A Prototype Implementation of a Multi-Agent Deontic Action Language

Interactive Logic: Games and Social Software
7th Augustus de Morgan Workshop
King's College London
November 4-7, 2005

Adam Zachary Wyner
Department of Computer Science
King's College London, London, UK

adam@wyner.info
www.dcs.kcl.ac.uk/pg/wyner

The Objective

To abstractly simulate agents acting in reaction to one another and with respect to deontic specifications (obligations, permissions, and prohibitions) on actions.

A Simple Example

Bill and Jill are in a room, but don't like one another. Will obligates Bill to remain in the room (he wants Bill to keep watch over it). If Bill remains in the room, Will pays Bill $\pounds 1$. If Bill leaves the room, he is obligated to pay Will $\pounds 5$. Will obligates Jill to leave the room (he wants her to leave). If she leaves, Will is obligated to pay her $\pounds 5$. If she remains in the room, she is obligated to pay Will $\pounds 5$. Jill prefers to fulfill her obligations (to save money), but if Bill has already violated his obligation, then Jill prefers to violate her obligation (she prefers to remain in the room but only so long as Bill is not there, even if it costs her).

Introduction

- Legal contract formation, execution, and monitoring for the purposes of E-Commerce (Daskalopulu and Maibaum (2001)) and Business Process Management.

Introduction

- Legal contract formation, execution, and monitoring for the purposes of E-Commerce (Daskalopulu and Maibaum (2001)) and Business Process Management.
- Theoretical background – static deontic logic (Carmo and Jones 2002) and dynamic deontic logic (Khosla and Maibaum 1987, Meyer 1988).

Introduction

- Legal contract formation, execution, and monitoring for the purposes of E-Commerce (Daskalopulu and Maibaum (2001)) and Business Process Management.
- Theoretical background – static deontic logic (Carmo and Jones 2002) and dynamic deontic logic (Khosla and Maibaum 1987, Meyer 1988).
- Sketch an implementation in Haskell (a functional programming language) which simulates actions executed by agents with respect to deontic specifications.

Introduction

- Legal contract formation, execution, and monitoring for the purposes of E-Commerce (Daskalopulu and Maibaum (2001)) and Business Process Management.
- Theoretical background – static deontic logic (Carmo and Jones 2002) and dynamic deontic logic (Khosla and Maibaum 1987, Meyer 1988).
- Sketch an implementation in Haskell (a functional programming language) which simulates actions executed by agents with respect to deontic specifications.
- Suggest extensions for interaction.

Overview of the Language

- Actions and Agents.

Overview of the Language

- Actions and Agents.
- Productive lexical relationships.

Overview of the Language

- Actions and Agents.
- Productive lexical relationships.
- Deontic specifiers (0bligation, Permission, Prohibition) are functions on Agentive Actions which give rise to flags for violation or fulfillment.

Overview of the Language

- Actions and Agents.
- Productive lexical relationships.
- Deontic specifiers (0bligation, Permission, Prohibition) are functions on Agentive Actions which give rise to flags for violation or fulfillment.
- Reasoning with respect to the flags.

Overview of the Language

- Actions and Agents.
- Productive lexical relationships.
- Deontic specifiers (0bligation, Permission, Prohibition) are functions on Agentive Actions which give rise to flags for violation or fulfillment.
- Reasoning with respect to the flags.
- Contract states are lists of deontically specified agentive actions.

Overview of the Language

- Actions and Agents.
- Productive lexical relationships.
- Deontic specifiers (0bligation, Permission, Prohibition) are functions on Agentive Actions which give rise to flags for violation or fulfillment.
- Reasoning with respect to the flags.
- Contract states are lists of deontically specified agentive actions.
- Actions are executed with respect to States-Of-Affairs (Contract States, Histories, Properties, and world/time indices).

Agentive Actions

They have fields for an Action and an Agent:

(action = (label = "Action6", xcond = ["prop1", "prop7", "prop5"], ycond = ["prop3", "neg-prop4", "neg-prop6"]), agent = "Jill")

Agentive Actions

where the action has a label, preconditions xcond, and postconditions ycond. The preconditions condition execution of the action, while the postconditions specify what holds after the execution:

(action = (label = "Action6", xcond = ["prop1","prop7","prop5"], ycond = ["prop3","neg-prop4","neg-prop6"]), agent = "Jill")

Agentive Actions

We can have another agent:

(action = (label = "Action6", xcond = ["prop1","prop7","prop5"], ycond = ["prop3","neg-prop4","neg-prop6"]), agent = "Bill")

Action Negation

If Jill fulfills her obligation by executing Action6, by what actions does Jill violate her obligation?

If we input the action to the opposition function:

OPP(action = (label = "Action6", xcond = ["prop1","prop7","prop5"], ycond = ["prop3","neg-prop4","neg-prop6"]), agent = "Jill")

Action Negation

We can calculate a particular action which gets an opposite result in some sense:

(action = (label = "Action7", xcond = ["prop1","prop7","prop5"], ycond = ["prop3","neg-prop4","prop6"]), agent = "Jill")

Deontic Specifications on Actions

When we apply a deontic operator to an agentive action:

obligatedCompFlag (action = (label = "Action6", xcond = ["prop1","prop7","prop5"], ycond = ["prop3","neg-prop4","neg-prop6"]), agent= "Jill")
testLexActions02 []

Deontic Specifications on Actions

We get out a list of records which relate the deontically specified action:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"), (actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

to the deontic specifier:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"), (actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

and an agent:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"),(actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

We indicate the action executed:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = Fulfilled"),(actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

and the valuation flag which execution of the action gives rise to:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"),(actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

These define contract states, which can be modified over time and which can be given *consistency constraints*:

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"), (actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated")]

Deontic Specifications on Actions

Contract states can be more complex, specifying other actions, agents, and deontic specifiers (prohibitions and permissions).

[(actionDone = "Action6", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Fulfilled"), (actionDone = "Action7", agent = "Jill", deonticSpec = "Obligated", onSpec = "Action6", valueFlag = "Violated"), (actionDone = "Action7", agent = "Bill", deonticSpec = "Permitted", onSpec = "Action7", valueFlag = ""), (actionDone = "Action9", agent = "Bill", deonticSpec = "Prohibited", onSpec = "Action9", valueFlag = "Violated")]

States-of-Affairs

Actions are executed with respect to States-of-Affairs, which are comprised of a Contract State

(contractFlagState=[(actionDone="Action6", agent="Jill", deonticSpec=" Obligated", onSpec="Action6", valueFlag="Fulfilled"),(actionDone="Action 7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag ="Violated")], history=[(actionDone="Action2", agent="Bill", deonticSpec="", onSpec="", time=1, valueFlag="", world=0)], properties=["prop1","prop7","prop5","neg-prop6"], time=1, world=0)

States-of-Affairs

And a History, which records what has been executed:

(contractFlagState=[(actionDone="Action6", agent="Jill", deonticSpec=" Obligated", onSpec="Action6", valueFlag="Fulfilled"),(actionDone="Action 7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag ="Violated")], history=[(actionDone="Action2", agent="Bill", deonticSpec="", onSpec="", time=1, valueFlag="", world=0)], properties=["prop1","prop7","prop5","neg-prop6"], time=1, world=0)

States-of-Affairs

And Properties, which condition the execution of the action and are updated by the action:

(contractFlagState = [(actionDone="Action6", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag="Fulfilled"), (actionDone="Action 7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag ="Violated")], history=[(actionDone="Action2", agent="Bill", deonticSpec="", onSpec="", time=1, valueFlag="", world=0)], properties=["prop1", "prop7", "prop5", "neg-prop6"], time=1, world=0)

Action Execution

doAgentiveActionSOA :: AgentiveAction -> SOA -> SOA

Jill executes Action7. Check SOA properties satisfy the action. Compare the Action with the Contract State for a match. If we find the action in the Contract State, introduce the matching information from the Contract State into the history. This indicates that Jill has violated her obligation. Otherwise, just indicate the action has been executed. Update the properties and time.

Action Execution

After Jill's Execution of Action7, the history, the properties, and the time have been updated:

(contractFlagState=[(actionDone="Action6", agent="Jill", deonticSpec=" Obligated", onSpec="Action6", valueFlag="Fulfilled"),(actionDone="Action 7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag ="Violated")], history=[(actionDone="Action2", agent="Bill", deonticSpec="", onSpec="", time=1, valueFlag="", world=0), (actionDone="Action7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", time=2, valueFlag="Violated", world=0)], properties=["prop1", "prop3", "neg-prop4", "neg-prop6"], time=2, world=0)

Sanction and Reward

Information in the history can trigger modification of the Contract State: because Jill has violated her first obligation, Jill is then prohibited from executing Action12, and we remove her previous obligation.

(contractFlagState=[(actionDone="Action12", agent="Jill", deonticSpec=" prohibited", onSpec="Action12", valueFlag="Violated")], history=[(actionDone="Action2", agent="Bill", deonticSpec="", onSpec="", time=1, valueFlag="", world=0), (actionDone="Action7", agent="Jill", deonticSpec="Obligated", onSpec="Action6", time=2, valueFlag="Violated", world=0)], properties=["prop1", "prop3", "neg-prop4", "neg-prop6"], time=2, world=0)

Other Components Implemented

- Inference from deontically specified actions.

Other Components Implemented

- Inference from deontically specified actions.
- Contrary-to-duties (or other paradoxes).

Other Components Implemented

- Inference from deontically specified actions.
- Contrary-to-duties (or other paradoxes).
- Deontic specification on complex actions (negation, sequence, simultaneous, choice).

Extensions – Interaction

- Enrich deontic specifications to indicate a beneficiary or ’maleficiary’.

Extensions – Interaction

- Enrich deontic specifications to indicate a beneficiary or ’maleficiary’.
- Enrich the agent representation to indicate preferred or dispreferred courses of action.

Extensions – Interaction

- Enrich deontic specifications to indicate a beneficiary or ’maleficiary’.
- Enrich the agent representation to indicate preferred or dispreferred courses of action.
- Relativize the execution function to agent preferences.

-Ficiaries

Add to the deontic specifications on actions fields which indicate who is the beneficiary or maleficiary of the obligation.

Where Jill does Action6, she fulfills her obligation to do that action, and it benefits Bill, but 'malefits' her.

(bene="Bill", male="Jill", actionDone="Action6", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag="Fulfilled")

-Ficiaries

Can *underspecify* (simplified examples).

Jill is obligated to do Action6, which benefits Bill.

(bene="Bill", male="", actionDone="Action6", agent="Jill", deonticSpec="Obligated", onSpec="Action6", valueFlag="Fulfilled")

Jill is prohibited from doing Action6, which malefits herself.

(bene = "", male = "Jill", actionDone = "Action6", agent = "Jill", deonticSpec = "Prohibited", onSpec = "Action6", valueFlag = "Violated")

Agent Preferences

Enrich the representation of *Agents* to indicate positive preferences and negative preferences.

(posPrefs = [. . .], negPrefs = [. . .], agent = "Jill")

Agent Preferences

Jill prefers to execute any obligatory action which benefits her or any action which malefits Bill. She prefers not to execute any prohibition which ’malefits’ her.

(posPrefs = [(bene = "Jill", deonticSpec = "Obligated"), (male = "Bill")],
negPrefs = [(male = "Jill", deonticSpec = "Prohibited")], agent = "Jill")

Agent Preferences

Action execution is relativized to agent preferences. Assume an agent has to do something.

- If an available action is compatible with Jill's positive preferences, she executes that.
- Else, she executes an available action which is not specified with respect to preferences.
- Otherwise, she executes an action which she does not prefer.

Agent Preferences

Relativize an agent's preference structure to the history.

- If Bill has left the room, then Jill's preferences are A.
- Otherwise, Jill's preferences are (default) B.

Hohfeldian Jural Relations – future work

Hohfeld (1913) provided an analysis of *Jural Relations* – which are notions of *rights* and *powers* ((Jones and Sergot 1993), Daskalopulu (2001)):

- Jill has the right with respect to Bill such that Bill sees to it that he stays off Jill's land.
- Bill has the duty with respect to Jill such that Bill sees to it that he stays off Jill's land.

Hohfeldian Jural Relations – future work

Aside from systematically representing jural relations in terms of argument positions and *sees-to-it-that*, what else can we do with them?

$$right(p_i, p_j, E(p_j, \beta)) \equiv duty(p_j, p_i, E(p_j, \beta))$$

Hohfeldian Jural Relations – future work

Suggestion to analyze them in terms of roles and deontic specifications along the lines of what we have proposed. These can then be used to guide simulated agent interactions.

Closing Remarks

- Have shown a way to calculate using deontic notions and interacting agents.

Closing Remarks

- Have shown a way to calculate using deontic notions and interacting agents.
- The system can be used to calculate ’risk’ – the agent samples the state space with prospective trails of executions of actions against a contract (the contract state, contract changes, and the behavior of other agents) keeping ’score’ of violations and fulfillments.

Closing Remarks

- Have shown a way to calculate using deontic notions and interacting agents.
- The system can be used to calculate ’risk’ – the agent samples the state space with prospective trails of executions of actions against a contract (the contract state, contract changes, and the behavior of other agents) keeping ’score’ of violations and fulfillments.
- In turn, this can be used for *contract negotiation*, where the agents zero in on contract terms which provide them with an acceptable exposure to risk.

Acknowledgements

- Thanks to Hewlett-Packard Research Labs, Bristol, UK for funding my PhD Studentship.
- Thanks to Prof. Tom Maibaum and Prof. Andrew Jones for supervision, encouragement, and practical support.
- Thanks to audiences of previous versions of this work given at:

  King's College London, London, England
  University of Edinburgh, Edinburgh, Scotland
  Harvard University, Cambridge, Massachusetts, USA.

Some References

- J. Carmo and A. Jones (2002): “Deontic Logic and Contrary-to-duties”, in D. Gabbay and F. Guenthner (eds.) *Handbook of Philosophical Logic*, Dordrecht: Kluwer Academic Publishers, 203-282.
- A. Daskalopulu and T. Maibaum (2001): "Towards Electronic Contract Performance", in *Legal Information Systems Applications, 12th International Conference and Workshop on Database and Expert Systems Applications*, IEEE C. S. Press, pp. 771-777
- K. Doets and J. van Eijck (2005): *Haskell Road to Logic, Maths and Programming*. King’s College Publications, London, UK.
- D. Harel, D. Kozen and Tiuryn, J. (2000): *Dynamic Logic*, MIT Press.
- Khosla, S. and Maibaum, T. (1987): “The Prescription and Description of State-Based Systems”, in B. Banieqbal, H. Barringer, and A. Pneuli (eds.) *Temporal Logic in Specification*, Berlin: Springer-Verlag, 243-294.

Some References – Continued

- J.-J.Ch. Meyer (1988): “A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic”, *Notre Dame Journal of Formal Logic*, 29.1, 109-136
- A. Wyner (2004): “Maintaining Obligations on Stative Expressions in a Deontic Action Logic”, in A. Lomuscio and D. Nute (eds.) *Deontic Logic in Computer Science*, Berlin: Springer-Verlag, 258-274.
- A. Wyner (2005): “Rudiments of Computational Jurisprudence”, in H. Bunt, J. Geertzen, and E. Thijse (eds.) *Proceedings of the Sixth International Workshop on Computational Semantics*, Tilburg University, The Netherlands, Jan. 12-14, 2005, 416-420.
- A. Wyner (2005) “On Conditional Obligations”, ms. King’s College London, London, UK.

Questions and Thank You!