clear (has nothing on top of it), it can be moved to the top of a different clear block. However, in a divergence from the standard blocks world, the table is not always clear—in this case, the table is divided into numbered bins, and blocks must be placed entirely inside a bin. Not all of the bins are wide enough to fit a block, though, and the block will fall to the floor if the agent places it there. So depending on the exact table the robot has encountered, the result of moving block A into bin 3 is either $inBin(A, bin3)$, or $onFloor(A)$.

To allow purely symbolic planning in this domain, the perception system of the robot must then compare each bin's size to the width of a block, and generate its state transition rules accordingly. In this way, creating the qualitative representation of the problem requires very task-specific calculations to be performed in the perception system in order for that representation to capture the consequences of actions. In contrast, if actions can be simulated in a spatial representation, this can instead be accounted for by simulating placing a block in the bin, and simply checking if it intersects a wall

While there is no proof that the symbolic representations used in Soar/SVS are general enough that all reasonable tasks can be represented, it is clear that the symbolic representation needed when actions can be simulated is much simpler than the representation needed to solve problems entirely at the symbolic level. The high-level perception system that builds this representation can then be much simpler if simulation is possible.

Similarly, the ability to simulate actions simplifies the requirements of the low-level action system. Consider again the standard planning approach to a problem like those in blocks world. In order to represent the problem completely symbolically, the effect of every possible action on every possible state must be characterized before the problem can be solved. In addition to increasing the need for task-dependant perception, as discussed above, this requires that the controllers used by the system have simple-to-characterize guarantees on their performance.

For instance, in blocks world, it is assumed that the controller can reliably perform an action like moving a block from the top of one stack to another for any two stacks present. If actions can instead be simulated, these guarantees are unnecessary—the behavior of the controller does not need to be symbolically characterized before reasoning can begin.

To further illustrate this point, contrast the RRT agent with obstacle avoidance described above with a more traditional graph-search approach. If a robot can be approximated by a circle and can turn in place, concise representations of relevant locations for the robot to move in the world can be derived. Using this representation, a graph of nodes and distances between them, optimal path planning can be done using an algorithm like A*. If this graph is the robot's perception, and its actions move between nodes, reasoning about action is a simple graph search. Consider the complication involved in using this form of perception and action, though. Perception involves computing the configuration space of the robot, and paths within it that lie entirely outside of obstacles. Actions must map directly onto transitions between nodes of the graph, so a controller must be used that guarantees accurate execution of these transitions.

In contrast, the RRT planner needs only very simple perceptions, such as which objects intersect which others, and the general direction and distance between objects. The actions of the robot do not have to be symbolically characterized beforehand for the planner to work. For a controller like the obstacle-avoiding car used above, it would be very hard to characterize in what cases it is able to guide the robot to the goal and in what cases it isn't.

Ignoring other concerns, such as the limited applicability of graph-search planning for more complicated robots, and the suboptimal performance of RRT for simple robots, it is clear that integrating reasoning and action through simulation requires simpler perception and less-constrained action systems than reasoning solely at the symbolic level.

This simplification of perception and action systems afforded by the use of simulation allows for a highly modular overall system. The interface between the symbolic system and spatial representation, which involves complicated transformations between symbolic and quantitative representations, is fixed; new capabilities can be added to the system without modifying this interface. For example, a different approach to path planning has been previously implemented in Soar/SVS (Wintermute and Laird, 2008). In this case, the problem was approached by detecting which obstacles lie between the car and the goal, and deliberately creating waypoints in the scene to divert around them. This is a very different overall algorithm than RRT, but the agent itself differs only in the high-level Soar rules that describe it. Changes in the interface between Soar and the spatial system were not needed, nor were changes in the controller.

This fixed symbolic/quantitative interface also allows for modularity in the controllers used. This is seen in the above examination of the RRT planner, where an obstacle-

avoiding controller was substituted for a simpler goal-seeking controller, while requiring minimal changes to the rest of the system. Controllers in the system take the majority of input and create output in terms of spatial objects. Their interface to the symbolic system simply points them to the relevant objects in the scene and provides time. Controllers can be built which work internally with continuous numbers, the same format in which the scene is represented, so there is no need for a difficult representation conversion at the controllers input and output like there might be if the controller interfaced only to the symbolic system.

Related Work

The integration of reactive behavior and deliberative reasoning in robotics architecture has been investigated, such as by Arkin (1989) and many others. However, most of these systems involve using deliberative reasoning to serialize different reactive behaviors (thus fitting Figure 1a), and do not involve simulation. Some previous systems do use simulation of reactive behavior to guide reasoning. MetaToto (Stein, 1994) emphasizes this, but doesn't focus on enabling general high-level reasoning as our system does. 4D/RCS (Albus, 2003) includes simulation ability, but is more of a scheme for organizing an agent, rather than a commitment to a specific set of components.

From an AI point of view, this work inherits much from research in diagrammatic reasoning (e.g., Chandrasekaran, 1997). Comirit (Johnston and Williams, 2008) is also a similar approach to ours, integrating logic with physical simulation, but is not used for action and doesn't have a spatial representation. Our use of a spatial representation to simplify symbolic reasoning builds on previous work looking at the frame problem (Huffman and Laird, 1992).

Conclusion

We have shown that Soar with SVS is able to use imagery to solve motion planning problems. This was done using the existing fixed functionality of SVS to communicate between Soar and the scene, and adding symbolic knowledge to Soar encoding the high-level RRT algorithm and low-level knowledge in the form of a controller. While this system has not yet been implemented on a real robot, RRT has (e.g., Leonard et al., 2008), so some version of this approach is feasible with current technology.

This system is presented as an example of integrating action and symbolic reasoning through spatial simulation. This allows the symbolic system to reason about the problem, but removes the requirement that the entire problem be represented symbolically. Because of this, the perception system needed is simpler and more problem-independent than would otherwise be required, and controllers can be used without symbolically characterizing them beforehand. A very modular system is then possible, since symbolic and control processing are mediated by the spatial system and its fixed qualitative/spatial interface. While many existing systems use major elements of this

approach, this work has attempted to make explicit the argument for how and why this form of integration is a step on the path toward more general AI systems.

Acknowledgements

John Laird provided guidance and support on this project, and assisted in editing. Joseph Xu and Nicholas Gorski provided useful conversations in formulating the ideas behind this paper. This research was funded by a grant from US Army TARDEC.

References

- Albus, J.S., 2003. 4D/RCS: A reference model architecture for intelligent unmanned ground vehicles. In *Proceedings of SPIE*.
- Arkin, R.C., 1989. Towards the unification of navigational planning and reactive control. In *AAAI Spring Symposium on Robot Navigation*. Stanford, CA.
- Brooks, R.A., 1991. Intelligence without representation. *Artificial Intelligence*, 47, 139-159.
- Chandrasekaran, B., 1997. Diagrammatic representation and reasoning: some distinctions. In *AAAI Fall Symposium on Diagrammatic Reasoning*. Boston, MA.
- Fajen, B.R. & Warren, W.H., 2003. Behavioral dynamics of steering, obstacle avoidance, and route selection. *J. Experimental Psychology: Human Perception and Performance*, 29(2).
- Forbus, K.D., Nielsen, P. & Faltings, B., 1991. Qualitative spatial reasoning: the CLOCK project. *Artificial Intelligence*, 51(1-3).
- Huffman, S. & Laird, J.E., 1992. Using Concrete, Perceptually-Based Representations to Avoid the Frame Problem. In *AAAI Spring Symp. on Reasoning with Diagrammatic Representations*.
- Johnston, B. & Williams, M., 2008. Comirit: Commonsense Reasoning by Integrating Simulation and Logic. In *Proc. First Conference on Artificial General Intelligence*.
- Laird, J.E., 2008. Extending the Soar Cognitive Architecture. In *Proc. First Conference on Artificial General Intelligence*.
- Lathrop, S.D., 2008. *Extending Cognitive Architectures with Spatial and Visual Imagery Mechanisms*. PhD Thesis, University of Michigan.
- LaValle, S.M., 2006. *Planning Algorithms*, Cambridge U. Press.
- Leonard, J. et al., 2008. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10)
- Lindemann, S.R. & LaValle, S.M., 2003. Current issues in sampling-based motion planning. In *Proceedings of the International Symposium of Robotics Research*. Springer.
- Newell, A., 1990. *Unified theories of cognition*, Harvard University Press Cambridge, MA.
- Stein, L.A., 1994. Imagination and situated cognition. *Journal of Experimental and Theoretical Artificial Intelligence*, 6.
- Wintermute, S. & Laird, J.E., 2008. Bimodal Spatial Reasoning with Continuous Motion. In *Proceedings of AAAI-08*. Chicago.
- Wintermute, S. & Lathrop, S.D., 2008. AI and Mental Imagery. In *AAAI Fall Symposium on Naturally Inspired AI*.

Neuroscience and AI Share the Same Elegant Mathematical Trap

Tsvi Achler, Eyal Amir

University of Illinois at Urbana-Champaign
201 N. Goodwin Ave, Urbana IL 61801, USA

Abstract

Animals display exceptionally robust recognition abilities to analyze scenes compared to artificial means. The prevailing hypothesis in both the neuroscience and AI literatures is that the brain recognizes its environment using optimized connections. These connections are determined through a gradual update of weights mediated by learning. The training and test distributions can be constrained to be similar so weights can be optimized for any arbitrary pattern. Thus both fields fit a mathematical-statistical framework that is well defined and elegant.

Despite its prevalence in the literature, it remains difficult to find strong experimental support for this mechanism within neuroscience. Furthermore this approach is not ideally optimized for novel combinations of previously learned patterns which typically form a scene. It may require an exponential amount of training data to achieve good precision.

The purpose of paper is to 1) review the difficulties associated with this approach in both neuroscience experiments and AI scenarios. 2) Direct the reader towards 'less elegant' mathematically-difficult inherently nonlinear methods that also address both literatures (better optimized for scenes and emulating experiments) but perform recognition without optimized weight parameters.

Introduction

Though modern day studies reveal important information about the brain, for example, which regions of the brain become active, the underlying circuits are still unknown.

The unparalleled robustness of brain processing serves as a motivation for AI. Hebb's seminal work led to the phrase 'what fires together wires together'. Since then, the predominant hypothesis is that the brain performs computations based on optimized connections determined by learning-dependent synaptic plasticity. It hypothesizes small incremental and distributed changes in the networks during learning of connection weights. The networks are not dependent on single individual connections and as a biological model are resistant to injury of single units. Moreover, the networks can learn any arbitrary function as long as the training distribution is similar to the test distribution. This requirement also facilitates probabilistic analysis, tying into probability theory and well-defined mathematical analysis. This approach elegantly connects neuroscience, probability theory and a practical ability to learn any arbitrary pattern. Subsequently it is the fundamental building block for the fields of machine learning and AI.

Searching Under the Streetlight

Despite the elegance of this mechanism, two problems persist: 1) The AI methods do not seem to scale to brain function. 2) Synaptic plasticity mechanisms are still mysterious after over a half-century of experiments.

Neuroscience. Synaptic plasticity in the context of learning is the most studied phenomenon in neuroscience. A search in the Pubmed database reveals 13,000 publications.

Synaptic plasticity is assumed to occur whenever a long-lasting change in communication between two neurons occurs as a consequence of stimulating them simultaneously. The changes are labeled Long Term Potentiation (LTP) or Long Term Depression (LTD), if the responses increase or decrease respectively.

Understanding these experiments' motivation and design enables the understanding of the variability of results.

In the simplest scenario, two electrodes are introduced into two neurons within brain tissue and activation of one electrode (neuron) will cause a response in the other (note: electrodes may be placed in numerous neurons before an appropriate pair is found). In these experiments, the electrode that is used to stimulate a response is the input electrode and the electrode that records the response is the output electrode. Once this configuration is found, an activation protocol is followed. Stimulation of the input electrode is adjusted until the output neuron fires 50% of the time. The electrical settings of the input electrode that satisfy this characteristic are labeled as the *Amplitude of 50%* (A_{50}).

'Synaptic change' is induced by stimulating the electrodes simultaneously and in rapid succession. After induction is complete, A_{50} is applied again to the input and any changes in the output electrode are observed. If the output activity is greater than the original 50% then LTP occurred. If the output activity is less than 50% then LTD occurred. These changes can last for hours or days (as long as the experimental preparation is viable).

A problem with this experimental design is that the high-frequency stimulation of Long Term Potentiation induction can affect many neurons, yet their contribution to the LTP phenomena is rarely considered. The synaptic plasticity hypothesis assumes that the only important interaction is between the input and output neurons. However this is highly variable suggesting a multi-cell mechanism. In experiments in brain regions that involve sensory processing, memory and logic, there are always more neurons present than a single input and output neuron.

Thus it is not surprising that it is difficult to determine under what induction patterns LTP occurs and under what induction patterns LTD occurs. Some studies find that if the input electrode spikes are activated within tens of milliseconds before the output electrode spikes, LTP is induced. The reversed order of firing results in LTD. In other studies, the electrode of the first spike or the last spike can determine the LTP or LTD and their magnitude. Yet other studies show that modification is frequency dependent. High-frequency bursts of spikes lead to LTP, regardless of the relative input-output electrode timing. Even other studies show that spikes are not necessary for the induction of LTP and LTD (Froemke, Tsay et al. 2006). Furthermore, these properties may change during development. Thus the criteria just to induce LTP or LTD are unclear.

In summary, reliable activity-dependent plasticity relations have not been determined in sensory regions, let alone more complex learning algorithms. Clearly, robust learning occurs in the brain. However learning rules that are based on connection weights and able to learn any arbitrary pattern may not be warranted biologically or even beneficial functionally.

AI. Learning algorithms require the training distribution and the test distribution to be the same. This limits AI because the distribution is violated in the natural environment such as a scene with many patterns. If a network is trained on pattern *A* and *B* separately and they are presented side-by-side simultaneously, this is outside the training distribution. Because of distribution limits, every pair of patterns possible (or triplets, quadruplets, etc.) must be trained. This type of problem is inescapable because it occurs with objects embedded within other objects (another common natural occurrence). This combinatorial explosion of training is long known as the 'superposition catastrophe problem' (Rosenblatt 1962; Rachkovskij and Kussul 2001). This problem is mostly due to the fundamental distribution assumptions in learning and has important implications for robustness. Thus, in order to implement AGI training requirements should be relaxed, and alternatives to connection parameter optimization explored. AGI algorithms should make intelligent structured inference onto different distributions.

Steps Forward

An example of a network that is less elegant in terms of solvability, but more powerful in handling situations outside of its training distribution is a self-regulatory feedback network (Achler, Omar et al. 2008).

This method was deduced from biological studies showing an overwhelming preponderance of feedback inhibition. Self-regulatory feedback, inhibitory feedback from outputs back to inputs (also known as pre-synaptic inhibition in neuroscience and negative feedback in engineering) are found in same numbers as feed-forward connections in the brain, especially in sensory processing regions.

Its functional characteristics are difficult to derive analytically (such connections are inherently nonlinear) but

are better optimized for novel combinations of previously learned patterns, scenes (Achler, Omar et al. 2008).

Though connections are determined by supervised learning, they are not trained in a conventional sense (i.e. through parameter optimization) since there are no connection weights to optimize.

However, this network implements a surprisingly robust multiclass classifier that can process simultaneous patterns (Achler, Omar et al. 2008) addressing the superposition catastrophe problem. Furthermore the networks make complex recognition decisions based on distributed processing (Achler and Amir 2008) addressing components of the binding problem. This structure requires minimal resources (less parameters) and training.

As importantly, given the scenario of an LTP experiment, the network behaves in a similar manner to that predicted by activity-dependent synaptic plasticity. However, no connection changes are needed (Achler 2008). This approach demonstrates that by relaxing training and distribution requirements there may be more benefits than difficulties.

Conclusion

To move forward towards AGI and better understanding of brain circuits, researchers must be willing to trade elegant mathematics for nonlinear methods that are difficult to solve, but function beyond the distribution limits.

Acknowledgements

Michelle Madonia. U.S. National Geospatial Agency Grant HM1582-06--BAA-0001.

References

- Achler, T. (2008). Plasticity Without the Synapse: A Non-Hebbian Account of LTP. Society for Neuroscience. Washington DC.
- Achler, T. and E. Amir (2008). "Input Feedback Networks: Classification and Inference Based on Network Structure." Artificial General Intelligence 1: 15-26.
- Achler, T., C. Omar, et al. (2008). "Shedding Weights: More With Less." Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IJCNN'08).
- Froemke, R. C., I. A. Tsay, et al. (2006). "Contribution of individual spikes in burst-induced long-term synaptic modification." J Neurophysiol 95(3): 1620-9.
- Rachkovskij, D. A. and E. M. Kussul (2001). "Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning." Neural Computation 13(2): 411-452.
- Rosenblatt, F. (1962). Principles of neurodynamics; perceptrons and the theory of brain mechanisms. Washington,, Spartan Books.

Relevance Based Planning: Why Its a Core Process for AGI

Eric B. Baum

Baum Research Enterprises
41 Allison Road
Princeton NJ 08540
ebaum@fastmail.fm

Abstract

Relevance Based Planning (RBP) is a general method that plans in interaction with a domain simulation and domain specialized procedures. I argue that exploitation of the properties of causality and Euclidean topology which hold in many domains is a critical inductive bias necessary if an AGI (or any intelligent program) is to generalize to new problems and new domains, and is critical to human thought, and that RBP achieves its efficiency by exploiting these properties in a novel and powerful way, faithful to introspection in an example task. RBP is proposed to be implemented as a scaffold within an AGI or a CAD tool for producing intelligent programs, that is to say as a general library procedure that takes as inputs domain specialized procedures such as a domain simulation and procedures for recognizing and dealing with problems within the simulation. Supplied with such procedures as inputs, the RBP scaffold provides a framework that orchestrates plan formation and refinement in such a way that only causally relevant configurations are considered.

Introduction

The critical question in AGI is generalization: how one can generalize to solve new problems never before seen. Consider figure 1. You may never have seen this Rube Goldberg image before, but given a few minutes you can work out how and whether it works, and if there were a problem could suggest a fix. More generally, you can learn in a few minutes or years, to solve generic problems in whole new domains, for example new games or new employment. The learning is remarkably fast, given the complexity of the endeavor, and once one has learned, the solution often happens in real time. Each such new domain (or, arguably, new problem) requires rapidly constructing a novel, powerful program within your mind and executing it to solve new problems. I suggest that the only way this is possible is because you exploit in constructing and applying these mental programs certain underlying structure and properties of the domains (causality, Euclidean 3-D topology, etc); for example to analyze the Rube Goldberg device, that you construct and run a mental simulation of the system, and that doing this requires retrieving and rapidly putting together special purpose *procedures* that know how (or at least roughly how) to exploit causality and apply mental simulations, and procedures to analyze local structure, for example methods

to simulate bird flight or ball rolling. The solving process follows causal chains, and may never bother to analyze the faucet because causal chains never reach it. An extensive literature indicates that human reasoning is model based (cf (Johnson-Laird 1983).

I further conjecture (a) that a program to perform such feats is naturally composed of scaffolds that take specialized procedures as arguments, for example a scaffold for handling the causal propagation and the interaction with a simulation domain, that takes as arguments procedures such as for recognizing and dealing with specific kinds of obstacles in specific domains. If one achieves a breakup like this, into general scaffold and specialized¹ procedures as inputs, then one can imagine gaining a concisely specified program that can generalize to rapidly construct programs dealing with new circumstances. And I conjecture (b) that the structure of such scaffolds can be informed by introspection.

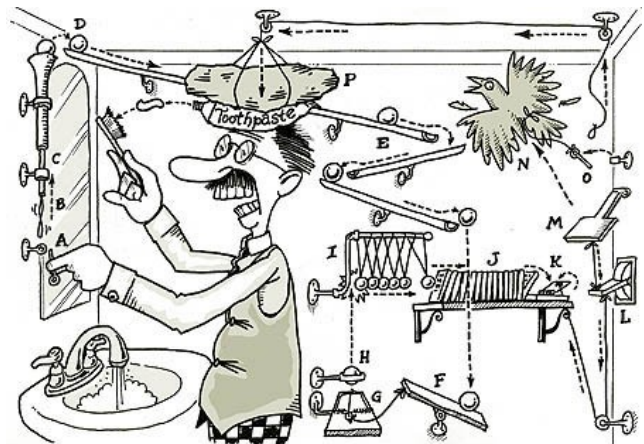


Figure 1: A Rube Goldberg Device

Relevance Based Planning(RBP) is a procedure that attempts to reproduce these aspects of thought. It is unlike any planning algorithm that I am aware of in the literature in its use of a simulation model and special purpose

¹I refer to these as domain specialized to reflect the they might be adapted or inherit from more general procedures, which would further facilitate learning and generalization to new domains.

procedures for simulating or dealing with local structures, and in its fidelity to introspection at least in examples of Sokoban where I have experimented. It was designed by initially introspecting how I solve simple problems within Sokoban and then abstracting the procedure to a high level process that handles the causality, use of simulation domain, and backup when problems are encountered, by use of domain specific procedures recognizing and dealing with various kinds of domain specific objects.

Here is some high level pseudo-code for RBP:

- (1) Find high level plans by search or dynamic programming over a simulation model, in which the search is over sequences of domain-appropriate operations, for example plans that allow counterfactual steps (steps that apply an action or stored procedure that would be applicable only if some obstacle in the simulation model may be corrected, where the obstacle is of a type that is known may be remediable). A high level plan then consists of a sequential-in-time series of subgoals (or obstacles to be overcome) that if solved should result in a solution.
- (2) Refine one of these candidate plans in time order.
- (3) As each obstacle in the plan is considered, invoke a (specialized) method that attempts to deal with the obstacle. The method typically performs some search on the simulation domain (and may invoke or even recursively call RBP).
- Such calls to clearing methods typically pass information about higher level goals within the plan, and the called clearing method may then avoid searching a set of actions to remove the obstacle that would have as prerequisite previously achieving a higher level goal.
- (4) If the search encounters other problems that would require earlier actions, (problematic configurations in the simulation domain that are perceived by agents for recognizing them) insert earlier in the plan an attempt to perform those actions first (typically by invoking a stored method for dealing with that kind of obstacle).
- (5) When changes are made in the domain simulation, mark them, so that if they encumber later actions, you can back up and try to insert the later actions first to avoid the problem.
- (6) Utilize a method of backing up to other high level plans when it is discovered that a high level plan can not be made to work, or of switching between high level plans as more information is uncovered about them.

RBP illustrates the power of using a domain simulation. It forms a high level plan over the simulation. Then it analyzes it in interaction with the simulation. As modules are executed to solve problems, the interaction with the simulation creates patterns that summon other modules/agents to solve them. This all happens in a causal way: things being done on the simulation cause other things to be done, and because the simulation realizes the underlying causal structure of the world, this causal structure is grounded. That is, it corresponds to reality. The RBP framework also focuses the search to consider only

quantities causally relevant to fixing problems.

Many planning methods choose not to work in time order for various good reasons (Russell and Norvig 375-461). But by working on a candidate plan in time order, RBP is assured, at the inner most loops of the program, of only spending time considering positions that can be reached, and which are causally relevant to a high level plan. Introspection sometimes works out of time order on causally local regions of a problem, which are later interleaved in a causal (and time-sequential) fashion. As discussed in (Baum 2008a) this can be to an extent handled within RBP at a higher level.

(Baum 2008a) describes RBP in more detail, giving a step by step walk-through of a particular example of the application within the domain of Sokoban, and another example (at a higher level goal within Sokoban) is sketched. To formalize the pseudo-code to any given domain or problem within a domain, requires supplying various procedures that say what the obstacles are and how they are dealt with, what deadlock configurations look like, what counter-factuals are permitted, etc. In complex situations it is expected that some of these will be too hard to be hand coded, and will instead be produced by learning from examples using module constructors such as Evolutionary Economic Systems (Baum, 2008b).

If an RBP scaffold is provided, it can be made accessible to module constructors within an AGI or CAD tool, so that new programs can be automatically constructed, evolved, or learned that invoke the RBP. An example where an RBP planner was used in this way was given in (Schaul 2005).

A larger meta-goal of the project described in (Baum 2008b) is to construct Occam programs, programs that are coded in extremely concise fashion so that they generalize to new problems as such problems arise (or so that the programs can be rapidly modified, say by a search through meaningful modifications, to solve new problems). Search programs can be incredibly concisely coded, so that coding as an interaction of a number of search programs can be a mode of achieving great conciseness. RBP's construction of a program by composing a series of feasible-sized goal oriented searches mirrors the basic structure of biological development (cf (Baum 2008b)), and is similarly concisely coded and robust.

References

- Baum, Eric B. 2008a. *Relevance Based Planning: A Worked Example*. <http://www.whatisthought.com/planning.pdf>.
- Baum, Eric B. 2008b. *Project to Build Programs That Understand*. Proceedings of AGI09 (this volume) <http://www.whatisthought.com/agipaper091.pdf>.
- Johnson-Laird, P. 1983. *Mental Models*. Harvard University Press, Cambridge Ma.
- Russell, S. and P. Norvig 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Saddle-River NJ.
- Schaul, T. 2005. *Evolution of a compact Sokoban solver*. Master Thesis, École Polytechnique Fédérale de Lausanne. posted on <http://whatisthought.com/eric.html>

General Intelligence and Hypercomputation

Selmer Bringsjord

Department of Cognitive Science
Department of Computer Science
Rensselaer Polytechnic Institute (RPI)
Troy NY 12180 USA
selmer@rpi.edu

From its revolutionary arrival on the behaviorism-dominated scene, the information-processing approach to both understanding human intelligence, and to the attempt to bestow such intelligence on a machine, has always proceeded under the assumption that human cognition is fundamentally computation (e.g., see (vE95) in connection with CogSci, and (Hau85) in connection with standard AI). But this approach can no longer leave it at that; it can no longer rest content with the coarse-grained creed that its allegiance is to the view that intelligence consists in processing information. There are two reasons: First, AGI is now on the scene, insisting, rightly, that *general* intelligence should be the focus. Second, ours is an age wherein the formal science of information processing takes quite seriously a detailed mathematical framework that generates a difficult and profound question: “What kind of information processing should those interested in general intelligence take the mind to rest upon, some form of standard, Turing-level computation, or that and *hyper*computation?” My question is not to be confused with: “Which form of Turing-level computation fits best with human cognition?” This question has been raised, and debated, for decades.¹ A recent position on this question is stated by (LEK⁺06), who argue that Turing-level neurocomputation, rather than Turing-level quantum computation, should be the preferred type of information processing assumed and used in cognitive science.

Recall that computation is formalized within the space of functions from the natural numbers $N = \{0, 1, 2, \dots\}$ (or pairs, triples, quadruples, ... thereof) to the natural numbers; that is, within

$$\mathcal{F} = \{f | f : N \times \dots \times N \longrightarrow N\}.$$

This is a rather large set. A very small (but infinite) proper subset of it, \mathcal{T} (hence $\mathcal{T} \subset \mathcal{F}$), is composed of functions that Turing machines and their equivalents (Register machines, programs written

in modern-day programming languages, the λ calculus, etc.; a discussion of these and others in the context of an account of standard computation can be found in Bringsjord (Bri94)) can compute; these are the *Turing-computable* functions. For example, multiplication is one such function: it’s laborious but conceptually trivial to specify a Turing machine that, with any pair of natural numbers m and n positioned on its tape to start, leaves $m \cdot n$ on its tape after its processing is complete. Neurocomputation, as defined by the literature (LEK⁺06) cite, is Turing-computable computation. If the mind/brain merely neurocomputes, then it can’t compute any of the functions in \mathcal{F} that are not in \mathcal{T} . (If we let \mathcal{N} denote those functions that neurocomputation can handle, we have that $\mathcal{N} = \mathcal{T}$.) The overwhelming majority of functions in \mathcal{F} would thus be beyond the reach of human persons to compute.

The situation is the same no matter what type of standard computation one selects. For example, those inclined to favor traditional symbolic computation aligned with first-order logic (over, say, connectionism), are at bottom using standard Turing machines. For a more exotic example, consider quantum computers: Standard quantum computers, first introduced by (Deu85), can only compute the functions in \mathcal{T} , but as some readers will know, some of this quantum computation can be surprisingly efficient. However, the efficiency of a machine is entirely irrelevant to the class of functions it can compute. All those functions commonly said to be intractable, such as the class of NP-complete functions, are in \mathcal{T} . The truly intriguing quantum computers would be those capable of hypercomputation. At the moment, it remains an open question as to whether some *recherché* forms of quantum computation can compute Turing uncomputable functions. So, where $\mathcal{Q} \subset \mathcal{F}$ contains the functions computable by standard quantum computers, we have $\mathcal{Q} = \mathcal{T}$.

Hypercomputation is the computing, by various extremely powerful machines, of those functions in \mathcal{F} that are beyond the so-called *Turing Limit*; i.e.,

Copyright © 2009, The Second Conference on Artificial General Intelligence (AGI-09.org). All rights reserved.

¹E.g., see (Bri91).

those functions (composing \mathcal{H}) in \mathcal{F} that aren't in \mathcal{T} . The mathematics of hypercomputation is now quite developed; the machines, definitions, and theorems in question are elegant and informative (e.g., see (SS94; Sie99; EN02; Cop98; HL00; BKS⁺06; BZ03)).

The evidence that human persons hypercompute comes in two forms: abstract and empirical.² The empirical evidence, in short, consists in the brute fact that the following prophecy of Descartes still stands.

If there were machines which bore a resemblance to our body and imitated our actions as far as it was morally possible to do so, we should always have two very certain tests by which to recognize that, for all that, they were not real men. The first is, that they could never use speech or other signs as we do when placing our thoughts on record for the benefit of others . . . And the second difference is, that although machines can perform certain things as well as or perhaps better than any of us can do, they infallibly fall short in others, by which means we may discover that they did not act from knowledge, but only for the disposition of their organs. For while reason is a universal instrument which can serve for all contingencies, these organs have need of some special adaptation for every particular action. From this it follows that it is morally impossible that there should be sufficient diversity in any machine to allow it to act in all the events of life in the same way as our reason causes us to act. ((Des11), p. 116)

The advent of AGI heralds an epoch in which the information-processing approach to intelligence boldly confronts what AI *simpliciter* has for the most part gradually retreated from: capturing *general* intelligence. In the face of Descartes' claim, this spells surrender for AI, and an acute challenge for AGI. It's a brute fact that human cognizers, in the logico-mathematical realm, conceive, manipulate, reason over . . . the space \mathcal{H} ($\mathcal{F} - \mathcal{T}$) above what Turing machines and their equivalents can muster. Were this not happening, we would not have the mathematics of hypercomputation summarized above, the first part of which was discovered in 1965, when one of the first hypercomputing machines (*trial-and-error machines*) were specified (Gol65; Put65). In this activity, the humans in question use formal schemes that cannot even be directly represented in any of the languages Turing machines and their equivalents are restricted to using. Many AI-niks will doubtless hold that in the future their field will discover how to re-express these highly expressive formal schemes, without loss of meaning, in some standard, austere format used to specify Turing-level computation. But AGI researchers, on the other hand, may choose instead

²Interested readers can assess some of the abstract evidence presented in (BA04; BKS⁺06).

to press forward in the hope of devising *new* formalisms and techniques up to the challenge of the 'G' in the acronym for their field.

References

- [BA04] Selmer Bringsjord and Konstantine Arkoudas. The modal argument for hypercomputing minds. *Theoretical Computer Science*, 317:167–190, 2004.
- [BKS⁺06] Selmer Bringsjord, Owen Kellett, Andrew Shilliday, Joshua Taylor, Bram van Heuveln, Yingrui Yang, Jeffrey Baumes, and Kyle Ross. A new Gödelian argument for hypercomputing minds based on the busy beaver problem. *Applied Mathematics and Computation*, 176:516–530, 2006.
- [Bri91] S. Bringsjord. Is the connectionist-logicist clash one of AI's wonderful red herrings? *Journal of Experimental & Theoretical AI*, 3.4:319–349, 1991.
- [Bri94] S. Bringsjord. Computation, among other things, is beneath us. *Minds and Machines*, 4.4:469–488, 1994.
- [BZ03] S. Bringsjord and M. Zenzen. *Superminds: People Harness Hypercomputation, and More*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.
- [Cop98] B. J. Copeland. Even Turing machines can compute uncomputable functions. In J. Casti, editor, *Unconventional Models of Computation*, pages 150–164. Springer-Verlag, London, UK, 1998.
- [Des11] R. Descartes. *The Philosophical Works of Descartes, Volume 1. Translated by Elizabeth S. Haldane and G.R.T. Ross*. Cambridge University Press, Cambridge, UK, 1911.
- [Deu85] D. Deutsch. Quantum theory, the Church-Turing principle, and the universal quantum computer. *Proceedings of the Royal Society of London, Series A*, 400:87–117, 1985.
- [EN02] G. Etesi and I. Nemeti. Non-turing computability via malament-hogarth space-times. *International Journal of Theoretical Physics*, 41(2):341–370, 2002.
- [Gol65] M. Gold. Limiting recursion. *Journal of Symbolic Logic*, 30(1):28–47, 1965.
- [Hau85] J. Haugeland. *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA, 1985.
- [HL00] J. D. Hamkins and A. Lewis. Infinite time Turing machines. *Journal of Symbolic Logic*, 65(2):567–604, 2000.
- [LEK⁺06] A. Litt, C. Eliasmith, F. Kroon, S. Weinstein, and P. Thagard. Is the brain a quantum computer? *Cognitive Science*, 30:593–603, 2006.
- [Put65] H. Putnam. Trial and error predicates and a solution to a problem of mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965.
- [Sie99] H. T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, Boston, MA, 1999.
- [SS94] H. Siegelmann and E.D. Sontag. Analog computation via neural nets. *Theoretical Computer Science*, 131:331–360, 1994.
- [vE95] Barbara von Eckardt. *What is Cognitive Science?* MIT Press, Cambridge, MA, 1995.

Stimulus processing in autonomously active cognitive systems

Claudius Gros

Institute of Theoretical Physics, J.W. Goethe University
60054 Frankfurt/Main, Germany

Abstract

The brain is autonomously active and possesses an ongoing internal dynamics which continues even in the temporary absence of external sensory stimuli. New experimental evidences, and theoretical considerations, indicate that this eigendynamics plays a central role in regulating the overall cognitive processing, a key property which is expected to be true also for prospective artificial cognitive systems, capable of synthetic generalized intelligence.

It is therefore of paramount importance to study possible frameworks for autonomously active cognitive systems. We report here the status of research for an approaches based on transient state dynamics and give an overall view of the embedding of this new paradigm within mainstream approaches in artificial intelligence and systems neuroscience.

What is an autonomously active cognitive systems?

Mainstream research both in artificial intelligence as well as in robotics involves task solving (Konar 2005), like hand-writing recognition, face recognition in pictures or autonomous navigation by robotic cars a la DARPA Grand Challenge. Tremendous progress has been reported for many computational tasks and its a long time that top human chess players had been able to beat the best chess programs.

In this context the notion of ‘autonomy’ is usually used in two different connotations. Autonomous robots tackle the tasks given without external help, making use of their own sensors and their previous programming. It is often assumed implicitly, that a steady increase in the complexity and in the versatility of autonomous robots could, in the long-time perspective, eventually lead to human-level universal artificial intelligence (AI).

There is an alternative view to this route to a general artificial intelligence which is motivated both by theory considerations and by recent advances in neurobiology (Gros 2009a; 2009b). In this view the term ‘autonomy’ has a different denotation. It is well known

that the brain is autonomously active in the sense that non-trivial cognitive neural activity persists even in the prolonged absence of sensory input. The brain has the task to process the sensory data input stream it receives in order to keep the body alive and functional, in a dynamic environment. The question is then, whether the autonomous eigendynamics is just a side effect, an epiphenomena, of the interwoven activity of the constituting cognitive modules. Alternatively it is possible that the eigendynamics has a central regulating role.

In this second view cognitive capabilities evolve through the mutual interaction between the autonomous neural eigendynamics and the input the brain receives from the sensory organs. Higher algorithmic task solving capabilities are not genetically prewired but evolve throughout the development of the growing animal or the synthetic intelligence.

The self-sustained neural activity of the brain is in this view a *conditio sine qua non*. Higher cognitive capabilities would neither evolve in the absence of this autonomous dynamics nor would they be present at birth. They are not encoded explicitly in the genes. The internal autonomous neural dynamics has, as a corollary of this view, at birth no semantic content, since semantic significance of neural activity patterns can arise only in conjunction with environmental information.

Modelling principles

At the basis of this concept of autonomously active cognitive systems lies the formulation of the internal self-sustained dynamical activity, for which various governing principles have been proposed (Gros 2009a). Our own approach is based on the notion of transient neural states (Gros 2007), illustrated in Fig. 1, in part motivated by experimental observations (Abeles *et al.* 1995; Kenet *et al.* 2003). The time scale of the transiently stable neural activity relates to the typical cognitive time scale of about 80-100ms. Transient state dynamics can be cast, within dynamical system theory (Gros 2008), into the framework of an attractor relic network (Gros 2007; 2009a).

The internal dynamics of an autonomously active cognitive system is not driven by the sensory input. Identical stimuli will generally lead to different re-

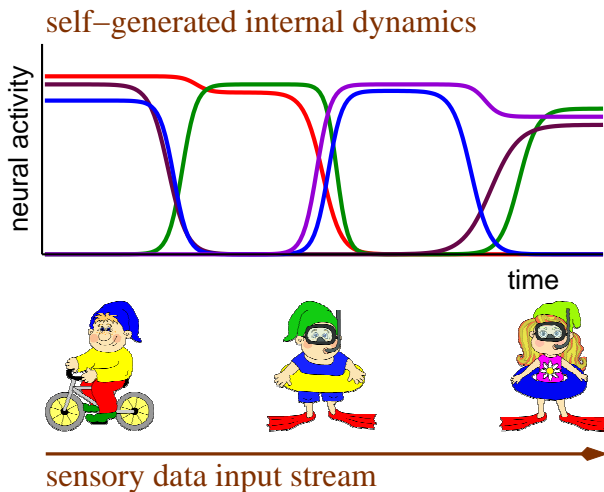


Figure 1: Cartoon of an autonomously active cognitive system. The self-generated eigendynamics in terms of a time series of transient states (top). This dynamical activity is a priori not related to the sensory data input stream (bottom), carrying environmental information. Emergent cognitive capabilities need to result from the emergence of correlation between the internal activity and the sensory data stream through unsupervised on-line learning.

sponses, or to no response at all, depending on the currently ongoing internal activity (Arieli *et al.* 1996). This state of affairs may be interpreted from two different perspectives. On one side one may view the brain as a finite-state machine for which the next state depends both on the current state and on the input. From a neurobiological perspective one may regard on the other side the brain as an autonomously active system for which the internal dynamics is modulated, but not forcefully driven, by external stimuli.

The second viewpoint allows to formulate a simple and yet very powerful principle governing the influence of the sensory data stream onto the internal activity. The internal transient-state dynamics is realized by competitive neural processes. This competition can be influenced and modulated by the sensory input when we assume that the sensory signals contribute to the internal neural competition on an equal basis. This principle has been implemented successfully (Gros & Kaczor 2008) and tested for the bars problem. The bars problem is a standardized setup for a non-linear independent component task.

The exciting result is now the following (Gros & Kaczor 2008; Gros 2009a): We have a generalized neural network which is continuously and autonomously active on its own. There is no external teacher and no explicit coding of any algorithm, all learning rules are online and Hebbian-style. In this setup the competition of the internal, autonomously generated transient state dynamics, with the data input stream leads to an

emergent cognitive capability, an independent component analysis. As a result of this process the internal transient states, the attractor relics, are mapped via their respective receptive fields to objects present in the environment, the independent components. The internal transient state dynamics acquires such a semantic content and turns into an associative thought process.

Conclusions

This is a non-technical position paper and we refer to the literature both for details on our own work (Gros 2007; 2009b), as well as for a review on the experimental situation and on alternative approaches (Gros 2009a). These results demonstrate the feasibility of the basic concept, the emergence of cognitive capabilities through the interaction of the internal eigendynamics and the external influences arriving via the sensory organs. It is our believe that the field of autonomously active cognitive system constitutes a field of emergent importance, both for systems neuroscience as well as for the eventual development of general artificial intelligences.

References

- Abeles, M.; Bergman, H.; Gat, I.; Meilijson, I.; Seidemann, E.; Tishby, N.; and Vaadia, E. 1995. Cortical Activity Flips Among Quasi-Stationary States. *Proceedings of the National Academy of Sciences* 92:8616–8620.
- Arieli, A.; Sterkin, A.; Grinvald, A.; and Aertsen, A. 1996. Dynamics of Ongoing Activity: Explanation of the Large Variability in Evoked Cortical Responses. *Science* 273:1868.
- Gros, C., and Kaczor, G. 2008. Learning in cognitive systems with autonomous dynamics. In *Proceedings of the International Conference on Cognitive Systems, Karlsruhe*. Springer.
- Gros, C. 2007. Neural networks with transient state dynamics. *New Journal of Physics* 9:109.
- Gros, C. 2008. *Complex and Adaptive Dynamical Systems: A Primer*. Springer.
- Gros, C. 2009a. Cognitive computation with autonomously active neural networks: an emerging field. *Cognitive Computation*. (in press).
- Gros, C. 2009b. Emotions, diffusive emotional control and the motivational problem for autonomous cognitive systems. In *Handbook of Research on Synthetic Emotions and Sociable Robotics: New Applications in Affective Computing and Artificial Intelligence*. J. Valverde, D. Casacuberta (Eds.), IGI-Global. (in press).
- Kenet, T.; Bibitchkov, D.; Tsodyks, M.; Grinvald, A.; and Arieli, A. 2003. Spontaneously emerging cortical representations of visual attributes. *Nature* 425:954–956.
- Konar, A. 2005. *Computational Intelligence: Principles, Techniques And Applications*. Springer.

Distribution of Environments in Formal Measures of Intelligence

Bill Hibbard

University of Wisconsin – Madison
SSEC, 1225 W. Dayton St., Madison, WI 53706

Abstract

This paper shows that a constraint on universal Turing machines is necessary for Legg's and Hutter's formal measure of intelligence to be unbiased. It also explores the relation of the No Free Lunch Theorem to formal measures of intelligence.

Introduction

A formal definition of intelligence can provide a well-defined goal to those developing artificial intelligence. Legg's and Hutter's formal measure of the intelligence of agents interacting with environments provides such a definition (Legg and Hutter 2006). Their model includes weighting distributions over time and environments. The point of this paper is to argue that a constraint on the weighting over environments is required for the utility of the intelligence measure.

A Formal Measure of Intelligence

In Legg's and Hutter's measure, based on reinforcement learning, an agent interacts with its environment at a sequence of discrete times, sending action a_i to the environment and receiving observation o_i and reward r_i from the environment at time i . These are members of finite sets A , O and R respectively, where R is a set of rational numbers between 0 and 1. The environment is defined by a probability measure $\mu(o_k r_k | o_1 r_1 a_1 \dots o_{k-1} r_{k-1} a_{k-1})$ and the agent is defined by a probability measure $\pi(a_k | o_1 r_1 a_1 \dots o_{k-1} r_{k-1} a_{k-1})$.

The value of agent π in environment μ is defined by the expected value of rewards:

$$V_\mu^\pi = \mathbf{E}(\sum_{i=1}^{\infty} w_i r_i)$$

where the $w_i \geq 0$ are a sequence of weights for future rewards subject to $\sum_{i=1}^{\infty} w_i = 1$ (Legg and Hutter combined the w_i into the r_i). In reinforcement learning the w_i are often taken to be $(1-\gamma)\gamma^{i-1}$ for some $0 < \gamma < 1$. Note $0 \leq V_\mu^\pi \leq 1$.

The intelligence of agent π is defined by a weighted sum of its values over a set E of computable environments. Environments are computed by programs, finite binary strings, on some prefix universal Turing machine (PUTM) U . The weight for $\mu \in E$ is defined in terms of its Kolmogorov complexity:

$$K(\mu) = \min \{ |p| : U(p) \text{ computes } \mu \}$$

where $|p|$ denotes the length of program p . The intelligence of agent π is:

$$V^\pi = \sum_{\mu \in E} 2^{-K(\mu)} V_\mu^\pi.$$

The value of this expression for V^π is between 0 and 1 because of Kraft's Inequality for PUTMs (Li and Vitányi 1997): $\sum_{\mu \in E} 2^{-K(\mu)} \leq 1$.

Legg and Hutter state that because $K(\mu)$ is independent of the choice of PUTM up to an additive constant that is independent of μ , we can simply pick a PUTM. They do caution that the choice of PUTM can affect the relative intelligence of agents and discuss the possibility of limiting PUTM complexity. But in fact a constraint on PUTMs is necessary to avoid intelligence measures biased toward specific environments:

Proposition 1. Given $\mu \in E$ and $\varepsilon > 0$ there exists a PUTM U_μ such that for all agents π :

$$V_\mu^\pi / 2 \leq V^\pi < V_\mu^\pi / 2 + \varepsilon$$

where V^π is computed using U_μ .

Proof. Fix a PUTM U_0 that computes environments. Given $\mu \in E$ and $\varepsilon > 0$, fix an integer n such that $2^{-n} < \varepsilon$. Then construct a PUTM U_μ that computes μ given the program "1", fails to halt (alternatively, computes μ given a program starting with between 1 and n 0's followed by a 1, and computes $U_0(p)$ given a program of $n+1$ 0's followed by p . Now define K using U_μ . Clearly:

$$2^{-K(\mu)} = 1/2$$

And, applying Kraft's Inequality to U_0 :

$$\sum_{\mu' \neq \mu} 2^{-K(\mu')} \leq 2^{-n} < \varepsilon.$$

So $V^\pi = V_\mu^\pi / 2 + X$ where $X = \sum_{\mu' \neq \mu} 2^{-K(\mu')} V_{\mu'}^\pi$ and $0 \leq X < \varepsilon$. \square

Whatever PUTM is used to compute environments, all but an arbitrarily small ε of an agent's intelligence is determined by its value in a finite number of environments. Lucky choices of actions at early, heavily weighted time steps in simple, heavily weighted environments, may give a less intelligent agent an advantage greater than ε , that a more intelligent agent cannot make up by good choices of actions in very difficult, but lightly weighted environments.

Note that as environment complexity increases, agents will require longer times to learn good actions. Thus, given a distribution of time weights that is constant over all environments, even the best agents will be unable to get any value as environment complexity

increases to infinity. It would make sense for different environments to have different time weight distributions.

Two points for consideration despite their being discouraged by reviewers of this paper:

1. If PUTM programs were answers (as in Solomonoff Induction, where an agent seeks programs that match observed environment behavior) then weighting short programs more heavily would make sense, since shorter answers are better (according to Occam's razor). But here they are being used as questions and longer programs pose more difficult questions so arguably should be weighted more heavily.

2. The physical universe contains no more than 10^{90} bits (Lloyd 2002) so is a finite state machine (FSM). Hence an intelligence measure based on FSMs is more realistic than one based on Turing machines.

No Free Lunch and a Finite Model

The No-Free-Lunch Theorem (NFLT) tells us that all optimization algorithms have equal performance when averaged over all finite environments (Wolpert and Macready 1997). It is interesting to investigate what relation this result has to intelligence measures that average agent performance over environments.

To define an intelligence measure based on finite environments take the sets A , O and R of actions, observations and rewards as finite and fixed. An environment is defined by a FSM:

$$f: S \times A \rightarrow S \times O \times R$$

where S is a finite set of states. The value of an agent in this environment is the expected value of a weighted sum over a finite sequence of future rewards, with weights summing to 1. The measured intelligence of an agent is a weighted sum of its values in environments whose state set sizes fall in a finite range, weights summing to 1.

This finite model lacks an important hypothesis of the NFLT: that the optimization algorithm never makes the same action more than once. The same result can be achieved by a no repeating state condition (NRSC) on environment FSMs: that they can never repeat the same state. Although this may seem artificial, it applies in the physical universe because of the second law of thermodynamics.

Assuming the NRSC and that all FSMs with the same number of states share the same environment weight and the same sequence of time weights, then all agents have the same measured intelligence, the average reward $(\sum_{r \in R} r) / |R|$ (Hibbard 2008).

Conclusion

According to current physics the universe is a FSM satisfying the NRSC. If we measure agent intelligence using a distribution of FSMs satisfying the NRSC in which all FSMs with the same number of states have the same weight, then all agents have the same measured intelligence. In this environment distribution past behavior of environments provides no information about their future behavior. For a useful measure of

intelligence, environments must be weighted to enable agents to predict the future from the past. This is the idea behind Kolmogorov complexity: to more heavily weight environments that can be generated by short programs since agents can more easily learn their behaviors.

However, Proposition 1 shows that a PUTM must be chosen carefully in an intelligence measure based on Kolmogorov complexity. This suggests a distribution of environments based on program length but less abstract than Kolmogorov complexity. So define a PUTM based on an ordinary programming language.

States and behaviors never repeat in the physical world, but human agents learn to predict future behavior in the world by recognizing current behavior as similar to previously observed behaviors and making predictions based on those previous behaviors. Similarity can be recognized in sequences of values from unstructured sets such as $\{0, 1\}$, but there are more ways to recognize similarity in sequences of values from sets with metric and algebraic structures such as numerical sets. Our physical world is described largely by numerical variables, and the best human efforts to predict behaviors in the physical world use numerical programming languages. So the sets A and O of actions and observations should be defined using numerical values, just as rewards are taken from a numerical set R . Including primitives for numerical operations in environment programs has the effect of skewing the distribution of environments toward similarity with the physical world.

An ordinary numerical programming language is a good candidate basis for a formal measure of intelligence. But the real point of this paper is that distributions over environments pose complex issues for formal intelligence measures. Ultimately our definition of intelligence depends on the intuition we develop from using our minds in the physical world, and the key to a useful formal measure is the way its weighting distribution over environments abstracts from our world.

References

- Hibbard, 2008. <http://www.ssec.wisc.edu/~billh/g/de.pdf>
- Legg, S. and M. Hutter. Proc. A Formal Measure of Machine Intelligence. *15th Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn 2006)*, pages 73-80. <http://www.idsia.ch/idsiareport/IDSIA-10-06.pdf>
- Li, M. and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed.*. Springer, New York, 1997. 637 pages.
- Lloyd, S. Computational Capacity of the Universe. *Phys.Rev.Lett.* 88 (2002) 237901. <http://arxiv.org/abs/quant-ph/0110141>
- Wolpert, D. and W. Macready, No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* 1, 67. 1997. <http://ic.arc.nasa.gov/people/dhw/papers/78.pdf>

The Importance of Being Neural-Symbolic – A Wilde Position

Pascal Hitzler

AIFB, Karlsruhe Institute of Technology, Germany

Kai-Uwe Kühnberger

IKW, University of Osnabrück, Germany

Abstract

We argue that Neural-Symbolic Integration is a topic of central importance for the advancement of Artificial General Intelligence.

What We Want

Artificial General Intelligence – the quest for artificially created entities with human-like abilities – has been pursued by humanity since the invention of machines. It has also been a driving force in establishing artificial intelligence (AI) as a discipline. 20th century AI, however, has developed into a much narrower direction, focussing more and more on special-purpose and single-method driven solutions for problems which were once (or still are) considered to be challenging, like game playing, speech recognition, natural language understanding, computer vision, cognitive robotics, and many others. 20th century AI can, in our opinion, be perceived as expert system AI, producing and pursuing solutions for specific tasks. We don't say that this is a bad development – quite in contrast, we think that this was (and still is) a very worthwhile adventure with ample (and in some cases well-proven) scope for considerable impact on our society.

The pursuit of Artificial General Intelligence (AGI), however, has been declining in the 20th century, presumably because the original vision of establishing systems with the envisioned capabilities turned out to be much harder to realise than it had seemed in the beginning. But in recent years a rejuvenation of the original ideas has become apparent, driven on the one hand by the insight that certain complex tasks are outside the scope of specialised systems, and on the other hand by rapid developments in the neurosciences based on the invention of substantially refined means of recording and analysing neural activation patterns in the brain. These are accompanied by interdisciplinary efforts within the cognitive science community, including psychologists and linguists with similar visions.

It is apparent that the realisation of AGI requires the cross-disciplinary integration of ideas, methods, and theories. Indeed we believe that disciplines such as (narrow) artificial intelligence, neuroscience, psychol-

ogy, and computational linguistics will have to converge substantially before we can hope to realise human-like artificially intelligent systems. One of the central questions in this pursuit is thus a meta-question: What are concrete lines of research which can be pursued in the immediate future in order to advance in the right direction? The general vision does not give any answers to this, and while it is obvious that we require some grand all-encompassing interdisciplinary theories for AGI, we cannot hope to achieve this in one giant leap. For practical purposes – out of pure necessity since we cannot shred our scientific inheritance – we require the identification of next steps, of particular topics which are narrow enough so that they can be pursued, but general enough so that they can advance us into the right direction.

What We Propose

Our proposal for such a research direction starts from two obvious observations.

- The physical implementation of our mind is based on the neural system, i.e. on a network of neurons as identified and investigated in the neurosciences. If we hope to achieve Artificial General Intelligence, we cannot expect to ignore this neural or subsymbolic aspect of biological intelligent systems.
- Formal modelling of complex tasks and human thinking is based on symbol manipulation, complex symbolic structures (like graphs, trees, shapes, and grammars) and mathematical logic. At present, there exists no viable alternative to symbolic modelling in order to encode complex tasks.

These two perspectives however – the neural and the symbolic – are substantially orthogonal to each other in terms of the state of the art in the corresponding disciplines. Neural systems are hard if not impossible to understand symbolically. It is quite unclear at present how symbolic processing at large emerges from neural systems. Symbolic knowledge representation and manipulation at the level required for AGI is way outside the scope of current artificial neural approaches.

At the same time humans – using their neural-based brains – are able to deal successfully with symbolic

tasks, to manipulate symbolic formalisms, to represent knowledge using them, and to solve complex problems based on them. So apparently there is a considerable mismatch between human neurophysiology and cognitive capabilities as role models for AGI on the one hand, and theories and computational models for neural systems and symbolic processing on the other hand.

It is our believe that significant progress in AGI requires the unification of neural and symbolic approaches in terms of theories and computational models. We believe that this unification is central for the advancement of AGI. We also believe that the pursuit of this unification is timely and feasible based on the current state of the art, which is what we discuss next.

Where We Are

We briefly mention some recent developments in neural-symbolic integration which we consider to be of particular importance. For further information on related topics and the state of the art, we recommend to consult (Bader and Hitzler, 2005; Hammer and Hitzler, 2007).

The line of investigation we want to mention takes its starting point from computational models in (narrow) AI and machine learning. It sets out to realise systems based on artificial neural networks which are capable of learning and dealing with symbolic logic. While this can be traced back to the landmark paper (McCulloch and Pitts, 1943) on the relation between propositional logic and binary threshold neural networks, it has been largely dormant until the 1990s, where first neural-symbolic learning systems based on these ideas were realised – see e.g. (Towell and Shavlik, 1994; d’Avila Garcez and Zaverucha, 1999; d’Avila Garcez et al., 2002). While these initial systems were still confined to propositional logics, in recent years systems with similar capabilities based on first-order logic have been realised – see e.g. (Gust et al., 2007; Bader et al., 2008). It is to be noted, however, that these systems – despite the fact that they provide a conceptual breakthrough in symbol processing by artificial neural networks – are still severely limited in their scope and applicability, and improvements in these directions do not appear to be straightforward at all.

Our selection is obviously purely subjective, and there are plenty of other related efforts which could be mentioned. The line of investigation which we presented, however, appears to be typical and representative in that it is driven by computer science, machine learning, or AI perspectives. *We know of no work in the area which is mainly driven by the AGI perspective, and this includes our own achievements on the topic.*¹

¹There are some investigations which are driven from a neuroscience perspective, see e.g. (Yang and Shadlen, 2007), but they do not yet cover higher-level cognitive modelling in any reasonable sense.

Where To Go

We need to advance the state of the art in neural-symbolic integration in order to get closer to the AGI vision. For this, we need to improve on the established approaches in order to find out to what limits they can be pushed. In particular, this requires us to adapt and improve them in order to become functional in cognitive systems application scenarios.

At the same time, however, we also require new ideas borrowed from other disciplines, in order to establish neural-symbolic systems which are driven by the AGI vision. Results from cognitive psychology on particularities of human thinking which are not usually covered by standard logical methods need to be included. Recent paradigms for artificial neural networks which are more strongly inspired from neuroscience – see e.g. (Maass, 2002) – need to be investigated for neural-symbolic integration. On top of this, we require creative new ideas borrowed e.g. from dynamical systems theory or organic computing to further the topic.

The challenges are ahead, and we hope to have conveyed the vital Importance of Being Neural-Symbolic.

References

- Bader, S. and Hitzler, P. (2005). Dimensions of neural-symbolic integration – a structured survey. In Artemov, S. et al., editors, *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 167–194. King’s College Publications, London.
- Bader, S., Hitzler, P., and Hölldobler, S. (2008). Connectionist model generation: A first-order approach. *Neurocomputing*, 71:2420–2432.
- d’Avila Garcez, A., Broda, K., and Gabbay, D. (2002). *Neural-Symbolic Learning Systems — Foundations and Applications*. Springer, Berlin.
- d’Avila Garcez, A. and Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77.
- Gust, H., Kühnberger, K.-U., and Geibel, P. (2007). Learning models of predicate logical theories with neural networks based on topos theory. In (Hammer and Hitzler, 2007), pages 233–264.
- Hammer, B. and Hitzler, P., editors (2007). *Perspectives of Neural-Symbolic Integration*. Springer, Berlin.
- Maass, W. (2002). Paradigms for computing with spiking neurons. In van Hemmen, J. et al., editors, *Models of Neural Networks*, pages 373–402. Springer.
- McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.
- Towell, G. and Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165.
- Yang, T. and Shadlen, M. (2007). Probabilistic reasoning by neurons. *nature*, 447:1075–1080.

Improving the Believability of Non-Player Characters in Simulations

¹Jere D. Miles and ²Rahman Tashakkori

¹Wilkes Community College

Division of Business and Public Service Technologies, Wilkes Community College, Wilkesboro, NC 28697, USA

²Appalachian State University

Department of Computer Science, Appalachian State University, Boone, NC 28608, USA

¹jere.miles@wilkescc.edu and ²rt@cs.appstate.edu

Abstract

In recent years the video game industry has experienced rapid expansion developing virtual environments that accurately mimic a real-world setting. However, the industry almost entirely relies on finite state machines for deploying computer-controlled characters within these environments. This has resulted in simulated inhabitants that lack the same degree of believability as their surroundings. As part of this research a simulation was developed using Java in which an agent was placed. In a survey students were asked to rank the believability of different artificial intelligence techniques employed by the simulation. The genetic algorithm developed for this simulation provided for an agent whose decisions were more believable than the decisions generated by a finite state machine or random selection process.

Introduction

The notion of believability is explored within this paper through the implementation of a virtual environment that simulates a popular consumer video game inhabited by a computer-controlled character. The research is particularly interested in the role that different decision-making mechanisms may play in the observed believability of the character.

A hypothesis that incorporating evolutionary computing techniques, such as a genetic algorithm, within the agent's decision-making process may provide an increased believability of the agent over those utilizing more traditional video game artificial intelligence approaches was established. In order to test this hypothesis, a simulator was developed with three separate decision-making mechanisms: a random selector, a finite state machine, and a genetic algorithm. These mechanisms were observed in operation by human viewers who rated each on their believability as a human-like character within the virtual environment.

This research is significant as there is little or no publication exploring the potential differences between these artificial intelligence techniques in terms of perceived believability. Many researchers have documented the improvement that evolutionary techniques

provide in a competitive or problem solving environment. Also, There have been documented efforts that have discovered some of the required characteristics to improve the believability of a computer-controlled character [3][5]. However, these efforts have not focused on the decision-making mechanism employed by the agent. This paper will address the potential usability of evolutionary artificial intelligence techniques within a video game environment. While exploring the usefulness of these methods, it was expected to demonstrate that these approaches were not only achievable, but that they would provide an increase in believability of the computer-controlled character to a human observer. This increased believability may provide for a more engaging interactive product that can be delivered by the video game industry

Background

The artificial intelligence systems of video games have historically relied on a finite state machine to provide a decision-making mechanism for the computer-controlled characters. These finite state machines are implemented in virtually all computer games that have been developed and released [1]. Recent additions have allowed scripting techniques to be used as a method to fine-tune the state machines that actually generate the logic for the agents [7]. It has been considered necessary for the video game to employ these techniques in order to ensure a stable and testable product.

However, in recent years, human players have begun a migration towards games that involve playing with other people rather than playing with a computer. This trend seems to be a result of human players finding the computer-controlled characters to be too predictable in their behaviors to offer a reasonable level of challenge. Also, other human players were preferred for their ability to be flexible and adapt their responses to dynamic situations [6]. There has been a renewed interest in the development of believable video game agents with the success of Electronic Arts' *The Sims* [2]. This series of games is considered to have raised the expectation of artificial intelligence within a video game and many developers are attempting to replicate a similar degree of believability [7]. The premise of *The Sims* is that the

player plays the role of a virtual human making the decisions of this character's life [4]. A character that is not directly under the control of a human player generates its actions based upon the behaviors that are present in all of the objects [8].

Results

In order to test the impact that genetic algorithms may have on the believability of a computer-controlled character, a virtual environment was created to mimic many of the capabilities and displays of Electronic Arts' *The Sims* [2]. The simulator was named *StudentLife* and was populated by a computer-controlled student character, which was then observed in operation by human volunteers. The virtual environment did not allow these observers to interact with the system in any way other than through observance. This insured that the observers would focus on the behaviors of the agent being surveyed. The system allowed the human observers to have a full view of the virtual environment and all of the internal conditions of the agent.

Ninety-seven surveys were administered to the observers. In Figure 1, a comparison is provided of the highest-ranking values from each of the observers. Of the human observers, 43% found that the genetic algorithm behaved the most believably of the options. The remainder was split with the finite state machine preferred by 27% and the random selector rated the highest by 12%. The remaining 18% of the observers gave a similarly high value to more than one of the logic engines on their survey.

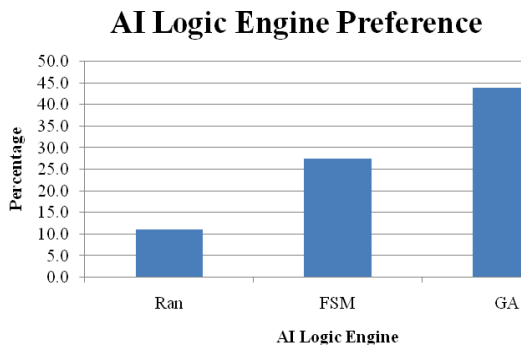


Figure 1: Logic engine preferences overall

It is worth noting that among the respondents, those who gave the random engine a higher score did so due to the variations in the activities that were performed. The observers expressed an interest in the believability of the random student because of the different actions that were performed, not because of the condition in which the agent was. This implied that for 11% of the respondents, the state of the agent's needs and happiness had no bearing on

the believability of the behavior of the computer-controlled character.

Conclusions

From the results that were collected, it has been concluded that a genetic algorithm could be developed with the ability to govern the decision-making process for a computer-controlled character within a simulated environment. The concerns of a dumb intelligence evolving [7] were adequately addressed through the strength of the fitness function in combination with the mutation capability to introduce new genes to an evolving set of solutions.

Based upon the results of the surveys, it was found that a genetic algorithm provided a more believable virtual inhabitant. These results could be applicable to all video games that utilize agents for the purposes of story advancement or cooperative game play. Games that have an inherent need for a competition from the computer, may not gain an improvement in believability through these techniques.

References

- [1] Buckland, Matt, 2005, *Programming Game AI by Example*, Woodware Game Developer's Library.
- [2] Electronic Arts, 2008, <http://thesims.ea.com>.
- [3] Laird, John, E., Duchi, John, 2000, "Creating Human-Like Synthetic Characters with Multiple Skill Levels," *AAAI 2000 Fall Symposium Series: Simulating Human Agents*.
- [4] Macedonia, Michael, 2000, "Using Technology and Innovation to Simulate Daily Life," *Computer*, V. 33, No 4.
- [5] Paiva, Ana, Dias, Joao, Sobral, Daniel, Aylett, Ruth, Sobreperez, Polly, Woods, Sarah, Zoll, Carsten, Hall, Lynne, 2004, "Caring for Agents and Agents that Care: Building Empathic Relations with Synthetic Agents," *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*.
- [6] Swetser, Penelope, Johnson, Daniel, Swetser, Jane, Wiles, Janet, 2003, "Creating Engaging Artificial Character for Games," *Proceedings of the Second International Conference on Entertainment Computing*.
- [7] Woodcock, Steven, 2001, "AI Roundtable Moderator's Report," *2001 Game Developer's Conference*.
- [8] Wright, Will, Forbus, Kenneth, 2001, "Some Notes on Programming Objects in The Sims," *Qualitative Reasoning Group*.

Understanding the Brain's Emergent Properties

Don Miner and **Marc Pickett** and **Marie desJardins**

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, Maryland 21250

Abstract

In this paper, we discuss the possibility of applying rule abstraction, a method designed to understand emergent systems, to the physiology of the brain. Rule abstraction reduces complex systems into simpler subsystems, each of which are then understood in terms of their respective subsystems. This process aids in the understanding of complex systems and how behavior emerges from the low-level interactions. We believe that this technique can be applied to the brain in order to understand the mind and its essential cognitive phenomena. Once a sufficient model of the brain and mind is created, our framework could then be used to build artificial general intelligence that is based on human intelligence.

Introduction

In this paper, we propose a method of understanding human intelligence by understanding how the mind emerges from the physiology of the brain. The brain may be viewed as a complex system that produces features of human-level intelligence from the low-level physical mechanisms in the neural system. We hypothesize that we can improve our understanding of how the brain works by reducing its emergent behavior into layers of successively more complex behaviors on top of the neurological subsystem. To achieve this goal, we propose the use of *rule abstraction*, our mechanism for creating hierarchies of emergent behaviors (discussed in more detail in the next section). The purpose of this paper is to stimulate discussion about the value of such an approach for understanding the human brain and, as a result, understand intelligence.

Understanding the mind by directly studying low-level structures, such as neurons and glial cells has not proven fruitful to date. For example, biologically inspired systems such as Jeff Hawkins' Memory Prediction (Hawkins & Blakeslee 2004) and Blue Brain (Markram 2006) have not led to general models of intelligence. The leap from neurons to high-level processes, such as reasoning and language, is too great for humans or machines to decipher in a single step as of 2009. However, in smaller-scale complex systems, such as boid flocking (Reynolds 1987), we can mathematically model how simple agent-level rules produce the flocking emergent behavior (Miner, Hamilton, & desJardins 2008).

We propose trying to understand the brain by first partitioning it into hierarchical sub-processes. Each sub-process has emergent behavior that results from the emergent behavior of its lesser sub-processes. Then, we find mathematical correlations between low-level behaviors and abstract-level properties using this sub-process structure. The result will be a hierarchy of successively more complex emergent systems.

In this paper, we outline how rule abstraction and hierarchies can be used to understand emergent systems. We then discuss the challenges in applying this method to the brain and intelligence.

Rule Abstraction and Hierarchies

Rule abstraction is the process of finding a mathematical correlation between low-level rules and abstract properties in an emergent system (Miner, Hamilton, & desJardins 2008). Low-level rules are the basic actions and atomic structures in the emergent system. Abstract properties are the higher-level emergent behaviors. For example, a multicellular organism exhibits emergent behaviors (the abstract properties) that result from the individual behaviors of the cells (the low-level rules).

Applying rule abstraction may be difficult in some emergent systems. The most obvious approach to rule abstraction is manually specifying the correlation. This requires a significant level of human intuition into how the abstract level properties of the complex system emerge. The lack of human understanding of complex systems makes manual specification impossible at this time. To make this problem tractable, a number of computer science and artificial intelligence techniques can be applied to learn the mathematical correlations between the low-level rules and the abstract properties. For example, we have experimented with a sampling and regression technique in which we observed several different configurations of a complex system and then used regression to create a continuous two-way mapping of the low-level rules and the abstract properties (Miner, Hamilton, & desJardins 2008). Similarly, a theory of a single cell's behavior could be developed by observing several different cells with different internal configurations.

This methodology provides the ability to use previously mapped abstract properties as low-level rules in higher-order complex systems. For example, once we have developed a

theory of a cell, we can use its mapping to develop theories of small multicellular organisms or organs in larger organisms. The intermediate step of the cell theory enabled an understanding of multicellular organisms that may not have been previously possible. These hierarchies of emergence are clear in many cases. However, in other complex systems, such as the brain, emergent sub-processes may not be easily identifiable.

Understanding the Brain

The brain is a mysterious complex system. Rule abstraction is general enough, in theory, to handle any emergent system. However, there are three key challenges. First, a certain level of human engineering is currently required to identify the appropriate level of abstraction. Second, in the brain, the correlation models that are required may be too complex and may require more sophisticated learning methods than what we have tried with rule abstraction. Finally, it could be the case that the system we are trying to understand simply has no “midpoints.” That is, emergent behavior results from the low-level rules and no meaningful reduction of abstraction layers can be found. Regardless of these hurdles, we hope that our approach will one day be able to build artificial general intelligence.

The first step to applying rule abstraction to the brain and mind, as with any complex system, is by declaring the obvious: the cognitive powers of the mind and brain result from the physiology’s emergent properties. This statement represents the initial state of the hierarchy. At this point, learning a mathematical correlation would be too difficult. To overcome this problem, we break down the complex system by adding additional mid-level abstract properties to the hierarchy and learning the correlations between these, instead. For example, the top-level emergent behavior may be a combination of lesser emergent properties such as language, memory, reasoning, etc. and could each be added as nodes in the hierarchy. A similar approach can be taken from the bottom-up: the neurons, glial cells, chemical reactions, and other physiology could be parts of emergent subsystems in the brain. These subsystems may include physical structures such as cortical columns or more abstract features such as prediction in the auditory cortex. Hopefully, at some point, no correlation in the hierarchy between higher-order emergent subsystems and lower-order emergent subsystems will be too difficult to learn.

We do not believe that achieving artificial general intelligence through this type of framework is possible at this time because neuroscience, cognitive science and other related fields not yet able to explain how the mind works as a whole. However, we do believe that our framework scales to any complex system and thus increasingly accurate rule abstraction hierarchies can be built as more scientific information on the brain and mind are gathered.

Discussion

We now ask several questions to ourselves and the research community. Answers to these questions would be useful in understanding emergence, general intelligence and specifically human intelligence.

How many midpoints or layers would be in a rule abstraction hierarchy model of a brain? If there are too many layers (greater than ten), too much error may be introduced and may yield a unfavorable results. If there are too few layers (less than three), current machine learning techniques may not be powerful enough to build correlations in these massive emergent systems. Also, how deep does the model have to be? Strong enough abstract models of cortical columns may make modeling individual neurons unnecessary. On the other hand, perhaps a neural-level base in the hierarchy is not deep enough.

Unfortunately, there is much speculation and much uncertainty in defining the mind’s subsystems, due to lack of scientific understanding in relevant fields. Concepts such as language, memory and reasoning are easily observable, but are there some phenomena that have not been discovered? Also, specifying the order of known phenomena in a hierarchy is difficult. Do some of these known phenomena, such as language, emerge from other essential subsystems and thus not a foundation of general intelligence?

We may be able to take lessons learned from using rule abstraction in simpler domains and apply them to the brain. Perhaps nature has used emergent systems similar to the ones in our brains in other complex systems such as ants, traffic jams, rat brains, and galaxies. Is there some overarching theme of emergence in our universe? Is there a general theory of emergence? This question may be harder to answer than understanding the brain and developing artificial general intelligence. However, any useful information that comes out of trying to answer this question may be helpful in understanding human-level intelligence.

Conclusions

We have given an overview of our method of rule abstraction, and explained how it can bring new understanding to emergent systems. We believe that rule abstraction could be applied to the brain in order to understand the mind. We hope that by introducing this new approach, we will stimulate discussion on our method’s usefulness for this domain and will inspire novel views of human intelligence.

References

- Hawkins, J., and Blakeslee, S. 2004. *On Intelligence*. Times Books.
- Markram, H. 2006. The Blue Brain Project. *Nature Reviews Neuroscience* 7(2):153–160.
- Miner, D.; Hamilton, P.; and desJardins, M. 2008. The Swarm Application Framework. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (Student Abstract)*.
- Reynolds, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.* 21(4):25–34.

Why BICA is Necessary for AGI

Alexei V. Samsonovich

Krasnow Institute for Advanced Study, George Mason University
4400 University Drive MS 2A1, Fairfax, VA 22030-4444, USA
asamsono@gmu.edu

Abstract

The challenge of creating AGI is better understood in the context of recent studies of biologically inspired cognitive architectures (BICA). While the solution is still far away, promising ideas can be derived from biological inspirations. The notions of a chain reaction, its critical mass and scaling laws prove to be helpful in understanding the conditions for a self-sustained bootstrapped cognitive growth of artifacts. Empirical identification of the critical mass of intelligence is possible using the BICA framework and scalability criteria.

Keywords: cognitive architectures, self-regulated learning, human-level intelligence, scaffolding, critical mass

Why BICA Challenge Is a Necessary Step

The challenge of artificial intelligence formulated more than half a century ago (McCarthy et al. 1955/2006) turned far more complicated and more vital for us than originally thought. Today we understand that a similar situation exists with the *BICA challenge*. This challenge is to create a biologically inspired cognitive architecture (BICA) capable of mimicking the human cognition and learning. It can be viewed as a more focused and constrained version of the general artificial intelligence (AGI) challenge. At the same time, the BICA challenge is primarily targeting the core of higher cognition. Could the BICA challenge be a necessary step in solving the AGI challenge? Biological solutions are not always suitable for engineering tasks: e.g., airplanes do not flap wings and would not benefit from doing this. Biological inspirations, however, appear to be necessary for reaching AGI, because in this case biology provides the only known model of a solution.

The original, informally stated goal of the DARPA IPTO BICA program (2005-2006) was to capture the ‘magic’ of human cognition, while no clear understanding of what this ‘magic’ is was available in 2005. Initially, the focus was on integration, biological fidelity, and test-driven design, with a special attention paid to human-specific higher cognitive abilities: meta-cognition, self-awareness, episodic memory, emotional intelligence, theory of mind, natural language and social capabilities. By the end of the funded period it became clear that the human-like learning is the key for solving the challenge,

and ‘*capturing the magic*’ amounts to reproducing the phenomenon of human cognitive growth from a two-year old to an adult, using various forms of scaffolding and guidance by human instructors. This scenario became a prototype for the big BICA Challenge intended for Phase II, which, unfortunately, was canceled.

Recently, BICA alumni and other interested researchers gathered in Arlington in a AAAI 2008 Fall Symposium on BICA (<http://binf.gmu.edu/~asamsono/bica/>), trying to understand what went right and what went wrong with the BICA program, and what can we learn from this enterprise. During the symposium, we learned that we are still interested in the BICA Challenge and are dedicated to solving it, because we view it as a critical stepping stone on the road to AGI (Samsonovich & Mueller, 2008).

The BICA Challenge understood as explained above has a potential impact of its solution extending far beyond immediate military or industrial goals. Its objective is to bridge the gap separating artificial and natural intelligence: the gap in autonomous cognitive growth abilities. This main dimension of the gap underlies its other dimensions, including robustness, flexibility and adaptability. In this sense, solving the BICA Challenge is a necessary, critical step in reaching AGI. It is argued below that using a BICA as a basis of a solution is also necessary.

Critical Mass Hypothesis and Biology

The *critical mass hypothesis* can be formulated as follows. By enabling the core human-like-learning mechanisms in artifacts, one can initiate a “chain reaction” of development of knowledge, skills and learning capabilities in artifacts. Agents learn how to learn and how to improve themselves. Given the right embodiment, embedding and scaffolding, this hypothetical process of bootstrapped cognitive growth will continue to a point when artifacts reach a human level of AGI. If this concept makes sense, then it is natural to accept that the cognitive chain reaction is characterized by a *critical mass*: a minimal set of architectural components and mechanisms, cognitive functions, features of the embodiment, interface, environment and scaffolding that together make the reaction possible, self-sustainable and sufficiently scalable. If this is the case, then identifying the critical mass (or, to be more parsimonious, any feasible supercritical mass) would constitute a substantial part of a solution to the BICA/AGI challenge.

Why and How Biology Gets Involved

Despite the impressive recent progress in many fields of artificial intelligence, as of now there is no clear prospect of a self-sustainable cognitive chain reaction with its critical mass, scaling laws and other parameters derived from studies of artificial intelligence. There is only one known example of a cognitive chain reaction: natural human development. As a matter of biological fact, we know that a normal human child under typical social conditions possesses a supercritical mass, while a lower animal or an out-of-shelf computer do not; therefore, there must be a critical mass somewhere in between. In order to identify it, one can start moving down from the known positive example, taking out functional components one by one. To do this, one may study human pathologies that prevent normal cognitive development. While a rigorous study of this sort would be difficult, it should be pointed here that children deprived of certain seemingly vital sensory and/or action abilities can nevertheless grow cognitively to an adult level. On the other hand, the ability to acquire some form of language interactively and the ability to exercise voluntary behavior appear to be critical for cognitive growth (e.g., Thelin & Fussner 2005).

Even if it is possible to reduce the AGI (or BICA) challenge to a language acquisition challenge for an embodied agent, this step does not make the task substantially easier. E.g., there is currently no good idea of how to approach passing the Turing test, or what tests would be useful to pass in order to ensure the language acquisition capacity. There is no clear understanding of how to build scaffolding for the growing agent, or what should be the driving force in its cognitive growth. Here biology provides another hint. The development and survivability of a biological organism depends on a number of “built-in” drives: from lower drives, such as hunger, pain and pleasure, to higher drives, such as curiosity and appreciation of beauty. It is reasonable to think that a core set of drives should be a part of the critical mass.

Defining and Detecting the Critical Mass

There are many questions related to identification of the critical mass in terms of BICA; here are some of them. What should we borrow from studies of the acquisition of language by young children? How the agent should develop an understanding of concepts associated with linguistic constructs? Is language processing an underlying capacity for other faculties like symbolic processing, or vice versa? What kinds of architectural mechanisms should be involved in symbolic and subsymbolic processing? What are the general critical mass requirements for memory systems, kinds of representations and principles of information processing? What cognitive abilities should be innate (preprogrammed), and which of them should develop through learning? What can we say regarding *the ladder*, i.e., the curriculum for artifacts understood as the sequence of tasks, paradigms and intermediate learning goals that will scaffold their rapid cognitive growth up to

the human level? An answer to these questions will be provided by the definition of a critical mass.

Assuming that there is a threshold level of intelligence that enables bootstrapping, we should expect certain scalability of the phenomenon. The notion of a chain reaction predicts an exponential scaling law. Therefore, scaling laws and metrics associated with them can be taken as a basis for the main criterion for a critical mass. It is therefore interesting and necessary to conduct a set of experiments with cognitive growth demonstrations using the BICA framework, starting from simplistic “toy” examples and gradually relaxing their limitations. Finally, the answer should be given in the form of a BICA, because the only known prototype exists in biology.

A BICA-Based Roadmap to AGI

A specific approach to solving the BICA Challenge (Samsonovich et al. 2008) was developed by our George Mason University team based on a combination of the GMU BICA cognitive architecture and the concept of self-regulated learning, or SRL (Zimmerman 2002). The key idea is to use GMU BICA as a model of SRL that will be dynamically mapped to the student mind. This approach will allow us to develop a faithful model of student SRL, to be used in education and in artificial intelligence, starting with intelligent tutoring systems that will be used as SRL assistants. While this is the current agenda, our eventual goal is to build Robby the Robot that can learn virtually everything like a human child.

References

- McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. (1955/2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. *AI Magazine*, 27 (4) 12-14.
- Samsonovich, A. V., Kitsantas, A., and Dabbag, N. (2008). Cognitive Constructor: A Biologically-Inspired Self-Regulated Learning Partner. In Samsonovich, A. V. (Ed.). *Biologically Inspired Cognitive Architectures. Papers from the AAAI Fall Symposium*. AAAI Technical Report FS-08-04, pp. 91-96. Menlo Park, CA: AAAI Press.
- Samsonovich, A. V., and Mueller, S. T. (2008). Toward a growing computational replica of the human mind. In Samsonovich, A. V. (Ed.). *Biologically Inspired Cognitive Architectures. Papers from the AAAI Fall Symposium*. AAAI Technical Report FS-08-04, pp. 1-3. Menlo Park, CA: AAAI Press.
- Thelin, J. W. and Fussner, J. C. (2005). Factors related to the development of communication in CHARGE syndrome. *American Journal of Medical Genetics Part A*, 133A (3): 282-290.
- Zimmerman, B. J. (2002). Becoming a self-regulated learner: An overview. *Theory into Practice*, 41 (2): 64-70.

Importing Space-time Concepts Into AGI

Eugene J. Surowitz

New York University (visiting)
PO Box 7639, Greenwich, CT 06836
surow@attglobal.net

Abstract

Feedback cycles are proposed as the general unit of intellect and intelligence. This enables the importation of space-time concepts from physics. The action-lens, a processing structure based on these measurable objects, is defined. Larger assemblies of this structure may support the Lowen model of information processing by the brain.

Cycles as the Fundamental State

The purpose of this paper is to provide part of the rationale for adopting feedback loops as the particular dynamical structure fundamental for the construction of intelligent systems. This question was raised with respect to a paper submitted to AGI-08. What favors the adoption of processor cycles as a primitive unit of intellect?

Predicting the structure and behavior of large assemblies of processors or neurons, without an analyzable and controllable generation mechanism appears to create a combinatoric scaling barrier to creating of an AGI from those base units. This is very analogous to the problem of discovering the existence, structure, and properties of DNA molecules by examining every possible combination of elements from the periodic table. It would seem desirable to have an engineering approach instead.

Random graph theory predicts the occurrence of cycles when the number of connections equals the number of nodes. The neurons of the brain far exceed this criteria and we may assume that cycle structures are common. Cycles can be counted, at least in principle, and the number of cycles provides a measure of the internal structure of a system; fluctuations in that count can be a measure of dynamical properties of the system.

Cycles are less primitive than neurons or processors. An appropriate analogy would be the difference between atomic physics and chemistry. They are also recursively constructive as they can be inter-connected and will then inter-communicate to form larger cycle based structures. They then take on the aspect of primitive sub-assemblies, or molecules in the chemistry analogy. Adopting cycles of processors as a fundamental structure may be an effective way of addressing the

scaling problem for AGI by reducing branching ratios as a prospective AGI system is enlarged.

Cycles create and define time-scales: the propagation time it takes for a signal's effects to completely travel the cycle's circumference and arrive at the originating node creates a clock based on this propagation time and defines a time-scale associated with the cycle. The cycle based time-scales enable the importation of space-time concepts from physics. Those concepts, in turn, bootstrap the construction of processor structures to be used as engineering elements to generate still larger structures.

A by-product of that construction is the ability to support multiple short and long time-scales with a single primitive structure. This addresses the time-scale dichotomy of the need of organisms and systems to act in the short-term on plans with long-term objectives. In the context of AGI, the cycle time-scales define persistent objects competing for resources leading to action toward short and long term objectives. Groups of cycles of similar circumference will define a range of times for such actions. This suggests a conceptual structure, borrowed from physics, that is useful to draw and visualize cycle structures and bootstrap our ability to examine the space-time behavior of large numbers of co-located cycles.

Cycle Space-Time and the Action-Lens

Let's proceed with constructing a diagram showing the relationship of cycles and time: The set of cycles with all possible circumferences forms an infinite disk when laid out as circles on a plane with a common center. (Here, we are assuming large numbers of nodes in each cycle which creates an image of continuity.) This is the "space" portion of the space-time we are constructing. Plot the respective cycle-times on a vertical axis thru the common center. This is the "time" portion of the space-time.

This visual aid suggests a structure combining a large base cycle, which processes over extended physical times, with a much smaller focus structure, even a single node, which works on much shorter time-scales, into a single object to be used as a building block to

construct systems. This structure will be called the “action-lens” in the sequel.

The nodes of a cycle of processors, that operates over multiples of its local cycle-time, are a persistent object, the base-cycle. Again, visualize a cone of connectivity with its base coincident with a cycle and its apex on the cycle-time axis at the point corresponding to the cycle-time of our particular cycle. The visualized cone itself would be the surface created by considering the base-cycle so dense in processor nodes as to be almost continuous. In general the connectivity of the base-cycle need not be so great. The cycle-times of cycles with similar circumference land in a small but not necessarily contiguous neighborhood on the cycle-time axis. The cycle-time axis can now be thought of as a linear array of nodes with each successive node situated at a slightly greater time along the axis.

The fraction of the nodes of a base-cycle directly linked to a single node, the focus-node, looks like a tepee’s poles without the shell. More generally, the focus-node can be replaced by a set of nodes that are almost completely connected, to some approximation of completeness in the graph theoretical sense. This also allows similar circumference cycles to feed a set of focus-nodes, with very rapid propagation of signals among the focus-nodes, relative to the base cycle-times. What does the focus-node/graph do? The focus-node or graph provides a second time-scale for the base-cycle. It creates short term effects from a persistent object whose existence is measured only on larger cycle-time scales. One might consider the activity of assembling a model from a kit of parts as quick part-steps carried out to yield the final model, the long-term objective.

The combination of base-cycles and a focus-graph constitute the “action-lens”. The term “lens” is borrowed from optics in order to cover more elaborate versions of this object suggested by the space-time construction. The action-lens is intended to replace the halt-free Turing machines used at the lowest level of cycle constructs. Rather than use arbitrary or random inter-connections of nodes, we can use the action-lens as an engineering object to construct larger structures.

Mesoscale Structure

The mesoscale, or middle scale of structure, between primitive processors and the macroscopic scale of general intelligence functioning may be approachable if we use recursive definition on the action-lens structure. As an example, we could replace the elementary processors of the basic action-lens with the action-lens structure itself thus generating an action-lens of action-lenses.

Under the control of executive functions, we could generate other action-lens based complexes. As examples, these could connect a lens’s base-cycle to other lens’s base-cycles, connect the focus-graph to other foci, and connect foci to base-cycle nodes. Respectively, these yield a stack structure, an hour-glass, and a multi-layer. Gluing action-lenses at nodes along their base-

cycles gives the stack; gluing focus to focus creates the hour-glass; gluing an inverted layer of action-lenses to an upright layer of action-lenses, creates a bilayer.

Unconstrained generation at the mesoscale could, and probably would, lead to a space of generated functions so large that determining those most useful for AGI applications would be difficult at best. A macroscopic model that provides a candidate set of target functions to be generated by mesoscale generation operators would be one possible way of imposing constraints somewhat in the manner of a variational problem. There is such a candidate model, the Lowen model, which serves to supply the target high-level functions.

The Lowen Model of Consciousness

On the macroscopic scale, Walter Lowen, late of the State University of New York at Binghamton, constructed a model of the conscious processes of the mind (Lowen 1982), inspired by Carl Jung’s theory of psychological types. Lowen founded his concepts on information processing and system theory.

The Lowen information processing model conceives of consciousness as constructed from specialized processes, variously termed capacities or poles in the model. Sensory data is taken and handed from capacity to capacity while eventually producing behavioral outputs. Some of these capacities such as sorting, path finding, and decision tree processing are identifiable as well studied topics of computer science. The list of capacities was limited to a set of sixteen which arose from an analysis of the Jungian classification personality types.

A basic result of the Lowen model is the discovery of two macroscopic processing circuits, in essence, cycles, among the capacities. These circuits weakly, but importantly, communicate with each other to create the response patterns we call personality types. If further support for their existence is found by or in psychological research, the presence of such macroscopic circuits would be a further argument for using the Lowen model as one end point of what amounts to a variational problem: How to construct the unconscious mesoscale, starting with a mass of primitive processors and ending with the preconscious capacities.

The suggestion that the action-lens structure constitutes a useful basis for the construction of Lowen-style AGI systems is being explored in these ways: The Lowen capacities do not all have identifications with established information processing technologies; such identifications need to be found and formalized. Can the action-lens model be used to generate larger structures that act as the classical specialized information processors of the Lowen model? Is it practical to use recursive function theory to generate and control the development of large scale structures effectively?

References

- Lowen, W. 1982. *Dichotomies of the Mind*. New York, NY: John Wiley.

HELEN: Using Brain Regions and Mechanisms for Story Understanding to Model Language as Human Behavior

Robert SWAINE

CTO, MindSoft Bioware / robertswaine@yahoo.com

Abstract. A new cognitive model is presented for large-scale representation of episodic situations and for manipulating these representations using the model's innate natural language processing mechanisms. As formulated, and implemented in part, the purpose of the model seeks to attain basic child level cognitive behavior of situational understanding. This includes general domain learning, by assigning internally-relevant, though subjective, value to common experience (input situations), and by being taught how to analyze/synthesize component representations of those input situations. Ultimately this will allow it to learn and autonomously create situations with meaning from those components to best fit problem situations. The current model is written in C++ as a visual interface and is implemented as a story understander with question answering and language generation capability.

Introduction

Two of the key difficulties in achieving a *human-level* intelligent agent are those of: first providing a flexible means for gaining common, wide-ranging human experience, and secondly applying common sense behavior and meaning for those and similar, but novel template experiences. The first problem is managed if there is a human analogous means flexible enough to *represent* many common human experience. The second problem is better handled if there is a model that can suitably link and implement the behavioral and meaning value *mechanisms* that relate to these common experience templates.

The model presented here provides one such solution, where the representation method and behavioral mechanisms are tied into one design. The model is a novel cognitive architecture called HELEN-KLR (namesake: Helen Keller, blind-deaf, **H**ierarchical **E**vent-language **L**earning **N**etwork for **K**nowledge **R**epresentation), and follows along the lines of the SOAR and ACT-R cognitive models. The layout of the model is inspired by a number of global and regional circuit maps in mammalian neuroanatomy.

Representation

The HELEN model representation is based on a framework of neuronal registers that regionally index patterns of synaptic weights and activation levels, and reciprocally map input/output through intervening filter layers. The registers' content-maps can access, read, and modify the contents of other registers. Using these registers, HELEN represents portions of human "experience" in two major interconnected divisions: (1) a

BODY, the animal's *presence* in a "world" environment, including the internal body state, and (2) a WORLD apart from the body that is conveyed via sensory perception. Both divisions are further subdivided into two relational domains: (A) the modeled-sensory content of the input types and (B) the relationships between any referenced content. The resulting four domains form the key representational structure of the HELEN model's version of human experience. The domains are named: **BODY [Value, Role]** and **WORLD [Relation, Feature]**.

Value reflects all sensed internal body states, e.g. goal, emotional body state, valence, consequence, etc. **Role** represents all effort levels and patterns of directional intent for planned actions, e.g. action sets, behavioral responses, skill, functions, etc. **Relation** are static and dynamic arrangement patterns of attention regions relative to one region (reference or view point), e.g. motion patterns, structural layout, size, accumulated change/quantity, etc. **Feature** contains all sensory forms and grouped representations having some component external to the body as they are modeled, e.g., visual, tactile, olfactory, models, etc. The four domains form the mixed content of a register matrix for any objectifiable **Instance** by the weight pattern set by activation within the specific regions of uninhibited attention, i.e., all domains content can be made into an "object" as an instance.

Each domain is hierarchically organized along a number of sensory modalities (input types); sensors path for each domain converge and interlink at different levels of the hierarchy to form various modal and cross-modal concept models. Some sensors are dedicated to a given do-

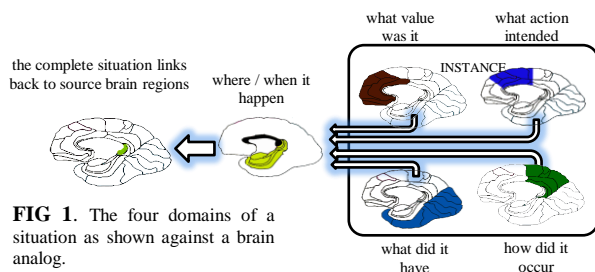


FIG 1. The four domains of a situation as shown against a brain analog.

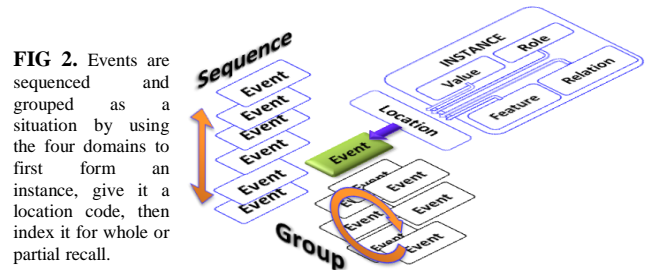


FIG 2. Events are sequenced and grouped as a situation by using the four domains to first form an instance, give it a location code, then index it for whole or partial recall.

main, while others share sensor data. FIG 1. and 2 show a basic layout of the domains based on a simple brain analogue. Outside the four domains, there are two other regions in the HELEN model: **Location** (non view-point referenced world position) and **Event**.^[1,2] Location is represented *via changes in* Relation and Feature domains, while Event is also similarly analyzed; they are both explicitly outside *direct* apprehension via the four domains. The model codes instances with a location, and then indexes co-occurring and co-located groups of instances as events. Events are in turn, indexed both sequentially and spatially as situations; situations are likened to episodes, or frames in some models. ^[3,4]

Mechanisms

In the model, mechanisms for simple and complex behavior can factor (analyze content) and manipulate (modify associated attributes of) the four domains, as well as steer the attention and focus-detail of content in situations perceived by the model so as to attach values. The structural layout of connections and the operation of the register elements comprise the major functional mechanisms, i.e., a large number of HELEN's mechanisms *innately arise* from it's representation method's register map. At the low-end, the predominant mechanisms include a "Long Term Potentiation" (LTP/D) decay spectrum for memories; synaptic-field copying to store activation-weight patterns; and "Combine-Compare-Contrast" circuits for reasoning and cognitive blending (*situational juxtapositioning* - used to synthesize or analyze). At the higher-end, dominant mechanisms include two regional indexing modules for learning and mapping meaning (*values*) to objects (*features*), and for planning, i.e., filtering then selecting actions by mapping intentions (*roles*) to position (*relation*). Two regions serve to globally link and then regionally store current and long term situation index maps. Many of the mechanisms in the model apply to its use in language understanding, i.e., giving situational meaning.

Taking a brief look at one mechanism, attention, the value system cycles between an inward and outward mode for action-response, observe-acknowledge, and store-recall modes. All situations have a salient or natural attention, a "what". Mitigated by value, natural attention serves as the "subject" of a sentence, the current "area" of a discussion, or the "what" of any event or object. These invariant or changing salient attention regions form time/space patterns that act as small/large situational borders. In the model, when taking in a situation, the size and duration of the

natural attention region will also determine focus *depth* (detail) and focus *rate* used to accept the current and next situation. When HELEN is recalling a situation (externally prompted or desired), the reverse occurs: the attention region and focus detail of the recalled situations will attempt to set the attention region and focus level to their salient content, unless inhibited (internal/external focus mode).

Remarks and Future Work

At the lowest, primitive-end, HELEN can *conceptualize*, i.e. represent, general sensory models such as light-dark, direction, change, body-state. At the highest-end, the largest situation that HELEN conceptualizes using it's representational domains is the "game-story" concept. These are groups-sequence steps of situational intentions and moves (goals&plans) and sets-sequence paths of a arranged space and character objects (setting&content). Based on it's biological analogue, one implication is that a "story-understander / game-player" is a viable method for approaching *human/mammal-like* cognitive behavior: HELEN attempts to fit (make sense of) *all* situations which it values, into some simple/intricate event-coherent "story" or "game" model (scheme); bad fitting input raises questions in HELEN. Thus problem-solving is handled as game-story situations (rules as Role-Relation links). This is equivalent to understanding the "games" and "stories" in physics or programming or dating, when reading and summarizing the content of a physics article, laying out the execution steps in an AI program, or making sense of your child's dating troubles. Additional to these, HELEN's four domain model serves as a flexible subject-ontology generator, metaphor map, and is used to dynamically categorize novel situational inputs on-the-fly into the domains.

In parallel with completing the implementation of the model with a more detailed representation and mechanisms, the model will incorporate a large situational corpus of human-centric situations of a child in the formative years, e.g. family and naive physics events, playing the spectrum of *roles* from observer to actor. Furthermore, this corpus of related/independent situations will be given most emphasis on common (cultural and language relevant) rules for manipulating (e.g. expressing) situations. As a dictionary of human childhood "experience", it will serve as the source of an initial meaning (value) template when HELEN is given situations presented in natural language as part of a novice-to-expert curriculums in various fields.

References

- [1] Conjunctive representation of position, direction, and velocity in entorhinal cortex. Sargolini F, Fyhn M, Hafting T, McNaughton BL, Witter MP, Moser MB, Moser EI. Science. 2006 May 5;312(5774):680-1
- [2] Place cells, grid cells, and the brain's spatial representation system. Moser EI, Kropff E, Moser MB. Annual Review of Neuroscience 2008; 31:69-89
- [3] Frame Activated Inferences in a Story Understanding Program. Peter Norvig IJCAI 1983: 624-626
- [4] Six Problems for Story Understanders. Peter Norvig AAAI 1983: 284-287

Holistic Intelligence: Transversal Skills & Current Methodologies

Kristinn R. Thórisson & Eric Nivel

Center for Analysis and Design of Intelligent Agents / School of Computer Science
Reykjavik University, Kringlunni 1, 103 Reykjavik, Iceland
{thorisson, eric}@ru.is

Abstract

Certain necessary features of general intelligence are more system-wide than others; features such as attention, learning and temporal grounding are *transversal* in that they seem to affect a significant subset of *all* mental operation. We argue that such transversal features unavoidably impose fundamental constraints on the kinds of architectures and methodologies required for building artificially intelligent systems. Current component-based software practices fall short for building systems with transversal features: Artificial general intelligence efforts call for new system architectures and new methodologies, where transversal features must be taken into account from the very outset.

Introduction

Animal intelligence, the best example of *general intelligence* that most agree on classifying as such, is a remarkable conglomeration of different sets of skills that work together in ways that make for a coordinated and coherent control of the limited resource we call a body. Looking at the progress AI in its first 50 years, advances have been slower than expected: we certainly have not yet reached a level of artificial intelligence anywhere near that of an animal. The nature of a scientifically studied phenomenon is the main determinant of the kinds of approaches relevant for its study. In the case of outer space lack of opportunity for experimentation hampered progress for millennia. In the study of general intelligence – whether for scientific inquiry or building practical machines – a major barrier is complexity. It behooves us to look very carefully at our research methods in light of the subject under study, and in particular at whether the tools and approaches currently used hold promise to deliver the advances we hope for.

The bulk of software engineering practices today focus on what one might call "component methodologies", such as object orientation and service-oriented architectures, to take two examples. Much of AI research is based on these standard practices as well, as is cognitive science. The modeling methodology relies on certain atomic units being put together tediously and by hand, in such a way as to create conglomerates of hand-crafted interacting units. The evidence from robotics research over the last several decades shows progress to be slow and limited to basic bodily control like balance (cf. [3]). As these are now closer than ever to being solved, researchers' attention is turning to integration; in this approach of putting together

well-understood "hand-made" modules in a LEGO-like fashion, progress will predictably continue at the same pace as prior efforts, being a linear function of the component methodology. Some may be able to live with that, at least as long as results are guaranteed. However, this is not even a given: A more likely scenario is that only slightly more complex intelligences than what we have in the labs today will be built by this method; sooner rather than later the complexity of integration becomes overpowering and all efforts grind to a halt. To see this one need only look at the results of putting together networks (cf. [2]) or large desktop applications (cf. [1]): The difficulty of designing such systems to run and scale well shows the inherent limitations of current software methodologies. And these systems are quite possibly several orders of magnitude simpler than those required for general intelligence.

The Architecture *is* the System

What building blocks we use, how we put them together, how they interact over time to produce the *dynamics* of a system: The discussion ultimately revolves around *architecture*. The types of system architectures we choose to explore for building intelligent systems will determine the capabilities of the system as a whole. The nature of these architectures will of course directly dictate the methodologies that we use for building them. One issue that cuts at the core of intelligence architectures is that of transversal functions – functions that affect the design and organization of the whole system. Three examples are dynamic allocation of attention across tasks, general learning and temporal awareness. A robot for the home, as an example, requires a high degree of cognitive flexibility: Not only should it be able to do the dishes, the laundry, clean, cook and play with the cat, it must be capable of *moving seamlessly between these tasks*. Such seamlessness builds on deep transversal functionality, the interwoven execution of attention, knowledge and learning of new contexts. An inflexible system can easily be thrown off by unseen variations in even the most mundane work environments: The cat jumping into the washing machine, a child sticking a screwdriver into an electrical outlet. Unless the machine has very flexible ways of directing its attention and – in closely coordinated fashion – switching between tasks quickly and efficiently, in fine-tuned

coordination, with proper event prioritization, the household robot of your dreams could quickly turn into a nightmare. To continue with the example, unless we invent some amazing "superglue software" that can dynamically (a) *bind together* the separate skill sets, (b) handle *smooth transition between tasks* within and between skill sets, (c) *learn* new combinations of actions and perceptions from different skill sets, (d) *identify "new"* (unspecified) things and (quickly) guesstimate the nature and implications of these, (e) automatically *control attention* of both its internal and external state, and during these (f) understand and *manage the passing of time*, a machine working in everyday environments will prove extremely dangerous and probably incapable of working amongst people.

The criticism of component-based approaches and standard software engineering methodologies apply to – at the risk of overgeneralizing perhaps only slightly – all architecturo-methodological approaches proposed to date for robots and other single-mind intelligent systems. Subsumption architectures, blackboard architectures, production systems, schema-based architectures – all have been implemented in the last decades in ways that seem unlikely to scale to the kinds of flexibility we would require of any artificial system with general intelligence. Progress towards artificial general intelligence cannot rely (solely) on current approaches, as these do not show sufficient promise for addressing key architectural characteristics of general intelligence.

Towards Generally Intelligent Systems

A generally intelligent machine must be able to learn anything, meaning essentially an enormously large range of things, regarding the world as well as itself. This calls for system-wide general-purpose learning mechanisms. By *system-wide learning* we mean a process capable of identifying and recognizing patterns of interaction between components regardless of their "location" in the architecture. Further, any practical, implementable intelligence will always be bound by limited CPU power and memory. To learn a large range of things it needs to be able to direct its computational resources towards achieving certain goals, and distractions need to be prioritizable and ignorable. This means that the machine needs a *general attentional mechanism*. Such a mechanism must permeate the very structure of the system and be integrated at a fundamental level of the system's operation. A third fundamental feature that has to be engineered into the very fabric of an artificial general intelligence is *temporal grounding*. For engineers, "real-time" means the time as it elapses in the real world. Hard real-time systems are imposed real-world deadlines by their designer – without information that allows systems to understand their purpose or meaning. Intelligent autonomous systems, on the other hand, are bound to the laws governing the

maximization of their utility function. To operate in the world – in real-time – means therefore something very different here: machine-time must be expressed by the *semantics* of in the system-world's state space. Intuitively, internal processes of the system are mapped onto world-time with regards to their contribution towards achieving goals. For example, a deadline in world-time could be grounded in a (time-bounded) *process*, getting to the bank before it closes, and contextualized by the goal get money to pay the baby-sitter. Such *temporal grounding* can affect pretty much any action, whether mental or physical, of a generally intelligent system and must therefore, by definition, be transversal.

Transversal learning, attention and temporal grounding is a requirement for *all key mental skills/processes*, including planning, motor control, prediction, understanding, etc. Whether one thinks achieving this is difficult, easy or impossible, it stands to reason that these requirements will have enormous implications for the cognitive architecture of a system. The implications are twofold. First, instead of using static components we must design architectures in terms of *dynamic* "components" – that is *processes* – that would instantiate the transversal functionalities cited above according to needs and contexts, both also dynamic. Second, *learning new tasks* means *instantiating new processes*, and architectures must provision for the dynamic management (creation and decay) of such processes. In light of this it should be clear that analogies between software architecture and electronic circuits is grossly inadequate for generally intelligent systems.

Continued ignorance of transversal functionality by the research community can only mean further delay on our path towards artificial general intelligence. We must factor these functionalities in from the very outset; they are fundamental and must directly guide our efforts in developing the architectural and methodological principles for building machines with general intelligence. Efforts by the authors to incorporate these principles in implemented, operational architectures are described in [4].

References

- [1] Abreu F.B. and Carapuça R. (1994). Object-Oriented Software Engineering: Measuring and Controlling the Development Process. *Proc. of 4th Int. Conference on Software Quality*, 3-5 October, McLean, VA, USA.
- [2] Hall N.R. and S. Preiser (1984). Combined Network Complexity Measures. *IBM J. Res. Dev.*, **28**(1):15-27.
- [3] Kagami, S., F. Kanehiro, Y. Tamiya, M. Inaba, H. Inoue (2001). AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots. In Bruce R. Donald, Kevin M. Lynch, Daniela Rus (Eds.), *Algorithmic and Computational Robotics*. New York: A.K. Peters.
- [4] Nivel, E. & Thórisson, K. R. (2008). Self-Programming: Operationalizing Autonomy. *This volume*.

Achieving Artificial General Intelligence Through Peewee Granularity

Kristinn R. Thórisson & Eric Nivel

Center for the Analysis and Design of Intelligent Agents / School of Computer Science
Reykjavik University, Kringlunni 1, 103 Reykjavik, Iceland
{thorisson, eric}@ru.is

Abstract

The general intelligence of any autonomous system must in large part be measured by its ability to automatically learn new skills and integrate these with prior skills. Cognitive architectures addressing these topics are few and far between – possibly because of their difficulty. We argue that architectures capable of diverse skill acquisition and integration, and real-time management of these, require an approach of modularization that goes well beyond the current practices, leading to a class of architectures we refer to as *peewee-granule systems*. The building blocks (modules) in such systems have simple operational semantics and result in architectures that are heterogeneous at the cognitive level but *homogeneous at the computational level*.

Introduction

Looking at the software architecture of present large-scale AI systems reveals a rather clear picture: A majority is built on principles of standard industrial software component methodologies. As we have argued elsewhere [7,9] such methodologies do not support sufficient architectural flexibility when it comes to building intelligent systems, in large part because they do not support well incremental expansion or *automated architectural construction*. We have developed a method for intelligent system construction, Constructionist Design Methodology (CDM) [10], that produces cognitive architectures exhibiting greater flexibility in expansion than typical of architectures of similar size [8] and better support for system integration [5]. In the last few years we have been moving towards architectures built out of ever-smaller components, or modules. Here we discuss what we see as a general trend towards “*peewee*”-granule systems – architectures with very small-grain components – and why we see this as a promising direction for artificial general intelligence.

Medium Granularity in Ymir / CDM

Ymir is a cognitive proto-architecture [11] from which the CDM was originally derived. One of Ymir's advantages is its addressing multiple skill integration in a realtime-capable system with multimodal output generation. Over the last decade the principles of Ymir and CDM have been

used to build several relatively large systems including a multimodal realtime interactive character, a realtime dialogue system [2] and a cognitive architecture for the Honda ASIMO robot [5]. These systems, all based on the idea of a fairly large set of small functional units (called *modules*, each typically less than 100 lines of code) interacting via blackboards, were developed incrementally according to the development steps set forth in the CDM. In Ymir modules are loosely coupled through message passing; messages are semantically self-describing (the content of modules' inputs and outputs is explicit in the message types). Typically a module's function lies at the cognitive level; any psychologically distinguishable behavior (e.g. taking turns in dialogue, reaching to grasp an object, etc.) is done through cooperation/interaction of 50-80 such modules. Ymir postulates three priority layers of modules, each layer having a particular upper bound on the perception-action loop time: The Reactive Layer with ~ 100 - 400 msec; the Process Control Layer with ~ 400 - 2000 msec; and the Content Layer from 2k msec and up. Ideally, modules are stateless (state is completely contained in the historical flow of messages); however, we have found that it is difficult to stay away from saving state in some subset of a system's modules.

The important benefits of Ymir's CDM principles and medium-granularity include better scaling of performance, increased breadth and more organizational flexibility at runtime, as reported for numerous systems (e.g. [2,5,7,8,10]). While recent work has shown Ymir-like architectures to be able to learn dynamically at runtime [2], runtime changes in Ymir-style architectures at present do not involve *new functions* (in terms of new modules); they are limited to changes in the behaviors of, and interactions between, already-existing modules. If we want to achieve complex, evolving systems that can self-improve *significantly* over time, however, *automatic synthesis* of new components must be made possible. Automatic management of self-improvement – via *reorganization of the architecture itself* – can only be achieved by giving the system instructions on how to measure its own performance and providing it with methods for introducing architectural changes to improve its own performance on those measures. Such *models of self* are very difficult to achieve in systems built with known software

methodologies – as well as the CDM. This leads us to the importance of *computational homogeneity*.

Towards Peewee Granularity

As shown with Ymir's priority layers [11] (see also [6]) the role of structures is to implement observation/control feedback loops; the scale of complexity levels is thus closely linked to the scale of response times: we need to exert a fine control over the process synthesis to tune accurately its constituents (sub-structures and sub-processes) at any relevant scale. Control accuracy over processes and process construction can be achieved only if (a) the granularity of program interaction is as fine as the size of the smallest model and (b) the execution time of models is much lower than the program interaction the models intend to control. For systems with shortest cognitive response times (typically 250-500 msec) this may mean grains of no longer than a few CPU cycles long.

Ikon Flux is a proto-architecture for building fully autonomous systems [3]. The details of Ikon Flux have been described elsewhere; here we will discuss two of its most salient traits, *computational homogeneity* and *peewee granularity* (very small modules). Ikon Flux has been designed to build systems that *embody a continuous process of architectural (re-)synthesis*. Such systems are engaged – in realtime – in observation/control loops to steer the evolution of their own structures and processes over short and long time horizons. In Ikon Flux, structures and processes result from a bottom-up synthesis activity scaffolded by top-down models: it finds its raw material in low-level axioms (commands from/to sensors/actuators, programming skills, etc.) while being regulated and structured by (initially) man-made bootstrap code. As Ikon Flux systems expand in functionality and scope the models necessary to control synthesis grow in complexity; to cover the whole spectrum of a system's operation they must encompass both low-level and higher-order structures/processes. An autonomous system has thus to evolve these heterogeneous models over time along a quasi-continuous scale of complexity levels. It is a practical impossibility to implement an architectural model for *each* of these levels – which in most cases cannot be known in advance. However, providing a uniform model that can self-improve is a challenge since the operational semantics grow significantly in complexity with the atomic set of system operations (module types). This can be solved by employing a *homogenous computational substrate* consisting of a small amount of atomic operational elements, each of peewee size. In Ikon Flux these are rewrite terms. Systems built in Ikon Flux grow massive amounts of (stateless) concurrent rewriting programs organized to allow composition of structures/processes of arbitrary size and architecture. Other research has acknowledged the need for computational homogeneity (cf. [1,4]), albeit to a lesser extent than Ikon Flux.

Architectures like Ymir [2,5,11] and others [1,4] have shown the benefits of medium-size granularity. While these systems can be expanded in performance, such expansion tends to be linear, due to an operational semantics complexity barrier. Ikon Flux presents a next step towards *massive amounts of small components*, embodying hundreds of thousands of peewee-size modules [3]. Yet Ikon Flux demonstrates *cognitive heterogeneity* on top of this computationally homogeneous substrate. As a result, systems built in Ikon Flux exhibit deep learning of new skills and integration of such skills into an existing cognitive architecture. We believe peewee granularity is a promising way to simplify operational semantics and reach a computational homogeneity that can enable automated architectural growth – which in itself is a *necessary step towards scaling of cognitive skills* exhibited by current state-of-the-art architectures. Only this way will we move more quickly towards artificial general intelligence.

References

- [1] Cardon A. 2003. Control and Behavior of a Massive Multi-agent System In W. Truszkowski, C. Rouff, M. Hinchey eds. WRAC 2002, LNAI 2564, 46-60. Berlin Heidelberg: Springer-Verlag.
- [2] Jonsdottir G.R., Thórisson K.R. and Nivel E. 2008. Learning Smooth, Human-Like Turntaking in Realtime Dialogue. *Intelligent Virtual Agents (IVA)*, Tokyo, Japan, September 1-3.
- [3] Nivel, E. 2007. Ikon Flux 2.0. Reykjavik University Department of Computer Science Technical Report RUTR-CS07006.
- [4] Pezzulo G. and Calvi G. 2007. Designing Modular Architectures in the Framework AKIRA. In *Multiagent and Grid Systems*, 3:65-86.
- [5] Ng-Thow-Hing V., List T., Thórisson K.R., Lim J., Wormer J. 2007. Design and Evaluation of Communication Middleware in a Distributed Humanoid Robot Architecture. *IROS '07 Workshop Measures and Procedures for the Evaluation of Robot Architectures and Middleware*. San Diego, California.
- [6] Sanz R., López I., Hernández C. 2007. Self-awareness in Real-time Cognitive Control Architectures. In *AI and Consciousness: Theoretical Foundations and Current Approaches*. AAAI Fall Symposium. Washington, DC.
- [7] Thórisson K.R. and Jonsdottir G.R. 2008. A Granular Architecture for Dynamic Realtime Dialogue. *Intelligent Virtual Agents (IVA)*, Tokyo, Japan, September 1-3.
- [8] Thórisson K.R., Jonsdottir G.R. and Nivel E. 2008. Methods for Complex Single-Mind Architecture Designs. *Proc. AAMAS*, Estoril, Portugal, June.
- [9] Thórisson K. R. 2007. Integrated A.I. Systems. *Minds & Machines*, 17:11-25.
- [10] Thórisson K.R., Benko H., Arnold A., Abramov D., Maskey, S., Vasekaran, A. 2004. Constructionist Design Methodology for Interactive Intelligences. *A.I. Magazine*, 25(4):77-90.
- [11] Thórisson K.R. 1999. A Mind Model for Multimodal Communicative Creatures and Humanoids. *International Journal of Applied Artificial Intelligence*, 13(4-5): 449-486.

Author Index

Achler, Tsvi	198	Lathrop, Scott	97
Amir, Eyal	198	Lebiere, Christian	103
Armstrong, Blair	132	Liu, Daphne	108
Baum, Eric	1, 200	Looks, Moshe	114
Bittle, Sean	7	Loosemore, Richard	120
Bringsjord, Selmer	202	Lorincz, Andras	126
Bugaj, Stephan	31	MacInnes, Joseph	132
Cassimatis, Nicholas	144	Marinier, Robert	91
Chandrasekaran, B.	85	Miles, Jere	210
Chella, Antonio	13	Miller, Matthew	138
Cree, George	132	Miner, Don	212
Crossley, Neil	19	Murugesan, Arthi	144
de Garis, Hugo	25	Nivel, Eric	140, 211, 213
desJardins, Marie	212	Pare, Dwayne	132
Fox, Mark	7	Pickett, Marc	212
Gaglio, Salvatore	13	Pitt, Joel	73
Ghosh, Sujata	37	Reed, Stephen	156
Goertzel, Ben	31, 73, 114	Samsonovich, Alexei	214
Gonzalez, Cleotilde	103	Saraf, Sanchit	37
Gros, Claudius	204	Schmid, Ute	19, 55, 162
Gust, Helmar	43	Schubert, Lenhart	108
Hall, John	49	Schwering, Angela	43
Hibbard, Bill	206	Sellman, George	73
Hitzler, Pascal	208	Stoytchev, Alexander	138
Hofmann, Martin	19, 55, 162	Surowitz, Eugene	216
Hutter, Marcus	61, 67	Swaine, Robert	218
Ikle, Matthew	73	Tashakkori, Rahman	210
Johnston, Benjamin	79	Thorisson, Kristinn R. ..	150, 220, 222
Joordens, Steve	132	Tripodes, Peter G.	168
Kitzelmann, Emanuel	19, 55, 162	Wang, Pei	174, 180
Krumnack, Ulf	43	Warwick, Walter	103
Kühnberger, Kai-Uwe	43, 208	Waser, Mark	186
Kurup, Unmesh	85	Williams, Mary-Anne	79
Löwe, Benedikt	37	Wintermute, Samuel	192
Laird, John	91, 97	Wong, Peter	138
Langley, Pat	91	Wray, Robert	91

Second Conference on Artificial General Intelligence
AGI 2009, Arlington, Virginia, USA | March 6-9, 2009

Artificial General Intelligence