Renate Schmidt
Georg Struth (eds.)

Relations and Kleene Algebra in Computer Science

PhD Programme at RelMiCS/AKA 2006

Manchester, UK
August 28 - September 2, 2006

Proceedings

Research Report CS-06-09, Department of Computer Science, University of Sheffield

Preface

This volume contains the tutorials and the contributed extended abstracts of the PhD Programme at the 9th International Conference on *Relational Methods in Computer Science (RelMiCS-9)* and the 4th International Workshop on *Applications of Kleene Algebra (AKA'06)*. The Programme has been organised for the first time in association with RelMiCS/AKA. It was hosted by the School of Computer Science at the University of Manchester, UK, from August 29 to September 2, 2006 and included invited tutorials, a student session and attendance at the conference. Ten students have been selected for the programme by the organisers due to the relevance and quality of their submissions.

The tutorials—*Foundations of Relation Algebra and Kleene Algebra* by Peter Jipsen (Chapman University, USA) and *Relational Methods for Program Refinement* by John Derrick (University of Sheffield, UK)—introduced the theory of relational methods and presented an important application.

The student session allowed the participants to present and discuss their own work. The extended abstracts of their talks nicely reflect the diverse applications of relations and Kleene algebras in computing.

The RelMiCS/AKA conference series is the main forum for the relational calculus as a conceptual and methodological tool and for topics related to Kleene algebras. The programme of RelMiCS/AKA 2006 featured 25 contributed talks and three invited lectures: *Weak Kleene Algebra and Computation Trees* by Ernie Cohen (Microsoft, USA), *Finite Symmetric Integral Relation algebras with no 3-Cycles* by Roger Maddux (Iowa State University, USA), and *Computations and Relational Bundles* by Jeff Sanders (Oxford, UK). The proceedings are published as volume 4136 of the Springer LNCS series.

The organisers would warmly like to thank all those who contributed to the success of the programme: the tutorial and keynote speakers for accepting our invitation, the students for their interest in the programme and the local organisers at the University of Manchester for their dedicated help; the staff in the ACSO office, especially Bryony Quick, the conference secretary, Helen Spragg, and Iain Hart; the staff of the finance office; the technical staff and the building management team; as well as Zhen Li, David Robinson and Juan Navarro-Perez. We are very grateful to the UK Engineering and Physical Sciences Research Council for funding the entire PhD programme (grant EP/D079926/1) and we are pleased to acknowledge support of RelMICS/AKA 2006 by the London Mathematical Society, the British Logic Colloquium and the University of Manchester.

Manchester and Sheffield, August 2006

Renate Schmidt
Georg Struth

Table of Contents

Foundations of Relations and Kleene Algebras

Peter Jipsen

Department of Mathematics and Computer Science, Chapman University, USA.
jipsen@chapman.edu

Foundations of Relations and Kleene Algebra

Peter Jipsen

Chapman University

August 24, 2006

Introduction

- Aim: cover the basics about relations and Kleene algebras within the framework of universal algebra
- This is a *tutorial*
- Slides give precise definitions, lots of statements
- Decide which statements are true (can be improved) which are *false* (and perhaps how they can be fixed).
- 90 minutes is fairly short, may have to continue on your own

Prerequisites

- Knowledge of sets, union, intersection, complementation
- Some basic first-order logic
- Basic discrete math (e.g. function notation)
- These notes take an *algebraic* perspective

Conventions:

- Minimize distinction between concrete and abstract notation
- $x, y, z, x_1, \ldots$ variables (implicitly universally quantified)
- $X, Y, Z, X_1, \ldots$ set variables (implicitly universally quantified)
- $f, g, h, f_1, \ldots$ function variables
- $a, b, c, a_1, \ldots$ constants
- $i, j, k, i_1, \ldots$ integer variables, usually nonnegative
- $m, n, n_1, \ldots$ nonnegative integer constants

Algebraic properties of set operation

Let U be a set, and $\mathcal{P}(U) = \{X : X \subseteq U\}$ the *powerset* of U

$\mathcal{P}(U)$ is an algebra with operations union $\cup$, intersection $\cap$, complementation $X^- = U \setminus X$

Satisfies many identities: e.g. $X \cup Y = Y \cup X$ for all $X, Y \in \mathcal{P}(U)$

How can we describe the set of all identities that hold?

Can we decide if a particular identity holds in all powerset algebras?

These are questions about the equational theory of these algebras

We will consider similar questions about several other types of algebras, in particular relation algebras and Kleene algebras

Binary relations

An *ordered pair*, written (u, v), has the defining property

$$(u, v) = (x, y) \text{ iff } u = x \text{ and } v = y$$

The *direct product* of sets U, V is

$$U \times V = \{(u, v) : u \in U, v \in V\}$$

A *binary relation* R from U to V is a subset of $U \times V$

Write uRv if $(u, v) \in R$, otherwise write $u \not{R} v$

Define $uR = \{v : uRv\}$ and $Rv = \{u : uRv\}$

Operations on binary relations

Composition of relations: $R;S = \{(u, v) : uR \cap Rv \neq \emptyset\}$

$$= \{(u, v) : \exists x \; uRx \text{ and } xSv\}$$

Converse of R is $R^{\smile} = \{(v, u) : (u, v) \in R\}$

Identity relation $I_U = \{(u, u) : u \in U\}$

A binary relation *on* a set U is a subset of $U \times U$

Define $R^0 = I_U$ and $R^{n+1} = R; R^n$ for $n \geq 0$

Transitive closure of R is $R^+ = \bigcup_{n \geq 1} R^n$

Reflexive transitive closure of R is $R^* = R^+ \cup I_U = \bigcup_{n \geq 0} R^n$

Properties of binary relations

Let R be a binary relation on U

R is *reflexive* if xRx for all $x \in U$

R is *irreflexive* if $x \not{R} x$ for all $x \in U$

R is *symmetric* if xRy implies yRx (implicitly quantified)

R is *antisymmetric* if xRy and yRx implies $x = y$

R is *transitive* if xRy and yRz implies xRz

R is *univalent* if xRy and xRz implies $y = z$

R is *total* if $xR \neq \emptyset$ for all $x \in U$ (otherwise *partial*)

Properties in relational form

Prove (and extend) or disprove (and fix)

R is reflexive iff $I_U \subseteq R$

R is irreflexive iff $I_U \not\subseteq R$

R is symmetric iff $R \subseteq R^{\smile}$ iff $R = R^{\smile}$

R is antisymmetric iff $R \cap R^{\smile} = I_U$

R is transitive iff $R;R = R$ iff $R = R^+$

R is functional iff $R;R^{\smile} \subseteq I_U$

R is total iff $I_U \subseteq R;R^{\smile}$

Binary operations and properties

A *binary operation* $+$ on U is a function from $U \times U$ to U

Write $+(x,y)$ as $x+y$

$+$ is *idempotent* if $x+x=x$ (all implicitly quantified)

$+$ is *commutative* if $x+y=y+x$

$+$ is *associative* if $(x+y)+z=x+(y+z)$

$+$ is *conservative* if $x+y=x$ or $x+y=y$

$+$ is *left cancellative* if $z+x=z+y$ implies $x=y$

$+$ is *right cancellative* if $x+z=y+z$ implies $x=y$

Connection with relations

Define R_+ on U by xR_+y iff $x+z=y$ for some $z \in U$

Prove (and extend) or disprove (and fix)

If $+$ is idempotent then R_+ is reflexive.

If $+$ is commutative then R_+ is antisymmetric.

If $+$ is associative then R_+ is transitive.

A *semigroup* is a set with an associative binary operation

A *band* is a semigroup $(U,+)$ such that $+$ is idempotent

A *quasi-ordered set (qoset)* is a set with a reflexive transitive relation

$\Rightarrow$ If $(U,+)$ is a band then (U,R_+) is a qoset

More specific connection with relations

Define $\le_+$ on U by $x \le_+ y$ iff $x+y=y$

Prove (and extend) or disprove (and fix)

$+$ is idempotent iff $\le_+$ is reflexive.

$+$ is commutative iff $\le_+$ is antisymmetric.

$+$ is associative iff $\le_+$ is transitive.

A *semilattice* is a band $(U,+)$ such that $+$ is commutative

A *partially ordered set* is a qoset (U,R) such that R is antisymmetric

$\Rightarrow$ If $(U,+)$ is a semilattice then $(U,\le_+)$ is a partially ordered set

A partially ordered set is called a *poset* for short

A *strict partial order* is an irreflexive transitive relation

Prove (and extend) or disprove (and fix)

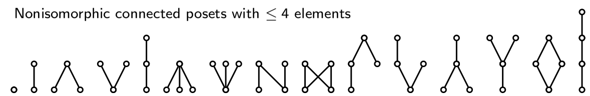
If $<$ is a strict partial order on U, then $(U, < \cup I_U)$ is a poset.
If $(U,\le)$ is a poset, then $< = \le \setminus I_U$ is a strict partial order.

For $a,b \in U$ we say that a is *covered* by b (written $a \prec b$) if $a<b$ and there is no x such that $a<x<b$

To visualize a finite poset we can draw a *Hasse diagram*:

a is connected with an upward sloping line to b if $a \prec b$

Nonisomorphic connected posets with ≤ 4 elements



Equivalence relations

An *equivalence relation* is a reflexive symmetric transitive relation

Prove (and extend) or disprove (and fix)

R is an equivalence relation on U iff $I_U \subseteq R = R^{\smile} ; R$

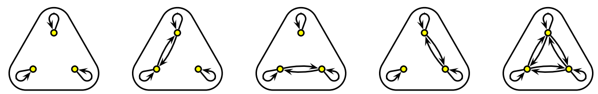
Let R be an equivalence relation on a set U, and $u \in U$

Then $uR = \{x : uRx\}$ is called an *equivalence class* of R

Usually written $[u]_R$ or simply $[u]$; u is called a *representative* of $[u]$

The *set of all equivalence classes* of R is $U/R = \{[u] : u \in U\}$

Equivalence relations on a 3-element set



Partitions

A *partition* of U is a subset P of $\mathcal{P}(U)$ such that

$$\bigcup P = U, \ \emptyset \notin P, \ \text{ and } X = Y \text{ or } X \cap Y = \emptyset \text{ for all } X, Y \in P$$

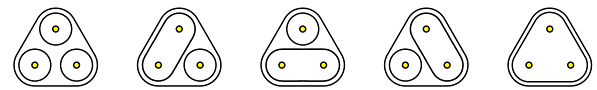
(where $\bigcup P = \{x : x \in X \text{ for some } X \in P\}$)

For a partition P define a relation by $x \equiv_P y$ iff $x, y \in X$ for some $X \in P$

Prove (and extend) or disprove (and fix)

The map $f(R) = U/R$ is a bijection from the set of equivalence relations on U to the set of partitions of U, with $f^{-1}(P)$ given by $\equiv_P$.

Partitions of a 3-element set



The poset induced by a quasi-order

For a qoset $(U, \sqsubseteq)$, define a relation on U by $x \equiv y$ iff $x \sqsubseteq y$ and $y \sqsubseteq x$

Now define $\leq$ on $U/\equiv$ by $[x] \leq [y]$ iff $x \sqsubseteq y$

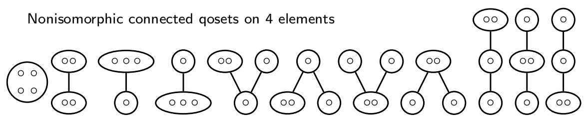
$\leq$ is said to be *well defined* if $[x'] = [x] \leq [y] = [y']$ implies $[x'] \leq [y']$

Prove (and extend) or disprove (and fix)

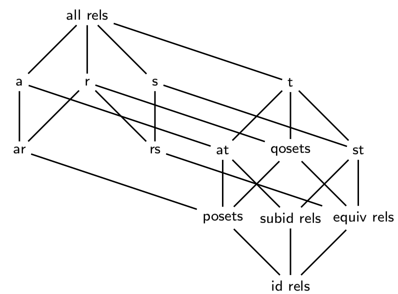
The relation $\leq$ is well defined and $(U/\equiv, \leq)$ is a poset.

Factoring mathematical structures by appropriate equivalence relations is a powerful way of understanding and creating new structures.

Nonisomorphic connected qosets on 4 elements



Some classes of binary relations



a = antisymmetric
r = reflexive
s = symmetric
t = transitive
a and s ⇒ t

Tuples and direct products

We have seen several examples of algebras and relational structures:

$(U, +)$ an algebra with one binary operation, e.g. $(\mathbb{N}, +)$, $(\mathcal{P}(U), \cup)$

(U, R) a relational structure with a binary relation, e.g. $(\mathbb{N}, \leq)$, $(\mathcal{P}(U), \subseteq)$

Applications usually involve several n-ary operations and relations

For a set I, an *I-tuple* $(u_i)_{i\in I}$ is a function mapping $i \in I$ to u_i.

A *tuple over $(U_i)_{i\in I}$* is an I-tuple $(u_i)_{i\in I}$ such that $u_i \in U_i$ for all $i \in I$

The *direct product* $\prod_{i\in I} U_i$ is the set of all tuples over $(U_i)_{i\in I}$

In particular, $\prod_{i\in I} U$ is the set U^I of all functions from I to U

If $I = \{1, \ldots, n\}$ then we write $U^I = U^n$ and $\prod_{i\in I} U_i = U_1 \times \cdots \times U_n$

Note: $U^0 = U^\emptyset = \{()\}$ has one element, namely the empty function $() = \emptyset$

Algebras and relational structures

A (*unsorted first-order*) *structure* is a tuple $\mathbf{U} = (U, (f^{\mathbf{U}})_{f\in\mathcal{F}_\tau}, (R^{\mathbf{U}})_{R\in\mathcal{R}_\tau})$

- U is the *underlying set*
- $\mathcal{F}_\tau$ is a set of *operation symbols* and
- $\mathcal{R}_\tau$ is a set of *relation symbols* (disjoint from $\mathcal{F}_\tau$)

The *type* $\tau : \mathcal{F}_\tau \cup \mathcal{R}_\tau \to \{0, 1, 2, \ldots\}$ gives the *arity* of each symbol

$f^{\mathbf{U}} : U^{\tau(f)} \to U$ and $R^{\mathbf{U}} \subseteq U^{\tau(R)}$ are the *interpretation* of symbol f and R

0-ary operation symbols are called *constant symbols*

$\mathbf{U}$ is a (universal) *algebra* if $\mathcal{R}_\tau = \emptyset$; use $\mathbf{A}, \mathbf{B}, \mathbf{C}$ for algebras

Convention: the string of symbols $f(x_1, \ldots, x_n)$ implies that f has arity n

The superscript $^{\mathbf{U}}$ is often omitted

Monoids and involution

Recall that $(A, \cdot)$ a semigroup if $\cdot$ is an associative operation

A *monoid* is a semigroup with an *identity element*

i.e. of the form $(A, \cdot, 1)$ such that $x \cdot 1 = x = 1 \cdot x$

An *involutive semigroup* is a semigroup with an *involution*

i.e. of the form $(A, \cdot, ^\smile)$ such that $^\smile$ has *period two*: $x^{\smile\smile} = x$, and $^\smile$ *antidistributes over* $\cdot$: $(x \cdot y)^\smile = y^\smile \cdot x^\smile$

Prove (and extend) or disprove (and fix)

If an involutive semigroup satisfies $x \cdot 1 = x$ for some element 1 and all x then it satisfies $1^\smile = 1$ and $1 \cdot x = x$

An *involutive monoid* is a monoid with an *involution*

A *group* is an involutive monoid such that $x \cdot x^\smile = 1$

Join-semilattices

A *semilattice* is a commutative idempotent semigroup

$(A, +, \leq)$ is a *join-semilattice* if
$(A, +)$ is a semilattice and $x \leq y \Leftrightarrow x + y = y$

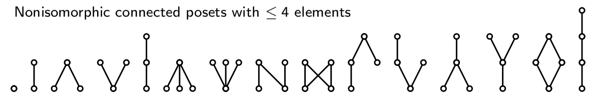
Prove (and extend) or disprove (and fix)

$(A, +, \leq)$ is a join-semilattice
iff $(A, \leq)$ is a poset and $x + y = z \Leftrightarrow \forall w(x \leq w$ and $y \leq w \Leftrightarrow z \leq w)$
iff $(A, \leq)$ is a poset and $x + y \leq z \Leftrightarrow x \leq z$ and $y \leq z$

$\Rightarrow$ any two elements x, y have a *least upper bound* $x + y$

Which of the following are join-semilattices?

Nonisomorphic connected posets with ≤ 4 elements



Lattices and duals

A *meet-semilattice* $(A, \cdot, \leq)$ is a semilattice with $x \leq y \Leftrightarrow x \cdot y = x$

$(A, +, \cdot)$ is a *lattice* if $+, \cdot$ are associative, commutative operations that satisfy the absorbtion laws: $x + (y \cdot x) = x = (x + y) \cdot x$

Prove (and extend) or disprove (and fix)

$(A, +, \cdot)$ is a lattice iff $(A, +, \leq)$ is a join-semilattice and $(A, \cdot, \leq)$ is a meet-semilattice where $x \leq y \Leftrightarrow x + y = y$.

Define $x \geq y \Leftrightarrow y \leq x$. The *dual* $(A, +, \leq)^d = (A, +, \geq)$
$(A, \cdot, \leq)^d = (A, \cdot, \geq)$ and $(A, +, \cdot)^d = (A, \cdot, +)$

Prove (and extend) or disprove (and fix)

The dual of a join-semilattice is a meet-semilattice and vice versa.
The dual of a lattice is again a lattice.

Distributivity and bounds

A lattice is *distributive* if it satisfies $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

Prove (and extend) or disprove (and fix)

A lattice is distributive iff $x + (y \cdot z) = (x + y) \cdot (x + z)$ iff $(x + y) \cdot (x + z) \cdot (y + z) = (x \cdot y) + (x \cdot z) + (y \cdot z)$

$\Rightarrow$ a lattice is distributive iff its dual is distributive

A *semilattice with identity* is a commutative idempotent monoid

$(A, +, 0, \cdot, \top)$ is a *bounded lattice* if
$(A, +, \cdot)$ is a lattice and $(A, +, 0)$, $(A, \cdot, \top)$ are semilattices with identity

Prove (and extend) or disprove (and fix)

Suppose $(A, +, \cdot)$ is a lattice. Then $(A, +, 0, \cdot, \top)$ is a bounded lattice iff $0 \leq x \leq \top$ iff $x \cdot 0 = 0$ and $x + \top = \top$

Complementation and Boolean algebras

$(A, +, 0, \cdot, \top, ^-)$ is a *lattice with complementation* if $(A, +, 0, \cdot, \top)$ is a bounded lattice such that $x + x^- = \top$ and $x \cdot x^- = 0$

Prove (and extend) or disprove (and fix)

Lattices with complementation satisfy $x^{--} = x$ and DeMorgan's laws $(x + y)^- = x^- \cdot y^-$ and $(x \cdot y)^- = x^- + y^-$

A *Boolean algebra* is a distributive lattice with complementation

Prove (and extend) or disprove (and fix)

Boolean algebras satisfy $x^{--} = x$ and DeMorgan's laws $(x + y)^- = x^- \cdot y^-$ and $(x \cdot y)^- = x^- + y^-$

Prove (and extend) or disprove (and fix)

$(A, +, 0, \cdot, \top, ^-)$ is a Boolean algebra iff $+$ is commutative with identity 0, $\cdot$ is commutative with identity 1, $+$ distributes over $\cdot$, $\cdot$ distributes over $+$, $x + x^- = \top$ and $x \cdot x^- = 0$.

Boolean algebras of sets

$\mathcal{P}(U) = (\mathcal{P}(U), \cup, \emptyset, \cap, U, ^-)$ is the *Boolean algebra of all subsets of U*

A *concrete Boolean algebra* is any collection of subsets of a set U that is closed under $\cup$, $\cap$, and $^-$

The *atoms* of a join-semilattice with 0 are the covers of 0

A join-semilattice with 0 is *atomless* if it has no atoms, and

atomic if for every $x \neq 0$ there is an atom $a \leq x$

Prove (and extend) or disprove (and fix)

$\mathcal{P}(U)$ is atomic for every set U

$H = \{(a_1, b_1] \cup \cdots \cup (a_n, b_n] : 0 \leq a_i < b_i \leq 1$ are rationals, $n \in \mathbb{N}\}$ is an atomless concrete Boolean algebra with U the set of positive rationals ≤ 1

Relation algebras

An *(abstract) relation algebra* is of the form $(A, +, 0, \cdot, \top, ^-, ;, 1, ^\smile)$ where

- $(A, +, 0, \cdot, \top, ^-)$ is a Boolean algebra
- $(A, ;, 1)$ is a monoid
- $(x;y) \cdot z = 0 \;\Leftrightarrow\; (x^\smile;z) \cdot y = 0 \;\Leftrightarrow\; (z;y^\smile) \cdot x = 0$

The last line states the Schröder equivalences (or DeMorgan's Thm K)

Prove (and extend) or disprove (and fix)

In a relation algebra $x^{\smile\smile} = x$ *and* $^\smile$ *is self-conjugated, i.e.* $x^\smile \cdot y = 0 \Leftrightarrow x \cdot y^\smile = 0$. *Hence* $(x+y)^\smile = x^\smile + y^\smile$, $x^{-\smile} = x^{\smile-}$, $(x \cdot y)^\smile = x^\smile \cdot y^\smile$, $^\smile$ *is an involution and* $x;(y+z) = x;y + x;z$.

Hint: In a Boolean algebra $u = v$ *iff* $\forall x(u \cdot x = 0 \Leftrightarrow v \cdot x = 0)$

Prove (and extend) or disprove (and fix)

A Boolean algebra expanded with an involutive monoid is a relation algebra iff $x;(y+z) = x;y + x;z$, $(x+y)^\smile = x^\smile + y^\smile$ *and* $x^\smile;(x;y)^- \cdot y = 0$

Concrete relation algebras

$\mathrm{Rel}(U) = (\mathcal{P}(U^2), \cup, \cap, \emptyset, U^2, ^-, ;, I_U, ^\smile)$ the *square relation algebra* on U

A *concrete relation algebra* is of the form $(\mathcal{C}, \cup, \cap, \emptyset, \top, ^-, ;, I_U, ^\smile)$ where $\mathcal{C}$ is a set of binary relations on a set U that is closed under the operations $\cup$, $^-$, $;$, $^\smile$, and contains I_U

Prove (and extend) or disprove (and fix)

Every square relation algebra is concrete.

Every concrete relation algebra is a relation algebra, and the largest relation is an equivalence relation

Relation algebras have applications in program semantics, specification, derivation, databases, set theory, finite variable logic, combinatorics, ...

Idempotent semirings

A *semiring* is an algebra $(A, +, 0, ;, 1)$ such that

- $(A, +, 0)$ is a commutative monoid
- $(A, ;, 1)$ is a monoid
- $x;(y+z) = (x;y) + (x;z)$, $\qquad (x+y);z = (x;z) + (y;z)$
- $x;0 = 0 = 0;x$

A semiring is *idempotent* if $x + x = x$

$\Rightarrow$ an idempotent semiring is a join-semilattice with $x \le y \Leftrightarrow x + y = y$, a bottom element 0, ; distributes over + and 0 is a zero for ;

Prove (and extend) or disprove (and fix)

In an idempotent semiring $x \le y$ *implies* $x;z \le y;z$ *and* $z;x \le z;y$

For any monoid $\mathbf{M} = (M, \cdot, 1)$, the *powerset idempotent semiring* is $\mathcal{P}(\mathbf{M}) = (\mathcal{P}(M), \cup, \emptyset, ;, \{1\})$ where $X;Y = \{x \cdot y : x \in X,\ y \in Y\}$

Kleene algebras

A *Kleene algebra* is of the form $(A, +, 0, ;, 1, ^*)$ where

- $(A, +, 0, ;, 1)$ is an idempotent semiring
- $1 + x + x^*;x^* = x^*$
- $x;y \le y \Rightarrow x^*;y \le y \qquad$ (where $x \le y \Leftrightarrow x + y = y$)
- $y;x \le y \Rightarrow y;x^* \le y$

Prove (and extend) or disprove (and fix)

Let $\mathbf{M} = (M, \cdot, 1)$ *be a monoid. Then* $\mathcal{P}(\mathbf{M})$ *can be expanded to a Kleene algebra if we define* $X^* = \bigcup_{n \ge 0} X^n$ *where* $X^0 = \{1\}$ *and* $X^{n+1} = X^n;X$

Prove (and extend) or disprove (and fix)

For any set U, $\mathrm{KRel}(U) = (\mathcal{P}(U^2), \cup, , ;, I_U, ^*)$ *is a Kleene algebra*

Kleene algebras continued

Traditionally we write $x;y$ simply as xy

A Kleene expression has an *opposite* given by reversing the expression.

The opposite axioms of Kleene algebras again define Kleene algebras, so any proof of a result can be converted to a proof of the opposite result

Prove (and extend) or disprove (and fix)

In a Kleene algebra $x^n \le x^*$ *for all* $n \ge 0$ (*where* $x^0 = 1$, $x^{n+1} = x^n x$)
$x \le y \Rightarrow x^* \le y^*$
$xx^* = x^*x$ $\quad x^{**} = x^*$ *and* $x^* = 1 + x^+$ *where* $x^+ = xx^*$
$xy + z \le y \Rightarrow x^*z \le y$ (*and its opposite*)
$xy = yz \Rightarrow x^*y = yz^*$
$(xy)^*x = x(yx)^*$ *and* $(x+y)^* = x^*(yx^*)^*$

Kleene algebras have applications in automata theory, parsing, pattern matching, semantic and logic of programs, analysis of algorithms,...

Kleene algebras with tests

Kleene algebras model concatenation, nondeterministic choice and iteration, but to model programs need guarded choice and guarded iteration

A *Kleene algebra with tests* (KAT) is of the form $(A, +, 0, ;, 1, ^*, ^-, B)$ where $(A, +, 0, ;, 1, ^*)$ is a Kleene algebra, B is a unary relation ($\subseteq A$) and $x, y \in B \Rightarrow x + y,\ x;y,\ x^-,\ 0, 1 \in B,\ x;x = x,\ x;x^- = 0,\ x + x^- = 1$

Prove (and extend) or disprove (and fix)

In a KAT, $(B, +, 0, ;, 1, ^-)$ *is a Boolean algebra*

[Kozen 1996] defines KATs as two-sorted algebras, but here they are one-sorted structures with $^-$ a partial operation defined only on B

The program construct `if` b `then` p `else` q is expressed by $b;p + b^-;q$

`while` b `do` p is expressed by $(b;p)^*;b^-$

Idempotent semirings with domain and range

Every Kleene algebra is a KAT with $B = \{0, 1\}$

In KRel(U) the tests are a subalgebra of $\mathcal{P}(I_U)$

Can also define *idempotent semirings with tests* (just omit *)

More expressive: add a domain operator [Desharnais Möller Struth 2006]

An *idempotent semiring with predomain* is of the form $(A, +, 0, ;, 1, ^-, \delta)$ where $(A, +, 0, ;, 1, ^-, \delta[A])$ is an idempotent semiring with tests,
$x \le \delta(x);x$ and $\delta(\delta(x);y) \le \delta(x)$

For *idempotent semirings with domain* add $\delta(x;\delta(y)) \le \delta(x;y)$

In Rel(U) the domain operator is definable by $\delta(R) = (R;R^\smile) \cap I_U$

Idempotent semirings with (pre)range operator are opposite

Terms and formulas

UA is a framework for studying and comparing all these algebras

Given a set X, the set of *τ-terms with variables from X* is the smallest set $T = T_\tau(X)$ such that

- $X \subseteq T$ and
- if $t_1, \ldots, t_n \in T$ and $f \in \mathcal{F}_\tau$ then $f(t_1, \ldots, t_n) \in T$.

The *term algebra over X* is $\mathbf{T}_\tau(X) = \mathbf{T} = (T_\tau(X), (f^{\mathbf{T}})_{f \in \mathcal{F}_\tau})$ with

$$f^{\mathbf{T}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \quad \text{for } t_1, \ldots, t_n \in T_\tau(X)$$

A *τ-equation* is a pair of τ-terms (s, t), usually written $s = t$

A *quasiequation* is an implication ($s_1 = t_1$ and ... and $s_n = t_n \Rightarrow s_0 = t_0$)

Models and theories

An *atomic formula* is a τ-equation or $R(x_1,\ldots,x_n)$ for $R \in \mathcal{R}_\tau$

A *τ-formula* $\phi ::=$ atomic frm.$|\phi$and$\phi|\phi$or$\phi|\neg\phi|\phi \Rightarrow \phi|\phi \Leftrightarrow \phi|\forall x\phi|\exists x\phi$

Write $\mathbf{U} \models \phi$ if τ-formula ϕ holds in τ-structure $\mathbf{U}$ (standard defn)

Throughout $\mathcal{K}$ is a class of τ-structures, F a set of τ-formulas

Write $\mathcal{K} \models F$ if $\mathbf{U} \models \phi$ for all $\mathbf{U} \in \mathcal{K}$ and $\phi \in F$

$\mathrm{Mod}(F) = \{\mathbf{U} : \mathbf{U} \models F\}$ = class of all *models* of F

$\mathrm{Th}(\mathcal{K}) = \{\phi : \mathcal{K} \models \phi\}$ = *first order theory* of $\mathcal{K}$

$\mathrm{Th}_e(\mathcal{K}) = \mathrm{Th}(\mathcal{K}) \cap \{\tau\text{-equations}\}$ = *equational theory* of $\mathcal{K}$

$\mathrm{Th}_q(\mathcal{K}) = \mathrm{Th}(\mathcal{K}) \cap \{\tau\text{-quasiequations}\}$ = *quasiequational theory* of $\mathcal{K}$

$\mathrm{Th}_q(\mathcal{K})$ is also called the *strict universal Horn theory* of $\mathcal{K}$

Substructures, homomorphisms and products

Let $\mathbf{U}, \mathbf{V}, \mathbf{V}_i$ ($i \in I$) be structures of type τ and let f, R range over $\mathcal{F}_\tau, \mathcal{R}_\tau$

- $\mathbf{U}$ is a *substructure* of $\mathbf{V}$ if $U \subseteq V$, $f^{\mathbf{U}}(u_1,\ldots,u_n) = f^{\mathbf{V}}(u_1,\ldots,u_n)$ and $R^{\mathbf{U}} = R^{\mathbf{V}} \cap \mathbf{U}^n$ for all $u_1,\ldots,u_n \in U$
- $h : \mathbf{U} \to \mathbf{V}$ is a *homomorphism* if h is a function from U to V, $h(f^{\mathbf{U}}(u_1,\ldots,u_n)) = f^{\mathbf{V}}(h(u_1),\ldots,h(u_n))$ and $(u_1,\ldots,u_n) \in R^{\mathbf{U}} \Rightarrow (h(u_1),\ldots,h(u_n)) \in R^{\mathbf{V}}$ for all $u_1,\ldots,u_n \in U$
- $\mathbf{V}$ is a *homomorphic image* of $\mathbf{U}$ if there exists a surjective homomorphism $h : \mathbf{U} \twoheadrightarrow \mathbf{V}$.
- $\mathbf{U}$ is *isomorphic* to $\mathbf{V}$, in symbols $\mathbf{U} \cong \mathbf{V}$, if there exists a bijective homomorphism from $\mathbf{U}$ to $\mathbf{V}$.
- $\mathbf{U} = \prod_{i\in I} \mathbf{V}_i$, the *direct product* of structures $\mathbf{V}_i$, if $U = \prod_{i\in I} V_i$, $(f^{\mathbf{U}}(u_1,\ldots,u_n)_i)_{i\in I} = (f^{\mathbf{V}_i}(u_{1i},\ldots,u_{ni}))_{i\in I}$ and $(u_1,\ldots,u_n) \in R^{\mathbf{U}} \Leftrightarrow \forall i(u_{1i},\ldots,u_{ni}) \in R^{\mathbf{V}_i}$ for all $u_1,\ldots,u_n \in U$

Substructures are closed under all operations; give "local information"

Homomorphisms are structure preserving maps, and their images capture global regularity of the domain structure

Direct products are used to build or decompose bigger structures

A structure with one element is called *trivial*

A structure is *directly decomposable* if it is isomorphic to a direct product of nontrivial structures

A direct product has *projection maps* $\pi_i : \prod_{i\in I} \mathbf{V}_i \twoheadrightarrow \mathbf{V}_i$ where $\pi_i(u) = u_i$

Prove (and extend) or disprove (and fix)

For any direct product the projection maps are homomorphisms

Isomorphisms preserve all logically defined properties (not only first-order)

Varieties and HSP

$\mathrm{H}\mathcal{K}$ is the class of homomorphic images of members of $\mathcal{K}$

$\mathrm{S}\mathcal{K}$ is the class of substructures of members of $\mathcal{K}$

$\mathrm{P}\mathcal{K}$ is the class of direct products of members of $\mathcal{K}$

A *variety* is of the form $\mathrm{Mod}(E)$ for some set E of equations

A *quasivariety* is of the form $\mathrm{Mod}(Q)$ for some set Q of quasiequations

Prove (and extend) or disprove (and fix)

If $\mathcal{K}$ is a quasivariety then $\mathrm{S}\mathcal{K} \subseteq \mathcal{K}$, $\mathrm{P}\mathcal{K} \subseteq \mathcal{K}$ and $\mathrm{H}\mathcal{K} \subseteq \mathcal{K}$

The next characterization marks the beginning of universal algebra

Theorem (Birkhoff 1935)

$\mathcal{K}$ is a variety iff $\mathrm{H}\mathcal{K} = \mathcal{K}$, $\mathrm{S}\mathcal{K} = \mathcal{K}$ and $\mathrm{P}\mathcal{K} = \mathcal{K}$

Varieties generated by classes

$\Lambda_\tau = \{\mathrm{Mod}(E) : E$ is a set of τ-equations$\}$ = set of all τ-varieties

Prove (and extend) or disprove (and fix)

For sets F_i of τ-formulas $\bigcap_{i\in I}\mathrm{Mod}(F_i) = \mathrm{Mod}(\bigcup_{i\in I} F_i)$

Hence Λ_τ is closed under arbitrary intersections

$\bigcap\Lambda_\tau = \mathrm{Mod}(\{x = y\})$ = the class O_τ of trivial τ-structures

The *variety generated by $\mathcal{K}$* is $\mathrm{V}\mathcal{K} = \bigcap\{$all varieties that contain $\mathcal{K}\}$

Prove (and extend) or disprove (and fix)

$\mathrm{SH}\mathcal{K} = \mathrm{HS}\mathcal{K}$, $\mathrm{PH}\mathcal{K} = \mathrm{HP}\mathcal{K}$ *and* $\mathrm{PS}\mathcal{K} = \mathrm{SP}\mathcal{K}$ *for any class* $\mathcal{K}$

Theorem (Tarski 1946)

$\mathrm{V}\mathcal{K} = \mathrm{HSP}\mathcal{K}$ *for any class $\mathcal{K}$ of structures*

Complete lattices

For a subset X of a poset $\mathbf{U}$ write $X \le u$ if $x \le u$ for all $x \in X$ and define $z = \sum X$ if $X \le u \Leftrightarrow z \le u$ (so $\sum X$ is the *least upper bound* of X)

$u \le X$ and the *greatest lower bound* $\prod X$ are defined dually.

Prove (and extend) or disprove (and fix)

If $\sum X$ exists for every subset of a poset then $\prod X = \sum\{u : u \le X\}$

A structure $\mathbf{U}$ with a partial order is *complete* if $\sum X$ exists for all $X \subseteq U$

$\Rightarrow$ every complete join-semilattice is a complete lattice; $x \cdot y = \prod x, y$

A complete lattice has a bottom $0 = \sum\emptyset$ and a top $\top = \prod\emptyset$

Prove (and extend) or disprove (and fix)

$\mathbf{U}$ *with partial order $\le$ is complete iff $\prod X$ exists for all $X \subset U$*

Λ_τ partially ordered by $\subseteq$ is a complete lattice

Congruences and quotient algebras

A *congruence* on an algebra $\mathbf{A}$ is an equivalence relation θ on A that is compatible with the operations of $\mathbf{A}$, i.e. for all $f \in Fn$

$$x_1\theta y_1 \text{ and } \dots \text{ and } x_n\theta y_n \Rightarrow f^{\mathbf{A}}(x_1,\dots,x_n)\theta f^{\mathbf{A}}(y_1,\dots,y_n)$$

$\mathrm{Con}(\mathbf{A})$ is the set of all congruences on $\mathbf{A}$

Prove (and extend) or disprove (and fix)

$\mathrm{Con}(\mathbf{A})$ *is a complete lattice with $\prod = \bigcap$, bottom I_A and top A^2*

For $\theta \in \mathrm{Con}(\mathbf{A})$, the *quotient algebra* is $\mathbf{A}/\theta = (A/\theta, (f^{\mathbf{A}/\theta})_{f\in\mathcal{F}_\tau})$ where

$$f^{\mathbf{A}/\theta}([x_1]_\theta,\dots,[x_n]_\theta) = f^{\mathbf{A}}(x_1,\dots,x_n)$$

Prove (and extend) or disprove (and fix)

The operations $f^{\mathbf{A}/\theta}$ are well defined and $h_\theta : A \to A/\theta$ given by $h_\theta(x) = [x]_\theta$ is a surjective homomorphism from $\mathbf{A}$ onto $\mathbf{A}/\theta$

Images, kernels and isomorphism theorems

For a function $f : A \to B$ the *image* of f is $f[A] = \{f(x) : x \in A\}$

The *kernel* of f is $\ker f = \{(x,y) \in A^2 : f(x) = f(y)\}$ (an equivalence rel)

Prove (and extend) or disprove (and fix)

If $h : \mathbf{A} \to \mathbf{B}$ is a homomorphism then $\ker h \in \mathrm{Con}(A)$

$h[A]$ is the underlying set of a subalgebra $h[\mathbf{A}]$ of $\mathbf{B}$

The first isomorphism theorem: $f : \mathbf{A}/\ker h \twoheadrightarrow h[\mathbf{A}]$ given by $f([x]_\theta) = h(x)$ is a well defined isomorphism

The second isomorphism theorem: For $\theta \in \mathrm{Con}(\mathbf{A})$, the subset $\uparrow\theta = \{\psi : \theta \subseteq \psi\}$ of $\mathrm{Con}(\mathbf{A})$ is isomorphic to $\mathrm{Con}(\mathbf{A}/\theta)$ via the map $\psi \mapsto \psi/\theta$ where $[x]\psi/\theta[y] \Leftrightarrow x\psi y$

In a join-semilattice, u is *join irreducible* if $u = x + y \;\Rightarrow\; u \in \{x, y\}$

u is *join prime* if $u \le x + y \;\Rightarrow\; u \le x$ or $u \le y$

u is *completely join irreducible* if there is a (unique) greatest element $< u$

u is *completely join prime* if $u \le \sum X \;\Rightarrow\; u \le x$ for some $x \in X$

(completely) meet irreducible and *(completely) meet prime* are given dually

Prove (and extend) or disprove (and fix)

In complete lattices, u is completely join irreducible iff $u = \sum X \Rightarrow u \in X$

Distributivity $\Rightarrow$ (completely) join irreducible = (completely) join prime

u is *compact* if $u \le \sum X \Rightarrow u \le x_1 + \cdots + x_n$ for some $x_1, \ldots, x_n \in X$

A complete lattice is *algebraic* if all element are joins of compact elements

Prove (and extend) or disprove (and fix)

Con($\mathbf{A}$) *is an algebraic lattice* *(hint: compact = finitely generated)*

Subdirect products and subdirectly irreducibles

An *embedding* is an injective homomorphism

An embedding $h : \mathbf{A} \hookrightarrow \prod_{i\in I} \mathbf{B}_i$ is *subdirect* if $\pi_i[h[A]] = B_i$ for all $i \in I$

$\mathbf{A}$ is a *subdirect product* of $(\mathbf{B}_i)_{i\in I}$ if there is a subdirect $h : \mathbf{A} \hookrightarrow \prod_{i\in I} \mathbf{B}_i$

Prove (and extend) or disprove (and fix)

Define $h : \mathbf{A} \hookrightarrow \prod_{i\in I} \mathbf{A}/\theta_i$ by $h(a) = ([a]_{\theta_i})_{i\in I}$

Then h is a subdirect embedding iff $\bigcap_{i\in I} \theta_i = I_A$

$\mathbf{A}$ is *subdirectly irreducible* if for any subdirect $h : \mathbf{A} \hookrightarrow \prod_{i\in I} \mathbf{B}_i$ there is an $i \in I$ such that $\pi_i \circ h$ is an isomorphism

Prove (and extend) or disprove (and fix)

$\mathbf{A}$ is subdirectly irreducible iff $I_A \in$ Con($\mathbf{A}$) is completely meet irreducible iff Con($\mathbf{A}$) has a smallest nonbottom element

Meet irreducibles and subdirect representations

Zorn's Lemma states that if every linearly ordered subposet of a poset has an upper bound, then the poset itself has maximal elements

Prove (and extend) or disprove (and fix)

In an algebraic lattice all members are meets of completly meet irreducibles

The next result shows that subdirectly irreducibles are building blocks

Theorem (Birkhoff 1944)

Every algebra is a subdirect product of its subdirectly irreducible images

$\mathcal{K}_{\mathrm{SI}}$ is the *class of subdirectly irreducibles* of $\mathcal{K}$

$\Rightarrow \mathcal{V} = \mathrm{SP}(\mathcal{V}_{\mathrm{SI}})$ for any variety $\mathcal{V}$

Filters and ideals

For a poset $(U, \le)$ the *principal ideal* of $x \in U$ is $\downarrow x = \{y : y \le x\}$

For $X \subseteq U$ define $\downarrow X = \bigcup_{x\in X} \downarrow x$; $\quad X$ is a *downset* if $X = \downarrow X$

X is *up-directed* if $x, y \in X \Rightarrow \exists u \in X(x \le u$ and $y \le u)$

X is an *ideal* if X is an up-directed downset

principal filter $\uparrow x$, $\uparrow X$, *upset*, *down-directed* and *filter* are defined dually

An ideal or filter is *proper* if it is not the whole poset

An *ultrafilter* is a maximal (with respect to inclusion) proper filter

A filter X in a join-semilattice is *prime* if $x + y \in X \Rightarrow x \in X$ or $y \in X$

Prove (and extend) or disprove (and fix)

The set Fil($\mathbf{U}$) *of all filters on a poset U is an algebraic lattice*
In a join-semilattice every maximal filter is prime
In a distributive lattice every proper prime filter is maximal

Ultraproducts

$\mathcal{F}$ is a *filter over a set I* if $\mathcal{F}$ is a filter in $(\mathcal{P}(I), \subseteq)$

$\mathcal{F}$ defines a congruence on $\mathbf{U} = \prod_{i\in I} \mathbf{U}_i$ via $x\theta_{\mathcal{F}}y \;\Leftrightarrow\; \{i \in I{:}x_i = y_i\} \in \mathcal{F}$

$\mathbf{U}/\theta_{\mathcal{F}}$ is called a *reduced product*, denoted by $\prod_{\mathcal{F}} \mathbf{U}_i$

If $\mathcal{F}$ is an ultrafilter then $\mathbf{U}/\theta_{\mathcal{F}}$ is called an *ultraproduct*

$\mathsf{P}_u\mathcal{K}$ is the class of ultraproducts of members of $\mathcal{K}$

$\mathcal{K}$ is *finitely axiomatizable* if $\mathcal{K} = \mathrm{Mod}(\phi)$ for a single formula ϕ

Prove (and extend) or disprove (and fix)

If $\mathcal{K} \models \phi$ then $\mathsf{P}_u\mathcal{K} \models \phi$ for any first order formula ϕ

If $\mathcal{K}$ is finitely axiomatizable then the complement of $\mathcal{K}$ is closed under ultraproducts

If $\mathcal{K}$ is a finite class of finite τ-structures then $\mathsf{P}_u\mathcal{K} = \mathcal{K}$

Congruence distributivity and Jónsson's Theorem

$\mathbf{A}$ is *congruence distributive* (CD) if $\mathrm{Con}(\mathbf{A})$ is a distributive lattice

A class $\mathcal{K}$ of algebras is *CD* if every algebra in $\mathcal{K}$ is CD

Theorem (Jónsson 1967)

If $\mathcal{V} = \mathsf{V}\mathcal{K}$ is congruence distributive then $\mathcal{V}_{\mathrm{SI}} \subseteq \mathsf{HSP}_u\mathcal{K}$

Prove (and extend) or disprove (and fix)

If $\mathcal{K}$ is a finite class of finite algebras and $\mathsf{V}\mathcal{K}$ is CD then $\mathcal{V}_{\mathrm{SI}} \subseteq \mathsf{HS}\mathcal{K}$

If $\mathbf{A}, \mathbf{B} \in \mathcal{V}_{\mathrm{SI}}$ are finite nonisomorphic and $\mathcal{V}$ is CD then $\mathsf{V}\mathbf{A} \neq \mathsf{V}\mathbf{B}$

$\mathcal{V}$ is *finitely generated* if $\mathcal{V} = \mathsf{V}\mathcal{K}$ for some finite class of finite algebras

Prove (and extend) or disprove (and fix)

A finitely generated CD variety has only finitely many subvarieties

Lattices of subvarieties

If $\mathcal{F}_\sigma \subset \mathcal{F}_\tau$ then the $\mathcal{F}_\sigma$-reduct of a τ-algebra $\mathbf{A}$ is $\mathbf{A}' = (A, (f^{\mathbf{A}})_{f\in\mathcal{F}_\sigma})$

Prove (and extend) or disprove (and fix)

If $\mathbf{A}'$ is a reduct of $\mathbf{A}$ then $\mathrm{Con}(\mathbf{A})$ is a sublattice of $\mathrm{Con}(\mathbf{A}')$

The variety of lattices is CD, so any variety of algebras with lattice reducts is CD

For a variety $\mathcal{V}$ the lattice of subvarieties is denoted by $\Lambda_{\mathcal{V}}$

The meet is $\bigcap$ and the join is $\sum_{i\in I} \mathcal{V}_i = \mathsf{V}(\bigcup_{i\in I} \mathcal{V}_i)$

Prove (and extend) or disprove (and fix)

For any variety $\mathcal{V}$, $\Lambda_{\mathcal{V}}$ is an algebraic lattice with compact elements = varieties that are finitely axiomatizable over $\mathcal{V}$

$\mathsf{HSP}_u(\mathcal{K} \cup \mathcal{L}) = \mathsf{HSP}_u\mathcal{K} \cup \mathsf{HSP}_u\mathcal{L}$ *for any classes $\mathcal{K}, \mathcal{L}$*

If $\mathcal{V}$ is CD then $\Lambda_{\mathcal{V}}$ is distributive and the map $\mathcal{V} \mapsto \mathcal{V}_{\mathrm{SI}}$ is a lattice embedding of $\Lambda_{\mathcal{V}}$ into $\mathcal{P}(\mathcal{V}_{\mathrm{SI}})$

Simple algebras and the discriminator

$\mathbf{A}$ is *simple* if $\mathrm{Con}(\mathbf{A}) = \{I_A, A^2\}$ i.e. has as few congruences as possible

Prove (and extend) or disprove (and fix)

Any simple algebra is subdirectly irreducible

$\mathbf{A}$ is a *discriminator algebra* if for some ternary term t
$\mathbf{A} \models x \neq y \Rightarrow t(x, y, z) = x$ and $t(x, x, z) = z$

Prove (and extend) or disprove (and fix)

Any subdirectly irreducible discriminator algebra is simple

$\mathcal{V}$ is a *discriminator variety* if $\mathcal{V}$ is generated by a class of discriminator algebras (for a fixed term t)

Unary discriminator in algebras with Boolean reduct

A *unary discriminator term* is a term d in an algebra $\mathbf{A}$ with a Boolean reduct such that $d(0) = 0$ and $x \neq 0 \Rightarrow d(x) = \top$

Prove (and extend) or disprove (and fix)

An algebra with a Boolean reduct is a discriminator algebra iff it has a unary discriminator term

[*Hint: let* $d(x) = t(0, x, \top)^-$ *and* $t(x, y, z) = x \cdot d(x^- \cdot y + x \cdot y^-) + z \cdot d(x^- \cdot y + x \cdot y^-)^-$]

In a concrete relation algebra the term $d(x) = \top;x;\top$ *is a unary discriminator term*

For a quantifier free formula ϕ we define a term ϕ^t inductively by $(r = s)^t = (r^- + s) \cdot (r + s^-)$, $(\phi \text{ and } \psi)^t = \phi^t \cdot \psi^t$, $(\neg\phi)^t = d((\phi^t)^-)$

Prove (and extend) or disprove (and fix)

In a discriminator algebra with Boolean reduct $\phi \Leftrightarrow (\phi^t = 1)$

Relation algebras are a discriminator variety

Let $\mathbf{A}a = (\downarrow a, +, 0, \cdot, a, ^{-a}, ;_a, 1\cdot a, ^{\smile a})$ be the *relative subalgebra* of relation algebra A with $a \in A$ where $x^{-a} = x^- \cdot a$, $x;_a y = (x;y)\cdot a$, and $x^{\smile a} = x^{\smile}\cdot a$

An element a in a relation algebra is an *ideal element* if $a = \top;a;\top$

Prove (and extend) or disprove (and fix)

$\mathbf{A}a$ *is a relation algebra iff* $a = a^{\smile} = a;a$

For any ideal element a *the map* $h(x) = (x\cdot a, x\cdot a^-)$ *is an isomorphism from* $\mathbf{A}$ *to* $\mathbf{A}a \times \mathbf{A}a^-$

A relation algebra is simple iff it is subdirectly irreducible
iff it is not directly decomposable
iff $0, \top$ *are the only ideal elements*
iff $\top;x;\top$ *is a unary discriminator term*

Representable relation algebras

The class RRA of *representable relation algebras* is $\mathsf{SP}\{\mathrm{Rel}(X){:}X \text{ is a set}\}$

Prove (and extend) or disprove (and fix)

An algebra is in RRA *iff it is embeddable in a concrete relation algebra*

The class $\mathcal{K} = \mathsf{S}\{\mathrm{Rel}(X) : X \text{ is a set}\}$ *is closed under* H, S *and* $\mathsf{P_u}$

[*Hint:* $\mathsf{P_u S} \subseteq \mathsf{SP_u}$ *so if* $\mathbf{A} = \prod_{\mathcal{U}} \mathrm{Rel}(X_i)$ *for some ultrafilter* $\mathcal{U}$ *over* I, *let* $Y = \prod_{\mathcal{U}} X_i$, *define* $h : \mathbf{A} \to \mathrm{Rel}(Y)$ *by* $[x]h([R])[y] \Leftrightarrow \{i \in I : x_i R_i y_i\} \in \mathcal{U}$ *and show* h *is a well defined embedding*]

$\Rightarrow (\mathsf{V}\mathcal{K})_{\mathsf{SI}} \subseteq \mathcal{K}$ *by Jónsson's Theorem*

$\Rightarrow \mathsf{V}\mathcal{K} = \mathsf{SP}\mathcal{K} = \mathrm{RRA}$ *by Birkoff's subdirect representation theorem*

$\Rightarrow$ [Tarski 1955] RRA is a variety

Theorem

[Lyndon 1950] *There exist nonrepresentable relation algebras (i.e.* $\notin$ RRA*)*

[Monk 1969] RRA *is not finitely axiomatizable*

[Jonsson 1991] RRA *cannot be axiomatized with finitely many variables*

Outline of nonfinite axiomatizability: There is a sequence of finite relation algebras A_n with n atoms and the property that A_n is representable iff there exists a projective plane of order n

By a result of [Bruck and Ryser 1949] projective planes do not exist for infinitely many orders

The ultraproduct of the corresponding sequence of nonrepresentable A_n is representable, so the complement of RRA is not closed under ultraproducts

$\Rightarrow$ RRA is not finitely axiomatizable

Checking if a finite relation algebra is representable

Theorem (Lyndon 1950, Maddux 1983)

There is an algorithm that halts if a given finite relation algebra is not representable

Lyndon gives a recursive axiomatization for RRA

Maddux defines a sequence of varieties RA_n such that $\mathrm{RA} = \mathrm{RA}_4 \supset \mathrm{RA}_5 \supset \ldots \mathrm{RRA} = \bigcap_{n\geq 4} \mathrm{RA}_n$ and it is decidable if a finite algebra is in RA_n

Implemented as a GAP program [Jipsen 1993]

Comer's one-point extension method often gives sufficient conditions for representability; also implemented as a GAP program [J 1993]

Theorem (Hirsch Hodkinson 2001)

Representability is undecidable for finite relation algebras

Complex algebras

Let $\mathbf{U} = (U, T, ^\smile, E)$ be a structure with $T \subseteq U^3$, $^\smile : U \to U$, $E \subseteq U$

The *complex algebra* $\mathrm{Cm}(\mathbf{U})$ is $(\mathcal{P}(U), \cup, \emptyset, \cap, U, ^-, ;, ^\smile, 1)$ where
$X;Y = \{z : (x,y,z) \in T$ for some $x \in X, y \in Y\}$,
$X^\smile = \{x^\smile : x \in X\}$, and $1 = E$

Prove (and extend) or disprove (and fix)

*$\mathrm{Cm}(\mathbf{U})$ is a relation algebra iff $x = y \Leftrightarrow \exists z \in E\ (x,z,y) \in T$,
$(x,y,z) \in T \Leftrightarrow (x^\smile, z, y) \in T \Leftrightarrow (z, y^\smile, x) \in T$, and
$(x,y,z) \in T$ and $(z,u,v) \in T \Rightarrow \exists w((x,w,v) \in T$ and $(y,v,w) \in T)$*

An algebra $\mathbf{A} = (A, \circ, ^\smile, e)$ can be viewed as a structure $(A, T, ^\smile, E)$ where $T = \{(x,y,z) : x \circ y = z\}$ and $E = \{e\}$

Prove (and extend) or disprove (and fix)

$\mathrm{Cm}(\mathbf{A})$ is a relation algebra iff $\mathbf{A}$ is a group

Atom structures

$J(\mathbf{A})$ denotes the set of completely join irreducible elements of $\mathbf{A}$

Prove (and extend) or disprove (and fix)

In a Boolean algebra $J(\mathbf{A})$ is the set of atoms of $\mathbf{A}$

Every atomic BA is embeddable in $\mathcal{P}(J(\mathbf{A}))$ via $x \mapsto J(\mathbf{A}) \cap {\downarrow} x$

Every complete and atomic Boolean algebra is isomorphic to $\mathcal{P}(J(\mathbf{A}))$

The *atom structure* of an atomic relation algebra $\mathbf{A}$ is $(J(\mathbf{A}), ^\smile, T, E)$ where $T = \{(x,y,z) \in J(\mathbf{A}) : x;y \geq z\}$ and $E = J(\mathbf{A}) \cap {\downarrow} 1$

Prove (and extend) or disprove (and fix)

$\mathbf{U} = (U, ^\smile, T, E)$ is the atom structure of some atomic relation algebra iff $\mathrm{Cm}(\mathbf{U})$ is a relation algebra

If $\mathbf{A}$ is complete and atomic then $\mathrm{Cm}(J(\mathbf{A})) \cong \mathbf{A}$

Integral and finite relation algebras

A relation algebra is *integral* if $x;y = 0 \;\Rightarrow\; x = 0$ or $y = 0$

Prove (and extend) or disprove (and fix)

A relation algebra $\mathbf{A}$ is integral iff 1 is an atom of $\mathbf{A}$ iff $x \neq 0 \Rightarrow x;\top = \top$

Rel(2) has 4 atoms and is the smallest simple nonintegral relation algebra

Nonintegral RAs can often be decomposed into a "semidirect product" of integral algebras, so most work has been done on finite integral RAs

For finite relation algebras one usually works with the atom structure

$\mathrm{Rel}(\emptyset)$ is the one-element RA; generates the variety $\mathcal{O} = \mathrm{Mod}(0 = \top)$

Rel(1) is the two-element RA, with $1 = \top$, $x;y = x \cdot y$, $x^\smile = x$

It generates the variety $\mathcal{A}_1 = \mathrm{Mod}(1 = \top)$ of *Boolean relation algebras*

Varieties of small relation algebras

Define $x^s = x + x^\smile$ and let $\mathbf{A}^s$ have underlying set $A^s = \{x^s : x \in A\}$

A relation algebra $\mathbf{A}$ is *symmetric* if $x = x^\smile$ (iff $\mathbf{A}^s = \mathbf{A}$)

Prove (and extend) or disprove (and fix)

If $\mathbf{A}$ is commutative, then $\mathbf{A}^s$ is a subalgebra of $\mathbf{A}$
There are two RAs with 4 elements: $\mathbf{A}_2 = \mathrm{Cm}(\mathbb{Z}_2)$ and $\mathbf{A}_3 = (\mathrm{Cm}(\mathbb{Z}_3))^s$

The varieties generated by $\mathbf{A}_2$ and $\mathbf{A}_3$ are denoted $\mathcal{A}_2$ and $\mathcal{A}_3$

By Jónsson's Theorem $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are atoms of Λ_{RA}

Theorem (Jónsson)

Every nontrivial variety of relation algebras includes $\mathcal{A}_1$, $\mathcal{A}_2$ or $\mathcal{A}_3$

Group RAs and integral RAs of size 8

A complex algebra of a group is called a *group relation algebra*

GRA is the variety generated by all group relation algebras

Prove (and extend) or disprove (and fix)

If $\mathbf{U}$ is a group then $\mathrm{Cm}(\mathbf{U})$ is embedded in $\mathrm{Rel}(U)$ via Cayley's representation, given by $h(X) = \{(u, u \circ x) : u \in U, x \in X\}$

$\Rightarrow$ GRA is a subvariety of RRA

For an algebra $\mathbf{A}$ and $x \in A$, $\mathrm{Sg}^{\mathbf{A}}(x)$ is the subalgebra generated by x

There are 10 integral relation algebras with 8 elements, all 1-generated subalgebras of group relation algebras, hence representable

$\mathbf{B}_1 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_4}\{2\}$	$\mathbf{B}_5 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_5}\{1,4\}$	$\mathbf{C}_1 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_7}\{1,2,4\}$
$\mathbf{B}_2 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_6}\{2,4\}$	$\mathbf{B}_6 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_8}\{1,4,7\}$	$\mathbf{C}_2 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Q}}\{r : r > 0\}$
$\mathbf{B}_3 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_6}\{3\}$	$\mathbf{B}_7 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_{12}}\{3,4,6,8,9\}$	$\mathbf{C}_3 = \mathrm{Cm}(\mathbb{Z}_3)$
$\mathbf{B}_4 = \mathrm{Sg}^{\mathrm{Cm}\mathbb{Z}_9}\{3,6\}$		

Integral relation algebras with 4 atoms

The 8-element integral RAs all have $\mathbf{A}_3$ as the only proper subalgebra

$\Rightarrow$ they generate join-irreducible varieties above $\mathcal{A}_3$

$\mathbf{B}_1, \ldots, \mathbf{B}_7$ are symmetric, $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$

[Comer] There are 102 integral 16-element RAs, not all representable

(65 are symmetric, and 37 are not)

[Jipsen Hertzel Kramer Maddux] 31 nonrepresentable (20 are symmetric)

Problem

What is the smallest representable RA that is not in GRA?
Is there one with 16 elements?

There are 34 candidates at `www.chapman.edu/~jipsen/gap/ramaddux.html` that are representable but not known to be group representable

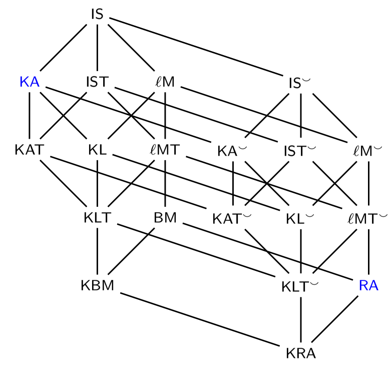
Summary of basic classes of structures

Qoset = *quasiordered sets* = sets with a reflexive and transitive relation
Poset = *partially ordered sets* = antisymmetric quosets
Equiv = *equivalence relations* = symmetric quosets
Sgrp = *semigroups* = associative groupoids
Bnd = *bands* = idempotent ($x + x = x$) semigroups
Slat = *semilattices* = commutative bands
JSlat = *join-semilattices* = semilattices with $x \le y \Leftrightarrow x + y = y$
Lat = *lattices* = two semilattices with absorption laws
Mon = *monoids* = semigroups with identity $x \cdot 1 = x = 1 \cdot x$
$\mathrm{Mon}^\smile$ = *involutive monoids* = monoids with $x^{\smile\smile} = x$, $(x{\cdot}y)^\smile = y^\smile{\cdot}x^\smile$
Grp = *groups* = involutive monoids with $x^\smile \cdot x = 1$
JSLat_0 = *join-semilattices with identity* $x + 0 = x$
$\mathrm{Lat}_{0\top}$ = *bounded lattices* = lattices with $x + 0 = x$ and $x{\cdot}\top = \top$
Lat^- = *complemented lattices* = $\mathrm{Lat}_{0\top}$ with $x + x^- = \top$ and $x{\cdot}x^- = 0$
DLat = *distributive lattices* = lattices with $x{\cdot}(y + z) = x{\cdot}y + x{\cdot}z$
BA = *Boolean algebras* = complemented distributive lattices

Some prominent subclasses of semirings

Srng = *semirings* = monoids distributing over commutative monoids and 0
IS = *(additively) idempotent semirings* = semirings with $x + x = x$
ℓM = *lattice-ordered monoids* = idempotent semirings with meet
RL = *residuated lattices* = ℓ-monoids with residuals
KA = *Kleene algebra* = idempotent semiring with *, unfold and induction
KA* = *$*$-continuous Kleene algebra* = KA with ...
KAT = *Kleene algebras with tests* = KA with Boolean subalgebra ≤ 1
KAD = *Kleene algebras with domain*
KL = *Kleene lattices* = Kleene algebras with meet
BM = *Boolean monoids* = distributive ℓ-monoids with complements
KBM = *Kleene Boolean monoids* = Boolean monids with Kleene-$*$
RA = *relation algebras* = Boolean monoids with involution and residuals
KRA = *Kleene relation algebras* = relation algebras with Kleene-$*$
RRA = *representable relation algebras* = concrete relation algebras
RKRA = *representable Kleene relation algebras* = RRA with Kleene-$*$

Subclasses from combinations of *, tests, meet, $^-$, $^\smile$



A = Algebra
B = Boolean
I = Idempotent
K = Kleene
L = Lattice
ℓ = lattice-ordered
M = Monoid
R = Relation
S = Semiring
T = with tests
$^\smile$ = with converse

Many, but not all, of these classes are varieties

Recall that quasivarietes are classes defined by implications of equations

Most notably, Kleene algebras and some of its subclasses are quasivarieties

In general, implications are not preserved by homomorphic images

To see that KA is not a variety, find an algebra in H(KA) $\setminus$ KA

Prove (and extend) or disprove (and fix)

Let $\mathbf{A}$ *be the powerset Kleene algebra of* $(\mathbb{N}, +, 0)$ *and let* θ *be the equivalence relation with blocks* $\{\emptyset\}$, $\{\{0\}\}$, $\{$*all finite sets* $\neq \{0\}, \emptyset\}$ *and* $\{$*all infinite subsets*$\}$*. Then* θ *is a congruence, but* $\mathbf{A}/\theta$ *is not a Kleene algebra.*

Theorem (Mal'cev)

A class $\mathcal{K}$ *is a quasivariety iff it is closed under* S, P *and* P_u
The smallest quasivariety containing $\mathcal{K}$ *is* $\mathsf{Q}\mathcal{K} = \mathsf{SPP}_u\mathcal{K}$

Free algebras

Let $\mathcal{K}$ be a class and let $\mathbf{F}$ be an algebra that is *generated* by a set $X \subseteq F$ (i.e. $\mathbf{F}$ has no proper subalgebra that contains X)

$\mathbf{F}$ is *$\mathcal{K}$-freely generated* by X if any $f : X \to \mathbf{A} \in \mathcal{K}$ extends to a homomorphism $\hat{f} : \mathbf{F} \to \mathbf{A}$

If also $\mathbf{F} \in \mathcal{K}$ then $\mathbf{F}$ is the *$\mathcal{K}$-free algebra on X* and is denoted by $\mathbf{F}_\mathcal{K}(X)$.

Prove (and extend) or disprove (and fix)

If $\mathcal{K}$ *is the class of all* τ*-algebras then the term algebra* $\mathbf{T}_\tau(X)$ *is the* $\mathcal{K}$*-free algebra on* X

If $\mathcal{K}$ *is any class of* τ*-algebras, let* $\theta_\mathcal{K} = \bigcap\{\ker h \mid h : \mathbf{T}_\tau(X) \to \mathbf{A}$ *is a homomorphism,* $\mathbf{A} \in \mathcal{K}\}$*. Then* $\mathbf{F} = \mathbf{T}_\tau(X)$ *is* $\mathcal{K}$*-freely generated and if* $\mathcal{K}$ *is closed under subdirect products, then* $\mathbf{F} \in \mathcal{K}$

$\Rightarrow$ free algebras exist in all (quasi)varieties (since they are S, P closed)

Examples of free algebras

A free algebra on m generators satisfies only those equations with $\leq m$ variables that hold in all members of $\mathcal{K}$

$\mathbf{F}_{\mathrm{Sgrp}}(X) \cong \bigcup_{n\geq 1} X^n \qquad \mathbf{F}_{\mathrm{Mon}}(X) \cong \bigcup_{n\geq 0} X^n \qquad x \mapsto (x)$

These sets of n-tuples are usually denoted by X^+ and X^*

$\mathbf{F}_{\mathrm{Slat}}(X) \cong \mathcal{P}_{\mathrm{fin}}(X) \setminus \{\emptyset\} \qquad \mathbf{F}_{\mathrm{Slat}_0}(X) \cong \mathcal{P}_{\mathrm{fin}}(X) \qquad x \mapsto \{x\}$

$\mathbf{F}_{\mathrm{Srng}}(X) \cong \{\text{finite multisets of } X^*\} \qquad \mathbf{F}_{\mathrm{IS}}(X) \cong \mathcal{P}_{\mathrm{fin}}(X^*)$

Prove (and extend) or disprove (and fix)

If equality between elements of all finitely generated free algebras is decidable, then the equational theory is decidable

$\Rightarrow$ the equational theories of Sgrp, Mon, Slat, Srng, IS are decidable

Free distributive lattices and Boolean algebras

The free algebras for DLat and BA are also easy to describe

$\mathbf{F}_{\mathrm{DLat}}(X) \cong \mathrm{Sg}_{\mathrm{DLat}}^{\mathcal{P}(\mathcal{P}(X))}(h[X])$

$\mathbf{F}_{\mathrm{BA}}(X) \cong \mathrm{Sg}_{\mathrm{BA}}^{\mathcal{P}(\mathcal{P}(X))}(h[X])$

where in both cases $h(x) = \{Y \in \mathcal{P}(X) : x \in Y\}$ and $x \mapsto h(x)$

For finite X, the free BA is actually isomorphic to $\mathcal{P}(\mathcal{P}(X))$

For lattices, the free algebra on > 3 generators is infinite but the equational theory is still decidable [Skolem 1928] (in polynomial time)

Kleene algebras and regular sets

Deciding equations in KA is also possible, but takes a bit more work

Let Σ be a finite set, called an *alphabet*

The *free monoid generated by Σ* is $\mathbf{\Sigma}^* = (\Sigma^*, \cdot, \varepsilon)$

Here ε is the empty sequence (), and $\cdot$ is concatenation

The *Kleene algebra of regular sets* is $\mathcal{R}_\Sigma = \mathrm{Sg}_{\mathrm{KA}}^{\mathcal{P}(\Sigma^*)}(\{\{(x)\} : x \in \Sigma\})$

Theorem (Kozen 1994)

$\mathcal{R}_\Sigma$ is the free Kleene algebra on Σ

In particular, a *regular set* is the image of a KA term

So deciding if $(s = t) \in \mathrm{Th}_e(KA)$ is equivalent to checking if two regular sets are equal

Membership in regular sets can be determined by finite automata

Automata

A *Σ-automaton* is a structure $\mathbf{U} = (U, (a^{\mathbf{U}})_{a\in\Sigma}, S, T)$ such that $a^{\mathbf{U}}$ is a binary relation and S, T are unary relations.

Elements of U, S, T are called *states*, *start states* and *terminal states* respectively

For $w \in \Sigma^*$ define $w^{\mathbf{U}}$ by $\varepsilon^{\mathbf{U}} = I_U$ and $(a \cdot w)^{\mathbf{U}} = a^{\mathbf{U}} ; w^{\mathbf{U}}$

The *language recognized* by $\mathbf{U}$ is $L(\mathbf{U}) = \{w \in \Sigma^* : w^{\mathbf{U}} \cap S \times T \neq \emptyset\}$

Rec_Σ is the set of all languages recognized by some Σ-automaton

Prove (and extend) or disprove (and fix)

$\emptyset$, $\{\varepsilon\}$, $\{a\} \in \mathrm{Rec}_\Sigma$ *for all* $a \in \Sigma$

Regular sets are recognizable

A finite automaton can be viewed as a directed graph with states as nodes and an arrow labelled a from u_i to u_j iff $(u_i, u_j) \in a^{\mathbf{U}}$

Given automata $\mathbf{U}, \mathbf{V}$, define $\mathbf{U} + \mathbf{V}$ to be the disjoint union of $\mathbf{U}, \mathbf{V}$

$\mathbf{U};\mathbf{V} = (U \uplus V, (a^{\mathbf{U}} \uplus a^{\mathbf{V}} \uplus (a^{\mathbf{U}} T^{\mathbf{U}} \times S^{\mathbf{V}}))_{a\in\Sigma}, S', T^{\mathbf{V}})$ where

$$S' = \begin{cases} S^{\mathbf{U}} & \text{if} S^{\mathbf{U}} \cap T^{\mathbf{U}} = \emptyset \\ S^{\mathbf{U}} \cup S^{\mathbf{V}} & \text{otherwise} \end{cases} \text{ and}$$

$a^{\mathbf{U}} T^{\mathbf{U}} = \{u : \exists v (u, v) \in a^{\mathbf{U}}, v \in T^{\mathbf{U}}\}$

$\mathbf{U}^+ = (U, (a^{\mathbf{U}} \uplus (a^{\mathbf{U}} T^{\mathbf{U}} \times S^{\mathbf{U}}))_{a\in\Sigma}, S^{\mathbf{U}}, T^{\mathbf{U}})$

Prove (and extend) or disprove (and fix)

$L(\mathbf{U} + \mathbf{V}) = L(\mathbf{U}) \cup L(\mathbf{V})$, $L(\mathbf{U};\mathbf{V}) = L(\mathbf{U}); L(\mathbf{V})$, and $L(\mathbf{U}^+) = L(\mathbf{U})^+$

$\Rightarrow$ every regular set is recognized by some finite automaton

Matrices in semirings and Kleene algebras

For a semiring $\mathbf{A}$, let $M_n(A) = A^{n\times n}$ be the set of $n \times n$ matrices over $\mathbf{A}$

$M_n(\mathbf{A})$ is again a semiring with usual matrix addition and multiplication

$\mathbf{0}$ is the zero matrix, and I_n is the identity matrix

If A is a Kleene algebra and $M = \left[\begin{array}{c|c} N & P \\ \hline Q & R \end{array}\right] \in M_n(A)$ define

$$M^* = \left[\begin{array}{c|c} (N + PR^*Q)^* & N^*P(R + QN^*P)^* \\ \hline R^*Q(N + PR^*Q)^* & (R + QN^*P)^* \end{array}\right]$$

This is motivated by the diagram:

Prove (and extend) or disprove (and fix)

The definition of M^* is independent of the chosen decomposition

If $\mathbf{A}$ is a Kleene algebra, so is $M_n(\mathbf{A})$

Finite automata as matrices

Given $\mathbf{U} = (U, (a^{\mathbf{U}})_{a\in\Sigma}, S, T)$ with $U = \{u_1, \ldots, u_n\}$ let (s, M, t) be a $0,1$-row n-vector, an $n \times n$ matrix and a $0,1$-column n-vector where

$s_i = 1 \Leftrightarrow u_i \in S$, $M_{ij} = \sum\{a : (u_i, u_j) \in a^{\mathbf{U}}\}$, and $t_i = 1 \Leftrightarrow u_i \in T$

Prove (and extend) or disprove (and fix)

$L(\mathbf{U}) = h(s; M; t)$ where $h : \mathbf{T}_{\mathsf{KA}}(\Sigma) \to \mathcal{R}_\Sigma$ is induced by $h(x) = \{(x)\}$

$\Rightarrow$ every recognizable language is a regular set [Kleene 1956]

But many different automata may correspond to the same regular set

$\mathbf{U}$ is a *deterministic* automaton if each $a^{\mathbf{U}}$ is a function on U and S is a singleton set

Prove (and extend) or disprove (and fix)

Any nondeterministic automaton $\mathbf{U}$ can be converted to a deterministic one $\mathbf{U}'$ with $U' = \mathcal{P}(U)$, $a'(X) = \{v : (u, v) \in a^{\mathbf{U}}$ for some $u \in X\}$, $S' = \{S\}$ and $T' = \{X : X \cap T \neq \emptyset\}$ such that $L(\mathbf{U}') = L(\mathbf{U})$

Minimal automata

A state v is *accessible* if $(u, v) \in w^{\mathbf{U}}$ for some $u \in S$ and $w \in \Sigma^*$

In a deterministic automaton, the accessible states are the subalgebra generated from the start state

Theorem (Myhill, Nerode 1958)

Given a deterministic automaton $\mathbf{U}$ with no inaccessible states, the relation $u\theta v$ iff $\forall w \in \Sigma^*\ w(u) \in T \Leftrightarrow w(v) \in T$ is a congruence on the automaton and $L(\mathbf{U}/\theta) = L(\mathbf{U})$

An automaton is minimal if all states are accessible and the θ congruence is the identity relation

Prove (and extend) or disprove (and fix)

Let $\mathbf{U}, \mathbf{V}$ be minimal automata. Then $L(\mathbf{U}) = L(\mathbf{V})$ iff $\mathbf{U} \cong \mathbf{V}$.

$\Rightarrow$ The equational theory of Kleene algebras is decidable

Try this in JFLAP: An Interactive Formal Languages and Automata

Th_q((idempotent)semirings) is undecidable

Theorem (Post 1947, Markov 1949)

The quasiequational theory of semigroups is undecidable

For a semigroup $\mathbf{A}$, let $\mathbf{A}_1$ be the monoid obtained by adjoining 1

Prove (and extend) or disprove (and fix)

Any semigroup $\mathbf{A}$ *is a subalgebra of the ;-reduct of* $\mathcal{P}(\mathbf{A})$

If $\mathcal{K} = \{$*;-reducts of semirings*$\}$ *then* $S\mathcal{K}$ = *the class of semigroups*

A quasiequation that uses only ; holds in $\mathcal{K}$ *iff it holds in all semigroups*

$\Rightarrow$ the quasiequational theory of (idempotent) semirings is undecidable

Since $\mathcal{P}(\mathbf{A})$ is a reduct of KA, KAT, KAD, BM the same result holds

The equational theory of RA is undecidable

Prove (and extend) or disprove (and fix)

For any semigroup $\mathbf{A}$*, the monoid* $\mathbf{A}_1$ *is embedded in the ;-reduct of* $\mathrm{Rel}(A_1)$ *via the Cayley map* $x \mapsto \{(x, xy) : y \in A_1\}$

If $\mathcal{K} = \{$*;-reducts of simple* RAs$\}$ *then* $S\mathcal{K}$ = *the class of semigroups*

The quasiequational theory of $\mathrm{RA}_{\mathrm{SI}}$, RA *and* RRA *is undecidable*

RA is a discriminator variety, hence any quasiequation (in fact any quantifier free formula) ϕ can be translated into an equation $\phi^t = 1$ which holds in RA iff ϕ holds in $\mathrm{RA}_{\mathrm{SI}}$

$\Rightarrow$ $\mathrm{Th}_e(\mathrm{RA})$ is undecidable

Undecidability is pervasive in Λ_{RA}

Theorem (Andréka Givant Nemeti 1997)

If $\mathcal{K} \subseteq$RA *such that for each* $n \geq 1$ *there is an algebra in* $\mathcal{K}_{\mathrm{SI}}$ *with at least n elements below the identity then* $\mathrm{Th}_e\mathcal{K}$ *is undecidable*

If $\mathcal{K} \subseteq$RA *such that for each* $n \geq 1$ *there is an algebra in* $\mathcal{K}$ *with a subset of at least n pairwise disjoint elements that form a group under ; and* $^\smile$ *then* $\mathrm{Th}_e\mathcal{K}$ *is undecidable*

Prove (and extend) or disprove (and fix)

The varieties of integral RAs, symmetric RAs and group relation algebras are undecidable

Summary of decidability and other properties

	Th_e dec	Th_q dec	Th dec	Var	CD	loc fin
Sgrp, Mon	✓	×	×	✓	×	×
Slat	✓	✓	×	✓	×	✓
Lat	✓	✓	×	✓	✓	×
DLat	✓	✓	×	✓	✓	✓
BA	✓	✓	✓	✓	✓	✓
Grp	✓	×	×	✓	×	×
Srng	✓	×	×	✓	×	×
IS	✓	×	×	✓	×	×
KA, KAT	✓	×	×	×	×	×
KAD		×	×	×	×	×
RsKA		×	×	✓	✓	×
RsL	✓	×	×	✓	✓	×
BM	×	×	×	✓	✓	×
RA	×	×	×	✓	✓	×
RRA	×	×	×	✓	✓	×
KRA	×	×	×	✓	✓	×

Categories

A *category* is a structure $\mathbf{C} = (C, O, \circ, 1, \mathrm{dom}, \mathrm{cod})$ such that

- C is a class of *morphisms*, O is a class of *objects*, $\mathrm{dom}, \mathrm{cod} : C \to O$ give the *domain* and *codomain*, $1 : O \to C$ gives an *identity morphism*, and *composition* $\circ$ is a partial binary operation on C
- $1(X)$ is denoted 1_X, $f : X \to Y$ means $\mathrm{dom}f = X$ and $\mathrm{cod}f = Y$
- $g \circ f$ exists iff $\mathrm{dom}g = \mathrm{cod}f$, in which case $\mathrm{dom}(g \circ f) = \mathrm{dom}f$, $\mathrm{cod}(g \circ f) = \mathrm{cod}g$ and if $\mathrm{dom}g = \mathrm{cod}h$ then $(f \circ g) \circ h = f \circ (g \circ h)$
- $\mathrm{dom}1_X = X = \mathrm{cod}1_X$, $\quad 1_{\mathrm{dom}f} \circ f = f$ and $f \circ 1_{\mathrm{cod}f} = f$
- The class $\mathrm{Hom}(X, Y) = \{f : \mathrm{dom}f = X \text{ and } \mathrm{cod}f = Y\}$ is a set

Set is a category with sets as objects and functions as morphisms

Rel is a category with sets as objects and binary relations as morphisms

Functors

Category theory is well suited for relating areas of mathematics

Functors are structure preserving maps (homomorphisms) of categories

For categories $\mathbf{C}, \mathbf{D}$ a *covariant functor* $\mathbf{F} : \mathbf{C} \to \mathbf{D}$ maps $C \to D$ and $O^{\mathbf{C}} \to O^{\mathbf{D}}$ such that

- $\mathbf{F}(1_X) = 1_{\mathbf{F}X}$ and if $f : X \to Y$ then $\mathbf{F}f : \mathbf{F}X \to \mathbf{F}Y$
- if $f : X \to Y$, $g : Y \to Z$ then $\mathbf{F}(g \circ f) = \mathbf{F}g \circ \mathbf{F}f$

For a *contravariant functor* $\mathbf{F} : \mathbf{C} \to \mathbf{D}$ the definition becomes

- $\mathbf{F}(1_X) = 1_{\mathbf{F}X}$ and if $f : X \to Y$ then $\mathbf{F}f : \mathbf{F}Y \to \mathbf{F}X$
- if $f : X \to Y$, $g : Y \to Z$ then $\mathbf{F}(g \circ f) = \mathbf{F}f \circ \mathbf{F}g$

Prove (and extend) or disprove (and fix)

A category with one object is (equivalent to) a monoid, and covariant functors between such categories are monoid homomorphisms

Heterogeneous relation algebras

The category **Rel** of typed binary relations is usually enriched by adding converse and Boolean operation on the sets $\mathrm{Hom}(X, Y)$

In this setting it is also natural to write composition $S \circ R$ as $R;S$

A *heterogeneous relation algebra* (HRA) is a structure $\mathbf{C} = (C, O, ;, 1, \mathrm{dom}, \mathrm{cod}, ^{\smile}, +, \top, \cdot, 0, ^{-})$ such that

- $(C, O, ;, 1, \mathrm{dom}, \mathrm{cod})$ is a category
- $^{\smile} : \mathrm{Hom}(x, y) \to \mathrm{Hom}(y, x)$ satisfies $r^{\smile\smile} = r$, $1_x^{\smile} = 1_x$, $(r;s)^{\smile} = s^{\smile};r^{\smile}$
- for all objects x, y, $(\mathrm{Hom}(x, y), +, \top, \cdot, 0, ^{-})$ is a Boolean algebra and
- for all $r;s, t \in \mathrm{Hom}(x, y)$, $(r;s)\cdot t = 0 \Leftrightarrow (r^{\smile};t)\cdot s = 0 \Leftrightarrow (t;s^{\smile})\cdot r = 0$

Prove (and extend) or disprove (and fix)

Relation algebras are (equivalent to) HRAs with one object

Other enriched categories

Suitably weakening the axioms of HRAs (see e.g. [Kahl 2004]) gives
ordered categories (with converse)
(join/meet)-semilattice categories
(idempotent) semiring categories
Kleene categories (with tests)
(distributive/division) allegories

Given a semiring $(A, +, \cdot)$, the set $\mathrm{Mat}(A) = \{A^{m\times n} : m, n \geq 1\}$ of all matrices over A is an important example of a semiring category, with matrix multiplication as composition

The categorical approach is helpful in applications since it matches well with typed specification languages

Conclusion

The foundations of relation algebras and Kleene algebras span a substantial part of algebra, logic and computer science

Here we have only been able to mention some of the basics, with an emphasis on concepts from universal algebra

Participants are encouraged to read further in some of the primary sources and excellent expositary works, some of which are listed below

References and further reading I

[Andreka Givant Nemeti 1994] *The lattice of varieties of representable relation algebras*, J. Symbolic Logic

[Andreka Givant Nemeti 1997] *Decision problems for equational theories of relation algebras*, Memoirs AMS

[Berghammer Möller Struth (Eds) 2004] *Relational and Kleene-algebraic Methods in Computer Science*, LNCS 3051, Springer

[Birkhoff 1935] *On the structure of abstract algebras*, Proc. Camb. Phil. Soc.

[Birkhoff 1944] *Subdirect unions in universal algebra*, Bull. AMS

[Brink Kahl Schmidt (Eds) 1997] *Relational Methods in Computer Science*, Springer

[Burris Sankappanavar 1981] *A course in universal algebra*, Springer, online

[Conway 1971] *Regular algebra and finite machines*, Chapman and Hall

[Desharnais Möller Struth 2003] *Kleene algebras with domain*, online

[Desharnais Möller Struth 2004] *Modal Kleene algebras and applications*, JoRMiCS, online

[Hirsch Hodkinson 2001] *Representability is not decidable for finite relation algebras*, Trans. AMS

[Hirsch Hodkinson 2002] *Relation algebras by games*, North-Holland

[Hodkinson Mikulas Venema 2001] *Axiomatizing complex algebras by games*, Algebra Universalis

References and further reading II

[Jipsen 1993] *Discriminator varieties of Boolean algebras with residuated operators*, in "Algebraic Logic", Banach Center Publ., online

[Jipsen 2004] *From semirings to residuated Kleene lattices*, Studia Logica, online

[Jipsen Lukács 1994] *Minimal relation algebras*, Algebra Universalis

[Jipsen Maddux 1997] *Nonrepresentable sequential algebras*, J. IGPL, online

[Jipsen Tsinakis 2002] *A survey of residuated lattices*, in "Ordered algebraic structures", Kluwer, online

[Jónsson 1967] *Algebras whose congruence lattices are distributive*, Math Scand.

[Jónsson 1982] *Varieties of relation algebras*, Algebra Universalis

[Jónsson 1991] *The theory of binary relations*, in "Algebraic Logic", North-Holland

[Jonsson Tarski 1951/2] *Boolean algebras with operators I, II*, Amer. J. Math.

[Kahl 2004] *Refactoring heterogeneous relation algebras around ordered categories and converse*, JoRMiCS, online

[Kozen 1994] *A completeness theorem for Kleene algebras and the algebra of regular events*, Infor. and Comput., online

[Kozen 1994] *On action algebras*, in "Logic and Information Flow", MIT Press, online

[Kozen 1997] *Automata and computability*, Springer

References and further reading III

[Kozen 2003] *Automata on guarded strings and applications*, Matémat. Contemp., online

[Kozen and Smith 1996] *Kleene algebras with test: Completeness and decidability*, in LNCS 1258, Springer, online

[Maddux 1982] *Some varieties containing relation algebras*, Trans. AMS

[Maddux 1983] *A sequent calculus for relation algebras*, Ann. Pure and Appl. Logic

[Maddux 1985] *Finite integral relation algebras*, in LNM 1149, Springer

[Maddux 2006] *Relation algebras*, Elsevier

[Pratt 1990] *Dynamic algebras as a well-behaved fragment of relation algebras*, in LNCS 425, Springer, online

[Pratt 1990] *Action logic and pure induction*, in LNCS 478, Springer, online

[Tarski 1946] *A remark on functionally free algebras*, Ann. Math.

[Tarski 1955] *Contributions to the theory of models III*, Konin. Nederl. Akad. Weten. Proc.

Relational Methods for Program Refinement

John Derrick

Department of Computer Science, University of Sheffield, UK.
jd@dcs.shef.ac.uk

RelMiCS 2006

Relational methods for Program Refinement

John Derrick
University of Sheffield

1

A Tutorial on Refinement in State-based Specification Languages

In this tutorial we aim to provide:

- An introduction to the idea of refinement as a formal development process.
- An insight into how refinement is defined in Z and other specification languages.
- An understanding of how the relational basis for Z leads to the derivation of the Z simulation rules as they are usually presented.
- An understanding of the relationship to various process algebraic refinement relations.

2

Outline:

- Overview of Refinement
- A relational framework
- Data refinement
- Simulations
- Refinement in Z
- Deriving simulations in Z
- Examples
- Concurrent models of refinement
- Unifying relational and concurrent refinement

3

What is Refinement?

- Write a program to input a number, double it and output the result on the screen.
- Draw a Jack playing card.

Specifications define what is *observable* and what is *hidden*.

They are also often loose, i.e., contain non-determinism.

4

Refinement is the process of development:

- the internal representation doesn't matter, all that matters is the observable behaviour.

- if options have been left open, we are free to make a choice i.e., reduce non-determinism.

So refinement is based upon:

> **The Principle of Substitutivity:** it is acceptable to refine one program by another,
>
> \- *provided* it is impossible for a user of the programs to observe that the substitution has taken place.

All formal methods have notions of refinement:

- Process algebras (CSP, LOTOS etc) - trace, failure-divergence, reduction etc.

- Automata;

- State-based languages (Z, VDM, B etc) - data refinement.

Refinement in Z

Data refinement is a methodology that allows state spaces to be altered in a development, and non-determinism to be reduced.

For example, a *digital watch* might have the state space

TimeHM
$hrs : 0..23$
$mins : 0..59$

TimeHMInit
$TimeHM'$

With operations to show and reset the time

ShowTime
$\Xi TimeHM$
$hrs! : 0..23$
$mins! : 0..59$

$hrs = hrs!$
$mins = mins!$

ResetTime
$\Delta TimeHM$
$hrs? : 0..23$
$mins? : 0..59$

$hrs' = hrs?$
$mins' = mins?$

A refinement to the watch might add some more detail to this design:

Change the internal representation, e.g.,

TimeHM
$hrs : 0..23$
$mins : 0..59$

CTimeHM
$time : \mathbb{N} \times \mathbb{N}$

and/or reduce some non-determinism

TimeHMInit
$TimeHM'$

CTimeHMInit
$CTimeHM'$

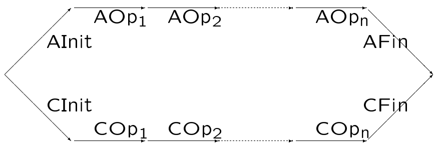
$time' = (0, 0)$

What is the correct specification of *ShowTime* using *CTimeHM*?

Programs in this specification are sequences of operations:

$$\mathit{TimeHMInit} \fatsemi \mathit{ShowTime} \fatsemi \mathit{ShowTime} \fatsemi \mathit{ResetTime} \fatsemi \mathit{ShowTime}$$

In a **refinement** we will need to match equivalent programs, so that an abstract program can be refined by a concrete program.



So a specification $\mathcal{A}$ is refined by $\mathcal{C}$ iff for each program P

$$P(\mathcal{C}) \subseteq P(\mathcal{A})$$

e.g.

$$\mathit{CInit} \fatsemi \mathit{COp}_1 \fatsemi \mathit{COp}_2 \fatsemi \mathit{CFin} \subseteq \mathit{AInit} \fatsemi \mathit{AOp}_1 \fatsemi \mathit{AOp}_2 \fatsemi \mathit{AFin}$$

Thus for our example, we need:

$$\mathit{CTimeHMInit} \fatsemi \mathit{CShowTime} \fatsemi \mathit{CShowTime} \fatsemi \mathit{CResetTime} \subseteq \mathit{TimeHMInit} \fatsemi \mathit{ShowTime} \fatsemi \mathit{ShowTime} \fatsemi \mathit{ResetTime}$$

(Don't worry about the *AFin* and *CFin* bit.)

But we need to verify this *for every possible program*.

Several issues

- What exactly are the observations of an *abstract* data type, and what is their relation to ADT *programs*?
- How do inputs and outputs fit in with this?
- What is the effect of using operations specified by relations which are not necessarily total?
- What is the function of initialisation?
- How can we verify refinements without checking all programs?

Data Refinement and Simulations

Next:

- the standard definition of data refinement for data types whose operations are *total* relations,
- the definitions of upward and downward simulations,
- the statement of their soundness and joint completeness.

To apply this theory to a specification language we look at how operations in a specification are modelled as partial relations.

The application of the simulation rules to specifications with partial operations leads to the simulation rules as they are normally presented in Z.

First, some definitions ...

Definition 1 (Data type)
A data type, (State, Init, $\{Op_i\}_{i \in I}$, Fin), *has operations* $\{Op_i\}$, *indexed by* $i \in I$, *that are* total *relations on the set* State*;* Init *is a relation from* G *to* State*;* Fin *is a relation from* State *to* G. □

A data type is canonical if Init, Op_i and Fin are all *functions*. Two data types are *conformal* if their global data space G and the indexing sets of their operations are equal.

Definition 2 (Complete program)
A complete program *over a data type* D *is an expression of the form* $Init \mathbin{;} P \mathbin{;} Fin$*, where* P*, a relation over* State*, is a* program *over* $\{Op_i\}_{i \in I}$. □

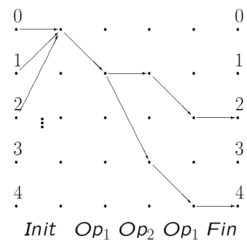
For example, if $p = \langle p_1, \ldots, p_n \rangle$ then
$p_D = Init \mathbin{;} Op_{p_1} \mathbin{;} \ldots \mathbin{;} Op_{p_n} \mathbin{;} Fin$.

Example

Let $G = \mathbb{N}$, $D = (\mathbb{N}, Init, \{Op_1, Op_2\}, Fin)$ where

$$Init = \{x : \mathbb{N} \bullet (x, 0)\}$$
$$Op_1 = \{x : \mathbb{N} \bullet x' = x + 1\}$$
$$Op_2 = \{x, y : \mathbb{N} \mid x' \in \{x, x + 2\}\}$$
$$Fin = id_{\mathbb{N}}$$

The program $[1, 2, 1]_D$ denotes
$Init \mathbin{;} Op_1 \mathbin{;} Op_2 \mathbin{;} Op_1 \mathbin{;} Fin$ which is
$\{x : \mathbb{N};\ y : \{2, 4\} \bullet (x, y)\}$.



Definition 3 (Data refinement)
For data types A *and* C, C refines A *(denoted* $A \sqsubseteq C$*) iff for each finite sequence* p *over* I, $p_C \subseteq p_A$. □

Data refinement is transitive and reflexive, i.e., it is a preorder.

Example

Let $G = \mathbb{N}$, with

$$A = (\mathbb{N}, id_{\mathbb{N}}, \{AOp\}, id_{\mathbb{N}})$$
$$C = (\mathbb{N}, id_{\mathbb{N}}, \{COp\}, id_{\mathbb{N}})$$
$$AOp = \{x, y : \mathbb{N} \mid x < y \bullet (x, y)\}$$
$$COp = \{x : \mathbb{N} \bullet (x, x + 1)\}$$

Show that for any sequence p:

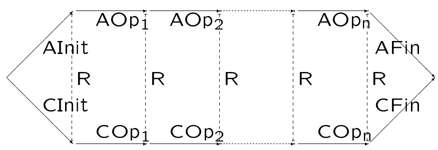
$p_A = \{x, y : \mathbb{N} \mid x + \#p \leq y \bullet (x, y)\}$, whereas
$p_C = \{x : \mathbb{N} \bullet (x, x + \#p)\}$

Thus $p_C \subseteq p_A$, and therefore C refines A.

How to verify a refinement

Simulations are used to verify refinements:

- they allow a step-by-step comparison of operations,
- instead of looking at the effect of the whole program.



R links the abstract and concrete state spaces.

Simulations

How to verify a refinement by considering step by step values.

We need a relation *R* between the two sets of states *AState* and *CState*.

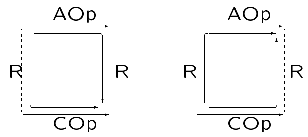
Then we can consider two types of step by step comparisons: downwards simulation and upwards simulation.

Facts: these two simulations are sound and jointly complete.

Every downwards or upwards simulation is a valid refinement.

Every valid refinement can be proved by a combination of downwards and upwards simulations. In fact every refinement needs just one upwards simulation followed by a downwards simulation.

Two ways of making the diagram commute:



Definition 4 (Downward simulation)
Given $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp_i}\}_{i\in I}, \mathsf{AFin})$ *and* $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp_i}\}_{i\in I}, \mathsf{CFin})$. *A downward simulation is a relation* R *from* AState *to* CState *satisfying*

$$\mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \mathsf{R} \tag{1}$$

$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin} \tag{2}$$

$$\forall\, i : I \bullet \mathsf{R} \fatsemi \mathsf{COp_i} \subseteq \mathsf{AOp_i} \fatsemi \mathsf{R} \tag{3}$$

If such a simulation exists, C *is called a downward simulation of* A. □

Downward simulations are also known as *forward* simulations.

Definition 5 (Upward simulation)
For data types A *and* C *as above, an* upward *simulation is a relation* T *from* CState *to* AState *such that*

$$\mathsf{CInit} \fatsemi \mathsf{T} \subseteq \mathsf{AInit} \tag{4}$$

$$\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin} \tag{5}$$

$$\forall\, i : I \bullet \mathsf{COp_i} \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp_i} \tag{6}$$

If such a simulation exists, we also say that C *is an upward simulation of* A. □

Another term for this is *backward* simulation.

Exercise:

Show that if a relation T between CState and AState is total and functional, then T is an upward simulation between A and C if and only if T^{-1} is a downward simulation between A and C.

Answer: Totality of T is encoded relationally as $\mathsf{T} \fatsemi \mathsf{T}^{-1} \supseteq \mathsf{id_{CState}}$, and functionality as $\mathsf{T}^{-1} \fatsemi \mathsf{T} \subseteq \mathsf{id_{AState}}$.

Together these allow the proof of equivalence for the corresponding upward and downward simulation conditions.

E.g.,

$$\begin{array}{l}
\mathsf{COp_i} \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp_i} \\
\qquad \Rightarrow \\
\mathsf{T}^{-1} \fatsemi \mathsf{COp_i} \fatsemi \mathsf{T} \fatsemi \mathsf{T}^{-1} \subseteq \mathsf{T}^{-1} \fatsemi \mathsf{T} \fatsemi \mathsf{AOp_i} \fatsemi \mathsf{T}^{-1} \\
\qquad \Rightarrow \{\ \text{totality on lhs, functionality on rhs}\ \} \\
\mathsf{T}^{-1} \fatsemi \mathsf{COp_i} \subseteq \mathsf{AOp_i} \fatsemi \mathsf{T}^{-1} \\
\qquad \Rightarrow \\
\mathsf{T} \fatsemi \mathsf{T}^{-1} \fatsemi \mathsf{COp_i} \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp_i} \fatsemi \mathsf{T}^{-1} \fatsemi \mathsf{T} \\
\qquad \Rightarrow \{\ \text{totality on lhs, functionality on rhs}\ \} \\
\mathsf{COp_i} \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp_i}
\end{array}$$

(The middle line is the condition for T^{-1} to be a downward simulation, which both implies and is implied by the top/bottom line.)

Theorem 1 (Horizontal composition) *For conformal data types* A *and* C *as above, and appropriately typed relations* R *and* T*,*

(a) *if (1) and (3) hold, then (1) holds for* $\mathsf{AInit} := \mathsf{AInit} \mathbin{;} \mathsf{AOp_i}$ *and* $\mathsf{CInit} := \mathsf{CInit} \mathbin{;} \mathsf{COp_i}$

(b) *if (3) holds, then it also holds for any sequence of two operations, i.e.,* $\forall i,j : I \bullet R \mathbin{;} \mathsf{COp_i} \mathbin{;} \mathsf{COp_j} \subseteq \mathsf{AOp_i} \mathbin{;} \mathsf{AOp_j} \mathbin{;} R$

(c) *if (3) and (2) hold, then (2) holds for* $\mathsf{AFin} := \mathsf{AOp_i} \mathbin{;} \mathsf{AFin}$ *and* $\mathsf{CFin} := \mathsf{COp_i} \mathbin{;} \mathsf{CFin}$

(d) *if (4) and (6) hold, then (4) holds for* $\mathsf{AInit} := \mathsf{AInit} \mathbin{;} \mathsf{AOp_i}$ *and* $\mathsf{CInit} := \mathsf{CInit} \mathbin{;} \mathsf{COp_i}$

(e) *if (6) holds, then it also holds for any sequence of two operations, i.e.,*

$$\forall i,j : I \bullet \mathsf{COp_i} \mathbin{;} \mathsf{COp_j} \mathbin{;} T \subseteq T \mathbin{;} \mathsf{AOp_i} \mathbin{;} \mathsf{AOp_j}$$

(f) *if (6) and (5) hold, then (5) holds for* $\mathsf{AFin} := \mathsf{AOp_i} \mathbin{;} \mathsf{AFin}$ *and* $\mathsf{CFin} := \mathsf{COp_i} \mathbin{;} \mathsf{CFin}$

Answer:

(a)

$$\begin{aligned}
&\mathsf{CInit} \mathbin{;} \mathsf{COp_i}\\
&\quad \subseteq \{\text{by } (1)\}\\
&\mathsf{AInit} \mathbin{;} R \mathbin{;} \mathsf{COp_i}\\
&\quad \subseteq \{\text{by } (3)\}\\
&\mathsf{AInit} \mathbin{;} \mathsf{AOp_i} \mathbin{;} R
\end{aligned}$$

(b)

$$\begin{aligned}
&R \mathbin{;} \mathsf{COp_i} \mathbin{;} \mathsf{COp_j}\\
&\quad \subseteq \{\text{by } (3)\}\\
&\mathsf{AOp_i} \mathbin{;} R \mathbin{;} \mathsf{COp_j}\\
&\quad \subseteq \{\text{by } (3)\}\\
&\mathsf{AOp_i} \mathbin{;} \mathsf{COp_j} \mathbin{;} R
\end{aligned}$$

(c)

$$\begin{aligned}
&R \mathbin{;} \mathsf{COp_i} \mathbin{;} \mathsf{CFin}\\
&\quad \subseteq \{\text{by } (3)\}\\
&\mathsf{AOp_i} \mathbin{;} R \mathbin{;} \mathsf{CFin}\\
&\quad \subseteq \{\text{by } (2)\}\\
&\mathsf{AOp_i} \mathbin{;} \mathsf{AFin}
\end{aligned}$$

Show the following:

Theorem 2 (Soundness of simulations) *If an upward or downward simulation exists between conformal data types* A *and* C*, then* C *is a data refinement of* A*.*

By induction on the (complete) programs, by proving a base case, and using Theorem 1 (a)/(d) as the induction step.

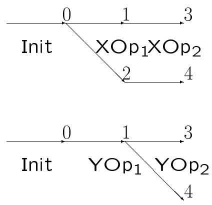
(in)Completeness

However, neither downward simulation nor upward simulation is *complete* on their own.

Example of ADTs which are related by data refinement where no downward simulation exists are as follows:

Let $G = 0..4$, which acts as the global state and one of the local states; the other local state is $YS = \{0,1,3,4\}$. Define
$X = (G, \mathsf{Init}, \{\mathsf{XOp_1}, \mathsf{XOp_2}\}, id_G)$,
$Y = (YS, \mathsf{Init}, \{\mathsf{YOp_1}, \mathsf{YOp_2}, \}, id_{YS})$, where

$\mathsf{Init} = \{x : G \bullet (x, 0)\}$,
$\mathsf{XOp_1} = \{(0,1),(0,2),(1,1),(2,2),(3,3),(4,4)\}$,
$\mathsf{XOp_2} = \{(0,0),(1,3),(2,4),(3,3),(4,4)\}$,
$\mathsf{YOp_1} = \{(0,1),(1,1),(3,3),(4,4)\}$,
$\mathsf{YOp_2} = \{(0,0),(1,3),(1,4),(3,3),(4,4)\}$.



A program here is a finite sequence over the numbers 1 and 2. The finalisation is the identity, so the final state is the only observable outcome of a program.

- Any program which does not include operation 1 (including the empty program) will end in state 0.

- Any program containing operation 1 but not 2 after it will end in state 1 or 2 for X, and in state 1 for Y.

- Any other program (i.e., containing at least operation 1 followed by 2 sometime later) could end in either state 3 or 4 for both X and Y.

As the outcomes for Y are always included in those for X, data refinement holds.

Assume we have a downward simulation R from G to YS.

From the finalisation condition (2) we get $R \subseteq id_{YS}$.

From initialisation and refinement for the first operation we then get that $(1,1) \in R$.

The refinement condition for the second operation then gives us:

$$\begin{array}{l} R \mathbin{⨾} YOp_2 \subseteq XOp_2 \mathbin{⨾} R \\ \qquad \Rightarrow \{(1,1) \in R, \text{transitivity of } \subseteq\} \\ \{(1,3),(1,4)\} \subseteq XOp_2 \mathbin{⨾} R \\ \qquad \equiv \\ \{(1,3),(1,4)\} \subseteq \{(1,3)\} \mathbin{⨾} R \\ \qquad \Rightarrow \{⨾\} \\ (3,4) \in R \end{array}$$

which contradicts that R is contained in the identity relation.

Thus, no such R can exist.

Exercise:

Show that it can be proved that $\{(0,0),(1,1),(1,2),(3,3),(4,4)\}$ does constitute an upward simulation in this case.

Exercise:

Construct a similar example where the abstract and concrete data type are swapped and $\{(1,2),(2,1)\}$ added to both finalisations to show a data refinement where no upward simulation exists.

Theorem 3 *Upward and downward simulation are jointly complete, i.e., any data refinement can be proved by a combination of simulations.*

Partiality

Not all operations are total, the meaning of an operation *Op* specified as a partial relation is:

- *Op* behaves as specified when used within its precondition (domain);
- outside its precondition, anything may happen.

We model this by totalising relations, i.e., adding a distinguished element $\perp$, denoting undefinedness.

Two ways to do this: contract vs behavioural.

In the "*contract*" approach, the domain (precondition) of an operation describes the area within which the operation should be guaranteed to deliver a well-defined result, as specified by the relation.

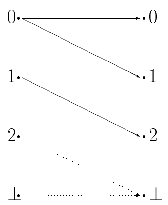
Outside that domain, the operation *may* be applied, but may return any value, even an undefined one (modelling, e.g., non-termination).

In the "*behavioural*" approach, operations may not be applied outside their precondition; doing so anyway leads to an undefined result.

In either case, in both approaches the simulation rules for partial operations are derived from those for total operations:

partial relations on a set S are modelled as total relations on a set $S_\perp$, which is S extended with a distinguished value $\perp$ not in S.

E.g., in the behavioural approach, values outside the domain are linked to $\perp$ only.



Definition 6 (Totalisation)
For a partial relation Op *on* State*, its totalisation is a total relation* $\widehat{\mathsf{Op}}$ *on* $\mathsf{State}_\perp$*, defined in the "contract" approach to ADTs by*

$$\widehat{\mathsf{Op}} == \mathsf{Op} \cup \{\mathsf{x}, \mathsf{y} : \mathsf{State}_\perp \mid \mathsf{x} \notin \mathit{dom}\, \mathsf{Op} \bullet (\mathsf{x}, \mathsf{y})\}$$

or in the behavioural approach to ADTs by

$$\widehat{\mathsf{Op}} == \mathsf{Op} \cup \{\mathsf{x} : \mathsf{State}_\perp \mid \mathsf{x} \notin \mathit{dom}\, \mathsf{Op} \bullet (\mathsf{x}, \perp)\}$$

Totalisations of initialisation and finalisation are defined analogously. □

Definition 7 (Extension)
A relation R *between* AState *and* CState *is extended to a relation* $\overline{\mathsf{R}}$ *between* $\mathsf{AState}_\perp$ *and* $\mathsf{CState}_\perp$*, defined in the contract approach by*

$$\overline{\mathsf{R}} == \mathsf{R} \cup (\{\perp_{\mathsf{AState}}\} \times \mathsf{CState}_\perp)$$

and in the behavioural approach by

$$\overline{\mathsf{R}} == \mathsf{R} \cup \{(\perp_{\mathsf{AState}}, \perp_{\mathsf{CState}})\}$$

□

Extracting the partiality

The simulation rules are defined in terms of totalised relations. We can extract the underlying rules for the original operations.

Thus our goal is to apply the simulation rules to the totalised versions of these data types, and then to remove all occurrences of $\widetilde{\ }$, $\widehat{\ }$ and $\perp$ in the rules.

For initialisation we have:

$$
\begin{array}{l}
\widehat{CInit} \subseteq \widehat{AInit} \fatsemi \widetilde{R} \\
\qquad \equiv \\
CInit \cup \{(\perp_G, \perp_{CState})\} \\
\qquad \subseteq \\
(AInit \cup \{(\perp_G, \perp_{AState})\}) \fatsemi (R \cup \{(\perp_{AState}, \perp_{CState})\}) \\
\qquad \equiv \\
CInit \cup \{(\perp_G, \perp_{CState})\} \subseteq AInit \fatsemi R \cup \{(\perp_G, \perp_{CState})\} \\
\qquad \equiv \\
CInit \subseteq AInit \fatsemi R
\end{array}
$$

We use the following abbreviation:

$$X \ntriangleleft == (\overline{\mathrm{dom}\, X})_\perp$$

For the operations we get the following:

$$
\begin{array}{l}
\widetilde{R} \fatsemi \widehat{COp} \subseteq \widehat{AOp} \fatsemi \widetilde{R} \\
\qquad \equiv \ldots \\
R \fatsemi COp \subseteq AOp \fatsemi R \\
\quad \wedge \\
(R \fatsemi (COp \ntriangleleft \times \{\perp_{CState}\})) \subseteq (AOp \ntriangleleft \times \{\perp_{CState}\})
\end{array}
$$

The second conjunct can be further simplified. As the ranges of both relations are $\{\perp_{CState}\}$, only their domains are relevant. Removing also the complements, we get

$$\mathrm{ran}(\mathrm{dom}\, AOp \lhd R) \subseteq \mathrm{dom}\, COp$$

For finalisation, using analogous steps:

$$
\begin{array}{l}
\widetilde{R} \fatsemi \widehat{CFin} \subseteq \widehat{AFin} \\
\qquad \equiv \\
R \fatsemi CFin \subseteq AFin \\
\quad \wedge \\
\mathrm{ran}(\mathrm{dom}\, AFin \lhd R) \subseteq \mathrm{dom}\, CFin
\end{array}
$$

Definition 8
Let $A = (AState, AInit, \{AOp_i\}_{i \in I}, AFin)$ *and* $C = (CState, CInit, \{COp_i\}_{i \in I}, CFin)$ *be data types where the operations may be partial.*

A downward *simulation is a relation* R *from* AState *to* CState *satisfying, in the contract interpretation*

$$
\begin{array}{l}
CInit \subseteq AInit \fatsemi R \\
R \fatsemi CFin \subseteq AFin \\
\mathit{ran}(\mathit{dom}\, AFin \lhd R) \subseteq \mathit{dom}\, CFin \\
\forall i : I \bullet \mathit{ran}(\mathit{dom}\, AOp_i \lhd R) \subseteq \mathit{dom}\, COp_i \\
\forall i : I \bullet (\mathit{dom}\, AOp_i \lhd R) \fatsemi COp_i \subseteq AOp_i \fatsemi R
\end{array}
$$

The five conditions are commonly referred to as initialisation, finalisation, finalisation applicability, applicability and correctness. In the behavioural interpretation, correctness is strengthened to:

$$\forall i : I \bullet R \fatsemi COp_i \subseteq AOp_i \fatsemi R$$

Upward Simulations

$\widehat{CInit} \fatsemi \widetilde{T} \subseteq \widehat{AInit}$ simplifies to $CInit \fatsemi T \subseteq AInit$.

For operations, we get:

$$
\begin{array}{l}
\widehat{COp} \fatsemi \widetilde{T} \subseteq \widetilde{T} \fatsemi \widehat{AOp} \\
\qquad \equiv \ldots \\
COp \fatsemi T \subseteq T \fatsemi AOp \\
\quad \wedge \\
\overline{\mathrm{dom}\, \mathit{COp}} \subseteq \mathrm{dom}(T \ntriangleright \mathrm{dom}\, AOp)
\end{array}
$$

For finalisation, as with downward simulation, we obtain an applicability condition comparable to that for operations

$$\forall c : CState \bullet c(\!|\, T \,|\!) \subseteq \mathrm{dom}\, AFin \Rightarrow c \in \mathrm{dom}\, CFin$$

plus the "undecorated" version of the original condition

$$CFin \subseteq T \fatsemi AFin$$

Definition 9 (Upward simulation)
Let A = (AState, AInit, $\{AOp_i\}_{i \in I}$, AFin) *and* C = (CState, CInit, $\{COp_i\}_{i \in I}$, CFin) *be data types where the operations may be partial.*

An upward *simulation is a relation* T *from* CState *to* AState *satisfying, in the contract interpretation*

$$
\begin{array}{l}
CInit \fatsemi T \subseteq AInit \\
CFin \subseteq T \fatsemi AFin \\
\forall c : CState \bullet c(\!|\ T\ |\!) \subseteq dom\, AFin \Rightarrow c \in dom\, CFin \\
\forall i : I \bullet \overline{dom\, COp_i} \subseteq dom(T ⩥ dom\, AOp_i) \\
\forall i : I \bullet dom(T ⩥ dom\, AOp_i) ⩤ COp_i \fatsemi T \subseteq T \fatsemi AOp_i
\end{array}
$$

In the behavioural interpretation, correctness is strengthened to:

$$\forall i : I \bullet COp_i \fatsemi T \subseteq T \fatsemi AOp_i$$

Theorem 4 (Upward simulations are total)
When the concrete finalisation is total, the upward simulation T *from* CState *to* AState *is* total *on* CState.

Transforming into Z

These rules can be written in the Z schema calculus, where they become:

- $\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$
- $\forall\, AState;\ CState \bullet$
  $\mathrm{pre}\, AOp \wedge R \Longrightarrow \mathrm{pre}\, COp$
- $\forall\, AState;\ CState;\ CState' \bullet$
  $\mathrm{pre}\, AOp \wedge R \wedge COp \Longrightarrow \exists\, AState' \bullet R' \wedge AOp$

These are the three conditions you need to prove to verify a refinement. You have to prove the 2nd and 3rd conditions for all operations.

Finalisation has disappeared because output appears at each operation step, rather than waiting to the end.

We now explore these issues.

Refinement in Z

To derive simulation rules in Z, interpret initialisation, operations etc as appropriate relations:

Init picks a suitable initial state and copies over the sequence of inputs from the global state. Op_i consumes an input, produces an output and the state is transformed according to the operation:

$$\text{Init is } (is, os) \mapsto (is, \langle\rangle, \theta State')$$

$$\text{Op}_i \text{ is } (\langle \theta ?Op_i \rangle \frown is, os, \theta State) \mapsto (is, os \frown \langle \theta !Op_i \rangle, \theta State')$$

Finalisation just makes all outputs visible:

$$\text{Fin is } (is, os, \theta State) \mapsto (\langle\rangle, os)$$

Given this embedding, we can translate the downward and upward simulation conditions into the Z schema calculus.

Example - without input or output.

The standard Z ADT $(B, Init, \{On, Off\})$ where

B
$b : \mathbb{B}$

Init
B'
---
b'

On
ΔB

$\neg b \wedge b'$

Off
ΔB

$b \wedge \neg b'$

is interpreted as the relational data type (B, Init, {On, Off}, Fin) where

$$
\begin{array}{l}
B == \{ \langle\!|\ b == true\ |\!\rangle, \langle\!|\ b == false\ |\!\rangle \} \\
Init == \{ * \mapsto \langle\!|\ b == true\ |\!\rangle \} \\
On == \{ \langle\!|\ b == false\ |\!\rangle \mapsto \langle\!|\ b == true\ |\!\rangle \} \\
Off == \{ \langle\!|\ b == true\ |\!\rangle \mapsto \langle\!|\ b == false\ |\!\rangle \} \\
Fin == \{ \langle\!|\ b == true\ |\!\rangle \mapsto *, \langle\!|\ b == false\ |\!\rangle \mapsto * \}
\end{array}
$$

A retrieve relation R between *AState* and *CState* must be embedded in the relational setting similarly to how we embedded operations:

$$\mathsf{R} == \{\mathsf{R} \bullet \theta\mathsf{AState} \mapsto \theta\mathsf{CState}\}$$

The retrieve relation R between B in and $N \mathrel{\widehat=} [x : \{0,1\}]$ given by

$$\begin{array}{|l}
\multicolumn{1}{l}{R} \\
\hline
B;\ N \\
\hline
b = (x = 1) \\
\hline
\end{array}$$

is represented by the relation

$$\begin{aligned}\mathsf{R} == \{&\langle\!|\ \mathsf{b} == true\ |\!\rangle \mapsto \langle\!|\ x == 1\ |\!\rangle, \\ &\langle\!|\ b == false\ |\!\rangle \mapsto \langle\!|\ x == 0\ |\!\rangle\}\end{aligned}$$

Deriving Downward Simulation in Z

Given the embedding above, we can translate the relational refinement conditions of simulations into refinement conditions for Z.

Initialisation:

$$\begin{aligned}
&\mathsf{CInit} \subseteq \mathsf{AInit} \mathbin{⨾} \mathsf{R} \\
&\quad \equiv \{\ \text{definition of } \subseteq\ \} \\
&\forall g, c' \bullet (g, c') \in \mathsf{CInit} \Rightarrow (\mathsf{g}, \mathsf{c}') \in \mathsf{AInit} \mathbin{⨾} \mathsf{R} \\
&\quad \equiv \{\ \text{definition of } ⨾\ \} \\
&\forall g, c' \bullet (g, c') \in \mathsf{CInit} \\
&\quad \Rightarrow \exists\, a' \bullet (g, a') \in \mathsf{AInit} \land (\mathsf{a}', \mathsf{c}') \in \mathsf{R} \\
&\quad \equiv \{\ \text{interpretation of } \mathit{Init} \text{ and } R';\ g = *\} \\
&\forall c' \bullet c' \in \{\mathit{CInit} \bullet \theta \mathit{CState}'\} \Rightarrow \\
&\qquad \exists\, a' \bullet a' \in \{\mathit{AInit} \bullet \theta \mathit{AState}'\} \\
&\qquad\quad \land (a', c') \in \{R' \bullet \theta \mathit{AState}' \mapsto \theta \mathit{CState}'\} \\
&\quad \equiv \{\ \text{use schema quantification}\ \} \\
&\forall\, \mathit{CState}' \bullet \mathit{CInit} \Rightarrow \exists\, \mathit{AState}' \bullet \mathit{AInit} \land R'
\end{aligned}$$

For finalisation:

$$\begin{aligned}
&\mathsf{R} \mathbin{⨾} \mathsf{CFin} \subseteq \mathsf{AFin} \\
&\quad \equiv \{\ \text{definition of } \subseteq\ \} \\
&\forall g, a \bullet (a, g) \in \mathsf{R} \mathbin{⨾} \mathsf{CFin} \Rightarrow (\mathsf{a}, \mathsf{g}) \in \mathsf{AFin} \\
&\quad \equiv \{\ \text{definition of AFin}\ \} \\
&\forall g, a \bullet (a, g) \in \mathsf{R} \mathbin{⨾} \mathsf{CFin} \Rightarrow (\mathsf{a} \in \mathsf{AState} \land \mathsf{g} = *) \\
&\quad \equiv \{\ \mathrm{ran}\, \mathsf{CFin} = \{*\},\ \mathrm{dom}\, \mathsf{R} \subseteq \mathsf{AState}\ \} \\
&true
\end{aligned}$$

and

$$\begin{aligned}
&\mathrm{ran}(\mathrm{dom}\, \mathsf{AFin} \lhd \mathsf{R}) \subseteq \mathrm{dom}\, \mathsf{CFin} \\
&\quad \equiv \{\ \mathrm{dom}\, \mathsf{CFin} = \mathsf{CState}\ \} \\
&true
\end{aligned}$$

Applicability:

$$\begin{aligned}
&\mathrm{ran}(\mathrm{dom}\, \mathsf{AOp_i} \lhd \mathsf{R}) \subseteq \mathrm{dom}\, \mathsf{COp_i} \\
&\quad \equiv \\
&\forall\, \mathit{CState};\ \mathit{AState} \bullet R \land \mathrm{pre}\, \mathit{AOp_i} \Rightarrow \mathrm{pre}\, \mathit{COp_i}
\end{aligned}$$

Correctness:

$$\begin{aligned}
&(\mathrm{dom}\, \mathsf{AOp_i} \lhd \mathsf{R}) \mathbin{⨾} \mathsf{COp_i} \subseteq \mathsf{AOp_i} \mathbin{⨾} \mathsf{R} \\
&\quad \equiv \\
&\forall\, \mathit{AState};\ \mathit{CState};\ \mathit{CState}' \bullet \\
&\qquad R \land \mathrm{pre}\, \mathit{AOp_i} \land \mathit{COp_i} \\
&\qquad\quad \Rightarrow \exists\, \mathit{AState}' \bullet \mathit{AOp_i} \land R'
\end{aligned}$$

Together, these conditions make up the rules for downward simulation for Z schemas in systems without input or output.

The relation R on $AState \wedge CState$ is a *downward simulation* from A to C if

$$\forall CState' \bullet CInit \Rightarrow \exists AState' \bullet AInit \wedge R'$$

and for all $i \in I$:

$$\begin{array}{l} \forall AState;\ CState \bullet \\ \qquad \text{pre}\, AOp_i \wedge R \Rightarrow \text{pre}\, COp_i \\ \forall AState;\ CState;\ CState' \bullet \\ \qquad \text{pre}\, AOp_i \wedge R \wedge COp_i \\ \qquad\qquad \Rightarrow \exists AState' \bullet R' \wedge AOp_i \end{array}$$

In the behavioural interpretation, the rule for correctness becomes

$$\begin{array}{l} \forall AState;\ CState;\ CState' \bullet \\ \qquad R \wedge COp_i \Rightarrow \exists AState' \bullet R' \wedge AOp_i \end{array}$$

When this behavioural correctness condition holds, the applicability condition above is equivalent to

$$\forall AState;\ CState \bullet R \Rightarrow (\text{pre}\, AOp_i \Leftrightarrow \text{pre}\, COp_i)$$

Deriving Upward Simulation

In a similar fashion:

$$\begin{array}{l} \mathrm{CInit} \mathbin{\fatsemi} \mathrm{T} \subseteq \mathrm{AInit} \\ \qquad \equiv \\ \forall AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit \end{array}$$

Finalisation produces an important condition:

$$\begin{array}{l} \mathrm{CFin} \subseteq \mathrm{T} \mathbin{\fatsemi} \mathrm{AFin} \\ \qquad \equiv \\ \forall CState \bullet \exists AState \bullet T \end{array}$$

Applicability gives:

$$\begin{array}{l} \overline{\text{dom}\, COp_i} \subseteq \text{dom}(\mathrm{T} \unrhd \text{dom}\, \mathrm{AOp_i}) \\ \qquad \equiv \\ \forall CState \bullet (\forall AState \bullet T \Rightarrow \text{pre}\, AOp_i) \Rightarrow \text{pre}\, COp_i \end{array}$$

For correctness, we have:

$$\begin{array}{l} \text{dom}(\mathrm{T} \unrhd \text{dom}\, \mathrm{AOp_i}) \unlhd \mathrm{COp_i} \mathbin{\fatsemi} \mathrm{T} \subseteq \mathrm{T} \mathbin{\fatsemi} \mathrm{AOp_i} \\ \qquad \equiv \\ \forall AState';\ CState;\ CState' \bullet \\ \qquad (\forall AState \bullet T \Rightarrow \text{pre}\, AOp_i) \\ \qquad\qquad \Rightarrow (COp_i \wedge T' \Rightarrow \exists AState \bullet T \wedge AOp_i) \end{array}$$

Together these become:

The relation T on $AState \wedge CState$ is an *upward simulation* from A to C if

$$\begin{array}{l} \forall CState \bullet \exists AState \bullet T \\ \forall AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit \end{array}$$

and for all $i \in I$:

$$\begin{array}{l} \forall CState \bullet (\forall AState \bullet T \Rightarrow \text{pre}\, AOp_i) \Rightarrow \text{pre}\, COp_i \\ \forall AState';\ CState;\ CState' \bullet \\ \qquad (\forall AState \bullet T \Rightarrow \text{pre}\, AOp_i) \\ \qquad\qquad \Rightarrow (COp_i \wedge T' \Rightarrow \exists AState \bullet T \wedge AOp_i) \end{array}$$

Under the assumption of totality of T, the applicability condition can be written

$$\forall CState \bullet \exists AState \bullet T \wedge (\text{pre}\, AOp_i \Rightarrow \text{pre}\, COp_i)$$

In the behavioural interpretation, the rule for correctness becomes

$$\begin{array}{l} \forall AState';\ CState;\ CState' \bullet \\ \qquad (COp_i \wedge T') \Rightarrow \exists AState \bullet T \wedge AOp_i \end{array}$$

Downward and upward simulations allow the same sort of changes in a refinement: preconditions can be weakened and non-determinism in postconditions resolved.

But ... downward simulations do not allow postponement of non-deterministic choice.

In a downward simulation a concrete program is simulated starting in the initial state, and each concrete step is then matched by an abstract one. They are sometimes called *forward* simulations.

In an upward simulation an arbitrary point in a concrete program is picked and the simulation works backwards to see if it could be simulated from some abstract initialisation. They are sometimes called *backward* simulations.

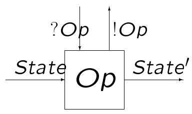
This also explains why upward simulations need to be total.

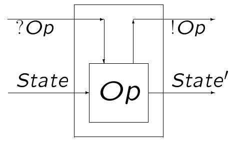
Totality is needed because the upward simulation begins at an arbitrary point in the concrete program, and we need to be sure that from any such point we can simulate backwards.

Because downward simulations begin at the initialisation totality is not necessary for their retrieve relations.

Embedding Inputs and Outputs

How to apply to operations with input and output?





Add input and output information to the global and local state:

$\mathsf{G} == \mathsf{seq\,Input} \times \mathsf{seq\,Output}$

$\mathsf{State} == \mathsf{seq\,Input} \times \mathsf{seq\,Output} \times \mathsf{State}$

Init picks a suitable initial state and copies over the sequence of inputs from the global state.

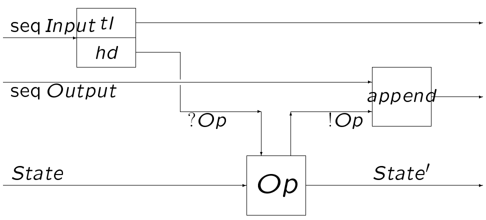
Init is $(is, os) \mapsto (is, \langle\rangle, \theta State')$

Finalisation just makes all outputs visible:

Fin is $(is, os, \theta State) \mapsto (\langle\rangle, os)$

$\mathsf{Op_i}$ consumes an input, produces an output and the state is transformed according to the operation:

$$\mathsf{Op_i} \text{ is } (\langle\theta?\mathsf{Op_i}\rangle \frown \mathsf{is}, \mathsf{os}, \theta\mathsf{State}) \mapsto (is, os \frown \langle\theta!Op_i\rangle, \theta State')$$



We can now derive the simulation rules again, the result is essentially the *same* set of rules except that we have to quantify over inputs and outputs.

The given finalisation is total, so the applicability condition for finalisation holds. We also have:

$$
\begin{array}{l}
\mathsf{R} \fatsemi \mathsf{CF} \subseteq \mathsf{AF} \\
\quad \equiv \{ \text{ definition of } \subseteq \} \\
\forall\, is, os, a, is', os' \bullet ((is, os, a), (is', os')) \in \mathsf{R} \fatsemi \mathsf{CF} \Rightarrow \\
\qquad\qquad ((is, os, a), (is', os')) \in \mathsf{AF} \\
\quad \equiv \{ \ \} \\
\forall\, is, os, a \bullet ((is, os, a), (\langle\rangle, os) \in \mathsf{R} \fatsemi \mathsf{CF} \Rightarrow \\
\qquad\qquad ((is, os, a), (\langle\rangle, os) \in \mathsf{AF} \\
\quad \Leftarrow \{ \text{ predicate calculus } \} \\
\forall\, is, os, a \bullet ((is, os, a), (\langle\rangle, os) \in \mathsf{AF} \\
\quad \equiv \{ \text{ definition of AF } \} \\
true
\end{array}
$$

Downward simulations - Operation refinement

When the retrieve relation is the identity, e.g., when we use the same state space in abstract and concrete, the rules are simplified:

- $\forall \mathit{State}' \bullet \mathit{CInit} \Rightarrow \mathit{AInit}$
- $\forall \mathit{State} \bullet \mathrm{pre}\ \mathit{AOp} \implies \mathrm{pre}\ \mathit{COp}$
- $\forall \mathit{State};\ \mathit{State}' \bullet \mathrm{pre}\ \mathit{AOp} \land \mathit{COp} \implies \mathit{AOp}$

Examples

1. Initialisation condition

TimeHM
$\mathit{hrs} : 0..23$
$\mathit{mins} : 0..59$

TimeHMInit
TimeHM'

CTimeHMInit
TimeHM'
$\mathit{hrs}' = 0 \land \mathit{mins}' = 0$

Then (TimeHM, CTimeHMInit) is a refinement of (TimeHM, CTimeHMInit).

We need to show that:

$\forall \mathit{TimeHM}' \bullet \mathit{TimeHMInit} \Rightarrow \mathit{TimeHMInit}$

$\forall \mathit{hrs}' : 0..23;\ \mathit{mins}' : 0..59 \bullet$
$\mathit{hrs}' = 0 \land \mathit{mins}' = 0 \Rightarrow \mathit{true}$

2. Reducing non-determinism in after-state

State
$x : \mathbb{N}$

AOp
$\Delta \mathit{State}$
$x' \in \{0, 1\}$

COp
$\Delta \mathit{State}$
$x' \in \{0\}$

We need to show that:

$\forall \mathit{State} \bullet \mathrm{pre}\ \mathit{AOp} \implies \mathrm{pre}\ \mathit{COp}$
$\forall \mathit{State};\ \mathit{State}' \bullet \mathrm{pre}\ \mathit{AOp} \land \mathit{COp} \implies \mathit{AOp}$

latter is (why?):

$\forall x, x' \bullet x' \in \{0\} \implies x' \in \{0, 1\}$

3. Reducing non-determinism in output

State
$x : \mathbb{N}$

Init
State'
$x' = 1$

AOp
$\Delta \mathit{State}$
$n! : \mathbb{N}$
$n! \in \{0, 1\}$

COp
$\Delta \mathit{State}$
$n! : \mathbb{N}$
$n! \in \{0\}$

We need to show that (+ initialisation condition):

$\forall \mathit{State} \bullet \mathrm{pre}\ \mathit{AOp} \implies \mathrm{pre}\ \mathit{COp}$
$\forall \mathit{State};\ \mathit{State}' \bullet \mathrm{pre}\ \mathit{AOp} \land \mathit{COp} \implies \mathit{AOp}$

latter is:

$\forall x;\ x';\ n! \bullet n! \in \{0\} \implies n! \in \{0, 1\}$

4. Widening the pre-condition

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{State}} \\ \hline x : \mathbb{N} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{Init}} \\ \hline \textit{State}' \\ \hline x' = 1 \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{AOp}} \\ \hline \Delta \textit{State} \\ n! : \mathbb{N} \\ \hline x = 1 \wedge n! \in \{0, 1\} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{COp}} \\ \hline \Delta \textit{State} \\ n! : \mathbb{N} \\ \hline (x = 1 \wedge n! \in \{0\}) \vee \\ (x = 2 \wedge n! = 2) \\ \hline \end{array}$$

We need to show that (+ initialisation condition):

$\forall \textit{State} \bullet \text{pre}\, \textit{AOp} \implies \text{pre}\, \textit{COp}$
$\forall \textit{State}; \textit{State}' \bullet \text{pre}\, \textit{AOp} \wedge \textit{COp} \implies \textit{AOp}$

What is pre $\textit{AOp}$ and pre $\textit{COp}$?

Data refinement

1. Changing variable names

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{AState}} \\ \hline x : \mathbb{N} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{AInit}} \\ \hline \textit{AState}' \\ \hline x' = 1 \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{AOp}} \\ \hline \Delta \textit{AState} \\ n! : \mathbb{N} \\ \hline x = 1 \wedge n! \in \{0, 1\} \\ \hline \end{array}$$

Refinement is:

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{CState}} \\ \hline y : \mathbb{N} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{CInit}} \\ \hline \textit{CState}' \\ \hline y' = 1 \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{COp}} \\ \hline \Delta \textit{CState} \\ n! : \mathbb{N} \\ \hline y = 1 \wedge n! \in \{0, 1\} \\ \hline \end{array}$$

Retrieve relation:

$$\begin{array}{|l} \multicolumn{1}{l}{R} \\ \hline \textit{AState} \\ \textit{CState} \\ \hline \ldots \\ \hline \end{array}$$

Conditions are:

- $\forall \textit{CState}' \bullet \textit{CInit} \Rightarrow \exists \textit{AState}' \bullet \textit{AInit} \wedge R'$
- $\forall \textit{AState}; \textit{CState} \bullet$
  $\text{pre}\, \textit{AOp} \wedge R \implies \text{pre}\, \textit{COp}$
- $\forall \textit{AState}; \textit{CState}; \textit{CState}' \bullet$
  $\text{pre}\, \textit{AOp} \wedge R \wedge \textit{COp} \implies \exists \textit{AState}' \bullet R' \wedge \textit{AOp}$

2. Watch example

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{TimeHM}} \\ \hline \textit{hrs} : 0..23 \\ \textit{mins} : 0..59 \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{TimeHMInit}} \\ \hline \textit{TimeHM}' \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{ShowTime}} \\ \hline \Xi \textit{TimeHM} \\ \textit{hrs}! : 0..23 \\ \textit{mins}! : 0..59 \\ \hline \textit{hrs} = \textit{hrs}! \\ \textit{mins} = \textit{mins}! \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{ResetTime}} \\ \hline \Delta \textit{TimeHM} \\ \textit{hrs}? : 0..23 \\ \textit{mins}? : 0..59 \\ \hline \textit{hrs}' = \textit{hrs}? \\ \textit{mins}' = \textit{mins}? \\ \hline \end{array}$$

Refine this to:

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{CTimeHM}} \\ \hline \textit{time} : \mathbb{N} \times \mathbb{N} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{CTimeHMInit}} \\ \hline \textit{CTimeHM}' \\ \hline \textit{time}' = (0, 0) \\ \hline \end{array}$$

CShowTime

$\Xi CTimeHM$
$hrs! : 0..23$
$mins! : 0..59$

$time.1 = hrs!$
$time.2 = mins!$

CResetTime

$\Delta CTimeHM$
$hrs? : 0..23$
$mins? : 0..59$

$time' = (hrs?, mins?)$

Retrieve relation:

R

$TimeHM$
$CTimeHM$

$\ldots$

$$
\begin{array}{l}
\forall\, CTimeHM' \bullet CTimeHMInit \Rightarrow \\
\qquad \exists\, TimeHM' \bullet TimeHMInit \wedge R' \\
\forall\, TimeHM;\ CTimeHM \bullet \\
\qquad \mathrm{pre}\, ShowTime \wedge R \implies \mathrm{pre}\, CShowTime \\
\forall\, TimeHM;\ CTimeHM;\ CTimeHM' \bullet \\
\qquad \mathrm{pre}\, ShowTime \wedge R \wedge CShowTime \implies \\
\qquad\qquad \exists\, TimeHM' \bullet R' \wedge ShowTime
\end{array}
$$

etc

3. Changing sets into sequences

$[Person]$

AState

$d : \mathbb{P}\, Person$

$\#d \leq Max$

AInit

$AState'$

$d' = \varnothing$

AEnter

$\Delta AState$
$p? : Person$

$\#d < Max$
$p? \notin d$
$d' = d \cup \{p?\}$

ALeave

$\Delta AState$
$p? : Person$

$p? \in d$
$d' = d \setminus \{p?\}$

Refinement uses sequences instead of sets to record the people in the class:

CState

$l : \mathrm{iseq}\, Person$

$\#l \leq Max$

CInit

$CState'$

$l' = \langle\rangle$

CEnter

$\Delta CState$
$p? : Person$

$\#l < Max$
$p? \notin \mathrm{ran}\, l$
$l' = l \frown \langle p? \rangle$

CLeave

$\Delta CState$
$p? : Person$

$p? \in \mathrm{ran}\, l$
$\mathrm{ran}\, l' = \mathrm{ran}\, l \setminus \{p?\}$

What is R?

What are the conditions we have to prove?

Example

The abstract specification evaluates the mean of a set of real numbers in a bag.

AState

$b : \mathrm{bag}\, \mathbb{R}$

AInit

$AState'$

$b' = [\![\,]\!]$

$Enter_A$

$\Delta AState$
$r? : \mathbb{R}$

$b' = b \uplus [\![r?]\!]$

$Mean_A$

$\Xi AState$
$m! : \mathbb{R}$

$b \neq [\![\,]\!] \wedge m! = (\Sigma\, b)/\#b$

The concrete specification only maintains a running sum and a count of the items in the bag.

CState

$s : \mathbb{R}$
$n : \mathbb{N}$

CInit

$CState'$

$s' = 0 \wedge n' = 0$

$$
\begin{array}{|l}
\hline
Enter_C \\
\Delta CState \\
r? : \mathbb{R} \\
\hline
s' = s + r? \\
n' = n + 1 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
Mean_C \\
\Xi CState \\
m! : \mathbb{R} \\
\hline
n \neq 0 \land m! = s / n \\
\hline
\end{array}
$$

The required retrieve relation is

$$
\begin{array}{|l}
\hline
R \\
AState \\
CState \\
\hline
s = \Sigma b \land n = \# b \\
\hline
\end{array}
$$

Unifying Concurrent and Relational Refinement

Why is a process algebra blessed with a multitude of refinement relations, whereas a language like Z only has one notion of refinement?

How can one use Z to specify concurrent systems?

What is the difference between failures-divergences refinement and an input/output model?

Other approaches to refinement

In process algebras there are a range of refinement relations, each with different strengths.

These include trace and failure-divergence refinement, plus equivalences such as weak and strong bisimulation.

Trace refinement

Trace refinement checks that safety properties are preserved.

These are characterised by saying 'nothing bad happens'.

If nothing bad happens in the abstract specification, then nothing bad should happen in a refinement.

This is achieved by asking for trace subsetting.

A *trace* is a sequence of events that can happen in the process.

E.g., for $P = a;\ b;\ stop$ the following are all valid traces of P:

$\epsilon, \langle a \rangle, \langle a, b \rangle$

We normally write $\langle a, b \rangle$ as ab.

The set of traces of a specification (or process) P is denoted $Traces(P)$.

Trace refinement is defined by saying P_0 is refined by P_1 (written $P_0 \sqsubseteq_{tr} P_1$) whenever:

$Traces(P_1) \subseteq Traces(P_0)$

Examples:

$a;\ b;\ stop \sqsubseteq_{tr} a;\ stop$

$a;\ b;\ stop \sqsubseteq_{tr} stop$

But:

$a;\ b;\ c;\ stop \not\sqsubseteq_{tr} a;\ c;\ stop$

$a;\ b;\ c;\ stop \not\sqsubseteq_{tr} a;\ b;\ c;\ d;\ stop$

Trace refinement fails to distinguish between processes that we would naturally think of as different (technically - be able to construct *tests* that distinguish them).

For example, we can't distinguish between

$a;\ (b;\ stop [] c;\ stop)$

and

$(a;\ b;\ stop) [] (a;\ c;\ stop)$

Failures record more information. A failure is a pair (tr, X) where tr is a trace of the process, and X is a *refusal set*.

A refusal set is a record of events that the process would refuse after that trace.

Failures(P) is the set of all failures of the process P.

E.g., for $P = a;\ b;\ stop$ the following are failures of P:

$(\epsilon, \{b\}), (a, \{a\}), (ab, \{a, b\})$

Failures refinement asks for subsetting of failure sets:

P_1 is a failures refinement of P_0 if:

$Failures(P_1) \subseteq Failures(P_0)$

We can now distinguish between

$a;\ (b;\ stop [] c;\ stop)$

and

$(a;\ b;\ stop) [] (a;\ c;\ stop)$

Why? Calculate their failures.

Failures-divergences refinement

The traces of events are recorded as opposed to an input/output relation.

For example, in failures-divergences refinement a process is modelled by the triple (A, F, D) where A is its alphabet, F is its *failures* and D is its *divergences*.

The failures of a process are pairs (t, X) where t is a finite sequence of events that the process may undergo and X is a set of events the process may refuse to perform after undergoing t.

A process Q is a refinement of a process P if

$failures\ Q \subseteq failures\ P$
$divergences\ Q \subseteq divergences\ P$

Differences

Relational refinement is only concerned with the relation between input and output, since that is all that is observed in the global state.

In a process algebra the event names have an importance not attributed to them in the relational setting.

Relational refinement looks at reduction of non-determinism visible in the global state as given by programs.

Failures-divergences refinement is not just concerned with traces but also with refusals and divergences.

The relationship between these two views is not immediately obvious.

cf. Work of Bolton and Davies shows that failures-divergences refinement is *not* the same as relational refinement as in Z.

Adding refusals to relational refinement

The basic observations the relational model makes are thus very weak:

- make more observations,
- the only way we can do this is to increase the expressitivity of the finalisation.

Generalise finalisation from being

$(is, os, \theta State) \mapsto (\langle\rangle, os)$

to becoming

$(is, os, \theta State) \mapsto (\langle\rangle, os, E)$

What goes in E depends on what we want to observe.

Example - no input or output

B

$b : \mathbb{B}$

Init

B'

---

b'

On

ΔB

$\neg b \wedge b'$

Off

ΔB

$b \wedge \neg b'$

Make E refusals, finalisation changes from

$\mathsf{Fin} == \{\ \langle\!\vert\ b == true\ \vert\!\rangle \mapsto *, \ldots\}$

to

$$\begin{aligned}\mathsf{Fin} == \{\ & \langle\!\vert\ b == true\ \vert\!\rangle \mapsto \{On\},\\ & \langle\!\vert\ b == true\ \vert\!\rangle \mapsto \varnothing,\\ & \langle\!\vert\ b == false\ \vert\!\rangle \mapsto \{Off\},\\ & \langle\!\vert\ b == false\ \vert\!\rangle \mapsto \varnothing\}\end{aligned}$$

Simulation rules - no input or output

1. Downward simulation

$\mathrm{R} \mathbin{⨾} \mathrm{CFin} \subseteq \mathrm{AFin}$

becomes

$\forall R \bullet (\{i : I \mid \neg \operatorname{pre} COp_i\} \subseteq \{i : I \mid \neg \operatorname{pre} AOp_i\})$

This places no extra conditions on a downward simulation.

2. Upward simulation

$\mathrm{CFin} \subseteq \mathrm{T} \mathbin{⨾} \mathrm{AFin}$

becomes

$$\begin{aligned}&\forall CState \bullet \exists AState \bullet T \wedge\\ &\quad (\{i : I \mid \neg \operatorname{pre} COp_i\} \subseteq \{i : I \mid \neg \operatorname{pre} AOp_i\})\end{aligned}$$

This is,

$\forall CState \bullet \exists AState \bullet \forall i : I \bullet \ldots$

instead of

$\forall i : I \bullet \forall CState \bullet \exists AState \bullet \ldots$

Notes

We have extended our observations beyond acceptance of traces and now record refusals.

This appears in the simulation rules via the quantification over i that occurs within the scope of the quantification over *AState* and *CState*.

Although we began by embedding refusals in a finalisation, the conditions can be unwound to be expressed in terms of the schema calculus directly.

The conditions extracted from this finalisation requirement work with both the blocking and the non-blocking model of preconditions.

The correspondence with failures-divergences refinement

This relational refinement corresponds in some sense to failures-divergences refinement.

In some sense means it is modulo an encoding of the specification as a set of events, however, this encoding is natural and uncontroversial.

Theorem 5 *In the blocking model, relational refinement with extended finalisations corresponds to failures-divergences refinement.*

In the non-blocking model the encoding is slightly different, e.g., there are divergences, but no refusals beyond those after a divergence.

Theorem 6 *In the non-blocking model, relational refinement with extended finalisations corresponds to failures-divergences refinement.*

76

Dealing with input and output (blocking model)

- Describe the correct corresponding process and its failures and divergences,
- Define the finalisation,
- Prove that relational refinement is the same as failures-divergences refinement, and
- Extract simulation rules from the relational refinement, expressing them in the schema calculus.

77

Refusals of an event with output:

- Demonic - the environment cannot influence the output, and there are refusals due to a particular output being chosen, or
- Angelic - the environment can influence the output, and there are no such refusals.

Finalisation includes these refusals.

78

The simulation rules

1. Downward simulation in the demonic model

Theorem 7 *The condition* $\mathrm{R} \mathbin{⨾} \mathrm{CFin} \subseteq \mathrm{AFin}$ *in the demonic model is subsumed by the normal applicability and correctness rules.*

2. Upward simulation in the demonic model

Example 1

A and C are *not* failures-divergences equivalent.

A has failure $(\langle B \rangle, \{TVF, ESF\})$ which is not present in C

To recover failures-divergences refinement in the blocking model one needs to add the strengthened applicability condition:

$$\forall\, CState \bullet \exists\, AState \bullet \forall\, i : I \bullet \\ T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$

79

Example 2

This example shows that we need an additional condition on refusal sets due to outputs as well.

Now the strengthened applicability condition holds

- each state *simple*, *luxury*, *tv* and *ensuite* have operations *HasES* and *HasTV* enabled -

so this does not pick up the different refusal information due to the outputs.

Simulation rules

Based on *maximal* refusal sets.

$Sim == (I \times Input) \rightarrow\!\!\!\!\!+ \; Output$

$E : Sim$ represents:

$\overline{\mathrm{dom}\, E}$ disabled;
$\mathrm{dom}\, E$ enabled, outputs in E chosen

encoded as $Maxref(E, State)$
The finalisation condition is:

$\forall\, CState;\ E : Sim \bullet Maxref(E, CState)$
$\Rightarrow \exists\, AState;\ E' \subseteq E \bullet T \wedge MaxRef(E', AState)$

Simpler for the angelic model of outputs.

Downward simulations - summary

Outputs:	none	Demonic	Angelic
Init		**Init**	
App	**App**		-
Corr		**CorrBlock**	
Fin	-		**FinAng**

Init $\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$

App $\forall\, R;\ i : I;\ Input \bullet \mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i$

CorrBlock $\forall\, i : I;\ Input;\ Output;\ R;\ CState' \bullet$
$COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$

FinAng $\forall\, R;\ i : I;\ Input;\ Output \bullet$
$Pre\ AOp_i \Rightarrow Pre\ COp_i$

where in angelic rules we alter the definition of pre Op to include existential quantification of the after state only, i.e.,
Pre $Op \mathrel{\widehat=} \exists\, State' \bullet Op$.

Upward simulations - summary

Outputs:	none	Demonic	Angelic
Init		**Init**	
App		-	
Corr		**CorrBlock**	
Fin	**FinRef**	**FinDem**	**FinAng**

Init $\forall\, T' \bullet CInit \Rightarrow AInit$

CorrBlock $\forall\, i : I;\ Input;\ Output;\ T';\ CState \bullet$
$COp_i \Rightarrow \exists\, AState \bullet T \wedge AOp_i$

FinRef $\forall\, CState \bullet \exists\, AState \bullet T \wedge \forall\, i : I;\ Input \bullet$
$\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i$

FinDem $\forall\, CState;\ E : Sim \bullet$
$Maxref(E, CState) \Rightarrow \ldots$

FinAng $\forall\, CState \bullet \exists\, AState \bullet T \wedge$
$\forall\, Input;\ Output;\ i : I \bullet \mathrm{Pre}\ AOp_i \Rightarrow \mathrm{Pre}\ COp_i$

Relations for Specifying the Invariant Behavior of Object Collaborations

Stephanie Balzer

ETH Zurich (Swiss Federal Institute of Technology)
Department of Computer Science
CH-8092 Zürich, Switzerland
stephanie.balzer@inf.ethz.ch

Abstract. The missing first-class support of object collaborations in class-based object-oriented programming languages is increasingly criticized as it results in distributing the information about such collaborations across multiple classes. In response, the introduction of *first-class relationships* is proposed. Relationships are the programming language abstractions that encapsulate the collaborations that emerge from the interacting objects. With first-class support, relationships exist in addition to classes and can — like classes — define their own members, such as attributes and methods. Our work enriches the concept of first-class relationships with the notion of *structural invariants*. Structural invariants specify the invariant behavior of object collaborations. They restrict the participation of objects in relationships and are expressed in terms of *mathematical relations*.

1 Introduction

Common class-based object-oriented programming languages do not provide the necessary abstractions to reflect object collaborations. Object collaborations are implemented using (unidirectional) references. This approximation results in the loss of a global view as the information about a given collaboration may be distributed across multiple classes [1]. In response, numerous authors propose to preserve collaborations from the design to the implementation stage [2–4, 1, 5]. Whereas Noble and Grundy [4] stay within the limits of the programming language and represent collaborations explicitly using separate classes, Rumbaugh [2], Albano et al. [3], and Bierman and Wren [1] go further and extend the programming language to include *relationships*, the abstractions necessary to encapsulate these object collaborations. Relationships together with classes then constitute the first-class notions of the programming language. Like classes, relationships can also declare attributes and methods.

Our work relies on the foundations of previous work on first-class relationships. However, we significantly differ from existing approaches in that we include *invariants* to allow reasoning about programs with classes and relationships. The application of invariants to class-based object-oriented programming and specification languages [6–10] has already proved viable to express the consistency conditions of the instances of a class. We want to exploit the benefits of invariants also to specify the invariant behavior of object collaborations. We thus introduce the concepts of *value-based* and *structural invariants* [11]. Unlike traditional class-based invariants, relationship invariants are specified on the collaborating classes. Due to a shortage of space, we only discuss structural invariants in this paper.

An important contribution of our work is the observation that structural invariants can be expressed by means of *mathematical relations*. Using relations, we have the full mathematical expressiveness at our disposal when restricting relationship participation. The mathematical basis further allows us to reason on the interdependences between structural invariants and any operations that change relationship participation.

The remainder of this paper is structured as follows: Section 2 provides a short introduction to the modeling of object collaborations. Section 3 discusses the concept of structural invariants and shows their application to the specification of object collaborations. Section 4 describes future work and Section 5 concludes this paper.

2 Example

Figure 1 depicts an Entity-Relationship (ER) diagram [12] modeling parts of a person information system, which serves as the running example of this paper. The example illustrates the different collaborations a person can take part in. For example, a person can be a writer and thus become the author of books. Additionally (or alternatively) a person can be a member of a golf club. Figure 1 also displays the cardinality and participation constraints [13] of the relationships. In the case of the author relationship, for example, these constraints specify that every book needs to have at least one author.

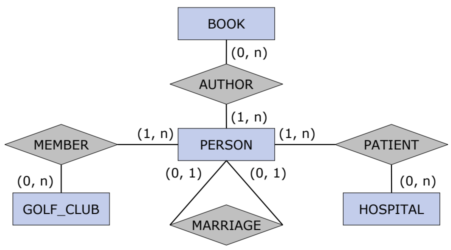


Fig. 1. Person Information System (ER)

Modeling the person information system (see Figure 1) using first-class relationships results in the classes `Person`, `Book`, `GolfClub`, and `Hospital` and the relationships `Author`, `Member`, `Patient`, and `Marriage`.

3 Structural Invariants

Structural invariants restrict the participation of objects in relationships based on the occurrence of objects. A possible restriction is, for example, to require that a particular object participates with at most one other object in a specific relationship. We use *mathematical relations*[1] to express structural invariants. Using relations, we have the full mathematical expressiveness and rigor at our disposal for defining participation restriction. The mathematical basis of our work is also the distinguishing feature that separates structural invariants from related concepts, such as the cardinality and participation constraints of relational databases [13] and the multiplicities as proposed by Bierman and Wren [1].

Applying structural invariants to the person information system results in *surjective relations* for `Author`, `Member`, and `Patient` and a *symmetric, irreflexive partial injection* for `Marriage`.

Figure 2 illustrates the concepts introduced so far on the basis of the `Author` relationship, using a Java-like notation. The relationship has the participant classes `Person` and `Book`. We use role names to denote the role of a participant in a relationship. For instance, a person plays the role of a writer, and a book the one of the author's work. Roles are particularly helpful when a relationship involves instances of the same class. The relationship further declares the attribute `submissionDate`, which records the date the manuscript was submitted. The structural invariant of the `Author` relationship is introduced by the keyword `invariant` and specifies a surjective relation.

```
relationship Author {

  participants(Person writer, Book work);

  //structural invariant
  invariant
    surjectiveRelation(writer, work)

  //attribute of relationship Author
  Date submissionDate;
}
```

Fig. 2. Relationship Author

An illustration of the structural invariant of the `Author` relationship is given in Figure 3. The figure depicts several illustration forms: mathematical relations

[1] To clarify the terminology, note that we use the term *relationship* to refer to the abstraction that encapsulates object collaborations and use the mathematical term *relation* to denote the set of participant pairs of relationships.

in the upper right corner, and a VEN diagram-like representation of relations in the lower right corner. If expressible in terms of cardinality and participation constraints, the respective ER diagram representation is provided additionally. Especially the VEN diagram-like representation nicely illustrates the surjectivity of the relation: every book is connected to a person at least through one line, but not every person is connected to a book.

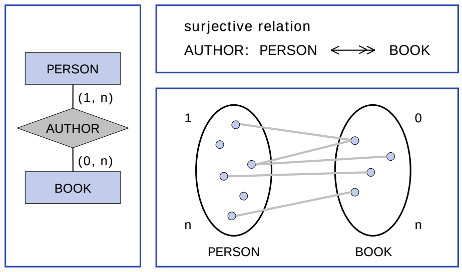


Fig. 3. Illustrations of Structural Invariants

4 Future Work

Structural invariants clearly must hold over the entire lifespan of relationships. Any programming language together with its supportive system implementing the concepts introduced in this paper thus has to provide means, preferably static ones, to maintain those invariants.

Before starting out implementing a particular invariant monitoring mechanism, we specify the semantics of structural invariants. Our current work thus involves the modeling of the interdependences between structural invariants and any operations that change relationship participation. We use *Event-B* [14] for this purpose, a methodology to model software systems based on discrete mathematics and refinement. The resulting model will define the semantics of structural invariants by indicating the invariant-maintaining actions for every kind of relation and operation applied.

5 Concluding Remarks

We have presented in this paper our ongoing work on the specification of object collaborations. The approach we describe relies on the concept of first-class relationships, the programming language abstraction to encapsulate the collaborations that emerge from the interacting instances of classes. The main contribution of our work is to include structural invariants to allow the specification of the invariant behavior of object collaborations. Structural invariants are expressed in terms of mathematical relations, which in turn facilitate the definition of the semantics of structural invariants.

References

1. Bierman, G.M., Wren, A.: First-class relationships in an object-oriented language. In Black, A.P., ed.: ECOOP. Volume 3586 of Lecture Notes in Computer Science., Springer-Verlag GmbH (2005) 262–286
2. Rumbaugh, J.: Relations as semantic constructs in an object-oriented language. In: OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications, New York, NY, USA, ACM Press (1987) 466–481
3. Albano, A., Ghelli, G., Orsini, R.: A relationship mechanism for a strongly typed object-oriented database programming language. In: VLDB. (1991) 565–575
4. Noble, J., Grundy, J.: Explicit relationships in object-oriented development. In Meyer, B., ed.: TOOLS'95: Conference proceedings on the Technology of Object-Oriented Languages and Systems, Prentice-Hall (1995) 211–226
5. Thomas, D.A.: On the next move in programming. Journal of Object Technology **5** (2006) 7–11
6. Meyer, B.: Object-Oriented Software Construction. Second edn. Prentice Hall Professional Technical Reference (1997)
7. Meyer, B.: Eiffel: The Language. Prentice Hall Professional Technical Reference (1991)
8. Barnett, M., Leino, K.R.M., Schulte, W.: The Spec# programming system: An overview. In Barthe, G., Burdy, L., Huisman, M., Lanet, J.L., Muntean, T., eds.: Construction and Analysis of Safe, Secure, and Interoperable Smart Devices: International Workshop, CASSIS 2004. Volume 3362 of Lecture Notes in Computer Science., Springer-Verlag GmbH (2005) 49–69
9. Leavens, G.T., Baker, A.L., Ruby, C.: Preliminary design of JML: A behavioral interface specification language for java. Technical Report 98-06-rev29, Iowa State University (2006)
10. Burdy, L., Cheon, Y., Cok, D.R., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An overview of jml tools and applications. STTT'05: International Journal on Software Tools for Technology Transfer **7** (2005) 212–232
11. Balzer, S., Eugster, P., Gross, T.R.: Value-based and structural invariants for object relationships. Technical report, ETH Zurich (2006)
12. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst. **1** (1976) 9–36
13. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 2nd Edition. Second edn. Benjamin/Cummings (1994)
14. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)

RelAPS: A Proof System for Relational Categories

Joel Glanfield and Michael Winter*

Department of Computer Science,
Brock University,
St. Catharines, Ontario, Canada, L2S 3A1
{jg00de|mwinter}@brocku.ca

Abstract. This paper provides a short introduction to the RelAPS system – an interactive system assisting in proving relation-algebraic theorems.

1 Introduction

In the past thirty years relational methods have become of fundamental importance in computer science. Especially, theories of relations based on category theory are used as a tool to describe programs and their behavior. Therefore, proving relation-algebraic theorems has become a part of certain areas of computer science. These proofs usually follow a specific style, e.g., they might be based on chains of inclusions. Since theorem-proving by hand can be rather error-prone and tedious we developed a system that would aid in the imitation of typical relation-algebraic proofs while eliminating some of the negative aspects.

The aim of the RelAPS system is to provide a graphical environment where a user may prove various theorems, within the context of the category of relations, as if the proof were being done by hand [4]. It should be noted that automatic theorem-proving is not a goal of the system, i.e., RelAPS is not a theorem-prover.

Contrary to the RALF system [5, 6] RelAPS is text-based and does not use a tree representation of the corresponding formula/term object. It should be mentioned that RelAPS focuses on a subset of the formulae, and is, therefore, not as general as RALF. However, future extensions of RelAPS aim in a different direction (see Section 5), which motivates this restriction.

2 Relational Preliminaries

Throughout this paper, we use the following notation. To indicate that a morphism R of a category $\mathcal{R}$ has source A and target B we write $R : A \to B$. The

* The author gratefully acknowledges support from the Natural Sciences and Engineering Research Council of Canada.

collection of all morphisms $R : A \to B$ is denoted by $\mathcal{R}[A,B]$ and the composition of a morphism $R : A \to B$ followed by a morphism $S : B \to C$ by $R;S$. Last but not least, the identity morphism on A is denoted by $\mathbb{I}_A$.

We recall briefly the notion of a Schröder category introduced in [7]. Similar approaches were taken in [2, 8, 9]. Within the numerous equivalent definitions of Schröder categories we have chosen the following axiomatization:

Definition 1. *A Schröder category $\mathcal{R}$ is a category satisfying the following:*

1. *For all objects A and B the collection $\mathcal{R}[A,B]$ is a Boolean algebra. Meet, join, negation, least and greatest element and the induced ordering are denoted by $\sqcap, \sqcup, \overline{}, \perp\!\!\!\perp_{AB}, \top\!\!\!\top_{AB}$ and $\sqsubseteq$, respectively. The morphisms are also called relations.*
2. *There is a unary operation $\smile$ (called converse) mapping a relation $R : A \to B$ to a relation $R^{\smile} : B \to A$.*
3. *For all relations $Q : A \to B, R : B \to C$ and $S : A \to C$ the modular law $Q;R \sqcap S \sqsubseteq Q;(R \sqcap Q^{\smile};S)$ holds.*
4. *There is a binary operation $\backslash$ (called right residual) defined by $Q;R \sqsubseteq S \iff R \sqsubseteq Q\backslash S$ for all $Q : A \to B$, $R : B \to C$ and $S : A \to C$.*

Notice, that we may define an operation / (called left residual) by $S/R = (R^{\smile}\backslash S^{\smile})^{\smile}$. This operation is characterized by $Q;R \sqsubseteq S \iff Q \sqsubseteq S/R$ for all $Q : A \to B$, $R : B \to C$ and $S : A \to C$. However, both residuals can be defined in terms of the other operations, i.e., we have $S/R = \overline{\overline{S};R^{\smile}}$ and $Q\backslash S = \overline{Q^{\smile};\overline{S}}$.

The RelAPS system uses a formal language introduced in [10]. The language is based on two kinds of entities - objects and relations. Relational variables are typed by object variables, and consequently, there are quantifiers for each kind of variable. For details we refer to [10].

Since the system is currently limited to the use of ASCII characters, it was necessary to develop a grammar (using ASCII characters) that would represent the language described in [10]. For example, the user may wish to enter formulae requiring the use of any of the symbols mentioned in Definition 1, but alternate representations would be required within this environment. Table 1 displays the translation of the necessary symbols into related ASCII character-tokens.

Expanding all definitions used we finally end up with the following set of axioms provided in the language of RelAPS:

```
(A1)  forall a forall b forall Q:a->b forall R:a->b forall S:a->b
         (Q&R)&S=Q&(R&S)
(A2)  forall a forall b forall Q:a->b forall R:a->b forall S:a->b
         (Q|R)|S=Q|(R|S)
(A3)  forall a forall b forall Q:a->b forall R:a->b Q&R=R&Q
(A4)  forall a forall b forall Q:a->b forall R:a->b Q|R=R|Q
(A5)  forall a forall b forall Q:a->b forall R:a->b Q|(Q&R)=Q
(A6)  forall a forall b forall Q:a->b forall R:a->b Q&(Q|R)=Q
(A7)  forall a forall b forall Q:a->b forall R:a->b
          Q<=R <=> Q&R=Q
```

Table 1. RelAPS Tokens

Token	ASCII
$\forall$	forall
$\exists$	exists
$\sqcap$	&
$\sqcup$	|
$\overline{}$	-
$\perp\!\!\!\perp_{ab}$	Oab
$\top\!\!\!\top_{ab}$	Lab
$\mathbb{I}_a$	Ia
$\sqsubseteq$	<=
$\breve{}$	^
;	;
$\backslash$	\
$\Longleftrightarrow$	<=>
$\Longrightarrow$	=>
$R : A \to B$	R:A->B

```
(A8)  forall a forall b forall Q:a->b forall R:a->b forall S:a->b
          Q&(R|S)=(Q&R)|(Q&S)
(A9)  forall a forall b forall Q:a->b Q&Oab = Oab
(A10) forall a forall b forall Q:a->b Q|Oab = Q
(A11) forall a forall b forall Q:a->b Q&Lab = Q
(A12) forall a forall b forall Q:a->b Q|Lab = Lab
(A13) forall a forall b forall Q:a->b Q&-Q = Oab
(A14) forall a forall b forall Q:a->b Q|-Q = Lab
(A15) forall a forall b forall c forall d
      forall Q:a->b forall R:b->c forall S:c->d
          (Q;R);S=Q;(R;S)
(A16) forall a forall b forall Q:a->b Ia;Q=Q
(A17) forall a forall b forall Q:a->b Q;Ib=Q
(A18) forall a forall b forall c
      forall Q:a->b forall R:b->c forall S:a->c
           Q;R&S<=Q;(R^&Q;S)
(A19) forall a forall b forall c
      forall Q:a->b forall R:a->c forall X:b->c
           X<=Q\R <=> Q;X<=R
```

3 The System

The RelAPS system is designed in such a manner to allow the greatest amount of flexibility with respect to the style of proving different theorems. The system accepts Horn-style formulae as input, using the notation explained in the previous section. The current version of RelAPS accept formulae of the style

$$[quantifiers][(A_{1,1} \wedge ... \wedge A_{1,N_1} \Rightarrow B_1) \wedge \ldots \wedge (A_{M,1} \wedge ... \wedge A_{M,N_M} \Rightarrow B_M)]$$

where $[quantifiers]$ is a list of universal quantifiers with object and/or relational variables, and $A_{1,1}, \ldots, A_{M,N_M}$ and $B_1 \ldots B_M$ are arbitrary atomic formulae. Notice that the system also allows $A \Leftrightarrow B$ in the mantissa for atomic formulae A and B since this is equivalent to $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

The interface itself is divided into different 'windows' or areas where the user can work with different aspects of a single formula. It is this concept that increases flexibility with respect to proof-styles. For example, the interface has areas for dealing with assumptions and assertions respectively, and another area designated as the *working area* where the user performs derivations. Within the assumption and assertions areas, the user may select different subterms or subformulae, move them to the working area, perform some derivation, and then apply the result to the original assumption or assertion. Hence, the metalogical rules, i.e., the rules of the sequence calculus restricted to the subset of formulae used in the system, are actually handled by the layout of the interface.

Other beneficial aspects of the system include, but are not limited to, the options to prove monotonicity, associativity, and commutativity - in order to allow the automatic use of these rules without having to continually specify when to use them.

4 Example

As an example, we will describe how a proof of the formula

$$Q \sqsubseteq R \iff Q \sqcap \overline{R} = \perp\!\!\!\perp_{ab}$$

would be completed using RelAPS. First of all, the formula is entered into the system (with the appropriate typing) as

```
forall a forall b forall Q:a->b forall R:a->b Q<=R <=> Q&-R=0ab
```

We then specify that the equivalence will be proven by separating the related implications. Both statements are to be proven separately. Starting with the first implication

$$Q \sqsubseteq R \Longrightarrow Q \sqcap \overline{R} = \perp\!\!\!\perp_{ab},$$

we first use the deduction theorem by moving the assumption into the respective assumption window. Working with the assertion, we select $Q \sqcap \overline{R}$ with the mouse, and move the term to the *working area*. We then perform a derivation by selecting subterms and applying appropriate axioms giving

$$Q \sqcap \overline{R} \sqsubseteq R \sqcap \overline{R} = \perp\!\!\!\perp_{ab},$$

which concludes this proof.

Next, we prove the converse implication in a similar manner, namely, by first separating the assumption and assertion. Then we select the entire assertion and move it into the *working area*. Using axiom (A7) (an equivalence) we modify the assertion getting $Q \sqcap R = Q$. We replace the assertion by the modified one, select the term $Q \sqcap R$, move it to the working area, and finish the proof equationally similar to the first proof.

Since both implications have been derived appropriately, the system considers the proof of the original equivalence to be complete. The theorem is then automatically appended to the system's collection of theorems.

5 Future Work

Although there are many possible extensions to the RelAPS system, the next phases will involve implementation of modules to consider decidable fragments of relational theory, and to generate LaTeX 2$_\varepsilon$ output of completed derivations.

As for the first, it has been shown in [1] that binary operations including $\langle \sqcap, ;, \smile \rangle$ are decidable. As for the second, having a LaTeX 2$_\varepsilon$ generator will allow the user to conveniently generate proof-text that may be included in publications with little or no modification.

References

1. Dougherty D., Gutiérrez C.: Normal Forms and Reduction for Theories of Binary Relations. LNCS 1833 (2000), 95-109.
2. Furusawa H., Kahl W.: A Study on Symmetric Quotients. Technical Report 1998-06, University of the Federal Armed Forces Munich (1998)
3. Freyd P., Scedrov A.: Categories, Allegories. North-Holland (1990).
4. Glanfield J.: ReWiRe – Reasoning With Relations. COSC 4F90 Computing Project, Brock University (2006)
5. Hattensperger C., Berghammer R., Schmidt G.: RALF - A Relation-Algebraic Formula Manipulation System and Proof Checker. Algebraic Methodology and Software Technology (AMAST '93), Springer (1993), 405-406.
6. Hattensperger C., Kempf P.: Towards a Formal Framework for Heterogeneous Relation Algebra. Inf. Sci. 119(3-4) (1999), 193-203.
7. Olivier J.P., Serrato D.: Catégories de Dedekind. Morphismes dans les Catégories de Schröder. C.R. Acad. Sci. Paris 290 (1980), 939-941.
8. Schmidt G., Ströhlein T.: Relationen und Graphen. Springer (1989); English version: Relations and Graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993).
9. Winter M.: Strukturtheorie heterogener Relationenalgebren mit Anwendung auf Nichtdetermismus in Programmiersprachen. Dissertationsverlag NG Kopierladen GmbH, München (1998)
10. Winter M.: A new Algebraic Approach to L-Fuzzy Relations Convenient to Study Crispness. INS Information Science 139, 233-252 (2001).

f-Generated Kleene Algebra

Peter Höfner

Institut für Informatik, Universität Augsburg
D-86135 Augsburg, Germany
hoefner@informatik.uni-augsburg.de

Abstract. When describing iterations or loops it is well known and common to use the Kleene star. We first show an example for iteration, where the star operation is not adequate, since it just iterate and do not modify the iterated element. Therefore we introduce, as a generalisation of Kleene algebra, the structure of f-generated Kleene algebra, that have an iteration operation which depends on a function f and modify the iterated element in each step.

1 Introduction and Motivation

The use of Kleene star for describing iterations or loops is well known and common (see e.g. [3, 5]). From a theoretical point of view, a^* is the least fixed point of $\lambda x.1 + a \cdot x$ and therefore characterises finite iteration. Nevertheless, as we will see, in some situations it is useful to have an additional function, which modifies an iterated element a in each step; i.e., instead of calculating $a \cdot a \cdot \ldots \cdot a$, we want to get $a \cdot f(a) \cdot \ldots \cdot f^{n-1}(a)$. More precisely, $\lambda x.1 + a \cdot x$ is replaced by $\lambda x.1 + a \cdot f(x)$. As far as we know this generalisation of Kleene star has not been discussed before. Let us motivate our idea by a concrete example.

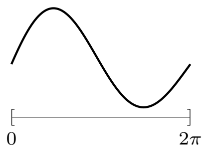
Example 1.1 In [4] we presented an algebra of hybrid systems, which is based on (lazy) Kleene algebra and uses sets of trajectories as elements.[1] A *trajectory* t is a pair (d, g), where $d \in \mathbb{R}_0^+$ and $g : [0, d] \to V$, where d is the *duration* of the trajectory and V a set of (possible) *values*. We define composition of trajectories (d_1, g_1) and (d_2, g_2) as

$$(d_1, g_1) \cdot (d_2, g_2) =_{df} \begin{cases} (d_1 + d_2, g) \text{ if } g_1(d_1) = g_2(0) \\ \text{undefined} \quad \text{otherwise} \end{cases}$$

with $g(x) = g_1(x)$ for all $x \in [0, d_1]$ and $g(x + d_1) = g_2(x)$ for all $x \in [0, d_2]$. Since the algebra uses sets of trajectories as elements, the composition is lifted pointwise to those sets.

[1] For lack of space, we only recapitulate the definition and composition of trajectories and not the whole algebra. Furthermore, we restrict the set of durations to $\mathbb{R}_0^+$, which simplifies the structure and excludes trajectories with infinite length.

We assume the set, which only includes the trajectory $(2\pi, \frac{\sin x}{5x})$. This single set is called $t_{\sin}$. It describes a single swing of a pendulum. The element $t_{\sin}^{*}$ describes sequences of swings, but does not consider the fact that the pendulum gets slower by gravity, friction and so on. □



Sketch of $t_{\sin}$

2 *f*-Generated Kleene Algebra

Motivated by the previous example, we now define an iteration operator w.r.t. to a function f. In the remainder we will use $a, b, c \ldots$ for arbitrary elements of an idempotent semiring S.

Definition 2.1 Let $f : S \to S$ be a homomorphism w.r.t. addition and multiplication, i.e., $f(a+b) = f(a) + f(b)$, $f(0) = 0$ and $f(a \cdot b) = f(a) \cdot f(b)$, $f(1) = 1$.[2] An *f-generated Kleene algebra* is a structure $(S, +, \cdot, 0, 1, {}^{f*})$, such that $(S, +, \cdot, 0, 1)$ is an idempotent semiring and ^{f*} satisfies

$$1 + a \cdot f(a^{f*}) \leq a^{f*} \qquad (f1) \qquad\qquad 1 + a \cdot f(b) \leq b \Rightarrow a^{f*} \leq b \qquad (f2)$$

Similarly to the Kleene star, ^{f*} is the least prefixed point of the function $\lambda x.1 + a \cdot f(x)$ and $(f1)$ can be strengthened to an equation. But in contrast to the Kleene star, we do not postulate the symmetrical laws of $(f1)$ and $(f2)$, since this would imply $a \cdot a^{f*} = a^{f*} \cdot a$. Using fixpoint iteration, we calculate for $\lambda x.1 + a \cdot f(x)$:

$$\begin{aligned}
&x_0 = 0 \\
&x_1 = 1 + a \cdot f(x_0) = 1 \\
&x_2 = 1 + a \cdot f(x_1) = 1 + a \\
&x_3 = 1 + a \cdot f(x_2) = 1 + a + a \cdot f(a) \\
&x_4 = 1 + a \cdot f(x_3) = 1 + a + a \cdot f(a) + a \cdot f(a) \cdot f(f(a))
\end{aligned}$$

In general, we get

$$x_n = 1 + a \cdot f(x_{n-1}) = \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} f^j(a) \ , \qquad (1)$$

where, as usual, $\prod_{i=0}^{-1} x = 1$. Before returning to our example of Section 1, we give some other simple examples, which illustrate that there are more applications of the theory than trajectories.

Example 2.2

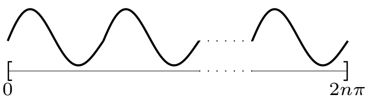
1. The standard Kleene star is subsumed by ^{f*} when choosing f as identity.
2. Using infinite lists, the f-generated Kleene star corresponds to

   $$\mathtt{sum\ scan}\ [\mathtt{1}, \mathtt{a}, \mathtt{f(a)}, \mathtt{f^2(a)}, \ldots]$$

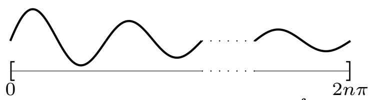
   in functional programming, which is discussed by Bird in [1]. □

[2] In the setting of semirings (monoids) $f(0) = 0$ does not follow from $f(a+b) = f(a) + f(b)$; this implication only holds in groups, rings, ...

Example 1.1 continued By setting $f((d,g)) =_{df} (d, \frac{g}{2})$, we are able to illustrate the difference between t^*_{sin} and t^{f*}_{sin}. A characteristic element of t^*_{sin} just repeats t_{sin} for a finite number of times; whereas a characteristic element of t^{f*}_{sin} repeats and modifies t_{sin}.



Sketch of an element of t^*_{sin}



Sketch of an element of t^{f*}_{sin}

Therefore we are able to describe a pendulum considering changes in time, like gravity, without changing the trajectory itself. Of course the function of our example as well as the trajectory are freely chosen and do not reflect reality. If one wants to describe a real pendulum, the trajectory and the function will getting much more complex; but do not change the idea of ^{f*}. $\square$

Of course, it is also possible to change the duration using another homomorphism f; e.g., to simulate Zeno effects one can set the function $f((d, g(x))) = (\frac{d}{2}, g(\frac{x}{2}))$.

3 Properties

In this section we discuss some properties of ^{f*}. Since the Kleene star is a very special f-generated star, we cannot expect to get all the (well-known) properties of star in our setting. Therefore we also discuss those properties, which hold for Kleene star but not for the f-generated one.

First we give some useful properties of homomorphisms.

Lemma 3.1 *Let $(M, +, 0)$ be a monoid and $f : M \to M$ be a homomorphism w.r.t. addition.*

1. *f preserves idempotency.*
2. *f is isotone, i.e, $a \leq b \Rightarrow f(a) \leq f(b)$.*

Following Kozen's approach to Kleene algebra with tests [5], we say that a *f-generated Kleene algebra with tests* is a f-generated Kleene algebra S with a distinguished Boolean subalgebra $\mathsf{test}(S)$ of $[0,1]$ with greatest element 1 and least element 0. Using tests yields an interesting result concerning f which follows directly from the homomorphism properties.

Lemma 3.2 *If $\mathsf{test}(S)$ is maximal, i.e., there is no Boolean subalgebra B of $[0,1]$ with $\mathsf{test}(S) \subset B$, then $f(p) \in \mathsf{test}(S)$ for all $p \in \mathsf{test}(S)$.*

In the remainder of the section we discuss some properties of ^{f*}. First, we get immediately from $(f1)$ by lattice algebra

$$1 \leq a^{f*} \quad \text{and} \quad a \cdot f(a^{f*}) \leq a^{f*} \ . \tag{2}$$

Furthermore, we get laws similar to the standard Kleene star.

Lemma 3.3

1. $\prod_{j=0}^{n} f^j(a) \leq a^{f*}$ *for all* $n \in \mathbb{N}$,
2. $a^{f*} \leq a^{f*} \cdot a^{f*}$,
3. $a \leq 1 \Rightarrow a^{f*} = 1$,
4. $a \leq b \Rightarrow a^{f*} \leq b^{f*}$,
5. $a^{f*} \leq (a^{f*})^{f*}$.

Proof.

1. By (1) we get $\prod_{j=0}^{n-1} f^j(a) \leq x_n$ and $x_{n-1} \leq x_n$. Then the claim follows by induction.
2. By (2) and isotony, we get $1 \leq a^{f*} \Rightarrow a^{f*} \leq a^{f*} \cdot a^{f*}$.
3. By lattice algebra, homomorphism and $(f2)$, we get $a \leq 1 \Leftrightarrow 1 + a \leq 1 \Leftrightarrow 1 + a \cdot f(1) \leq 1 \Rightarrow a^{f*} \leq 1$. The other direction is by (2).
4. By $(f2)$, assumption, isotony and $(f1)$ $a^{f*} \leq b^{f*} \Leftarrow 1 + a \cdot f(b^{f*}) \leq b^{f*} \Leftarrow 1 + b \cdot f(b^{f*}) \leq b^{f*} \Leftrightarrow \mathsf{true}$.
5. By 4., homomorphism and (2) (twice), we get $a^{f*} \leq (a^{f*})^{f*} \Leftarrow a \leq a^{f*} \Leftrightarrow a \cdot f(1) \leq a^{f*} \Leftarrow a \cdot f(a^{f*}) \leq a^{f*} \Leftarrow \mathsf{true}$.

$\square$

Due to Example 2.2.1 the definition of ^{f*} is a generalisation of the standard Kleene star. Therefore we cannot expect that all properties of *, like leapfrog, hold for it. More precisely, we have

Lemma 3.4

1. $a^{f*} \cdot a^{f*} \not\leq a^{f*}$.
2. $(a^{f*})^{f*} \not\leq a^{f*}$.
3. $a^{f*} \cdot (b \cdot a^{f*})^{f*} \not\leq (a + b)^{f*}$ *and* $(a + b)^{f*} \not\leq a^{f*} \cdot (b \cdot a^{f*})^{f*}$.
4. $a \cdot (b \cdot a)^{f*} \not\leq (a \cdot b)^{f*} \cdot a$ *and* $(a \cdot b)^{f*} \cdot a \not\leq a \cdot (b \cdot a)^{f*}$.

The proof is straightforward by choosing explicit elements. E.g., $a \cdot a \leq a^{f*} \cdot a^{f*}$ and $a \cdot a \not\leq a^{f*}$ implies the first item.

Since the function f often simulates physical behaviours, it will explicitly occur in algebraic expressions. Therefore we briefly discuss the interaction between f and ^{f*}.

Lemma 3.5 *Assume a f-generated Kleene algebra. Then $f(a)^{f*} \leq f(a^{f*})$. If f is even universally disjunctive, then*

$$f(a)^{f*} = f(a^{f*}).$$

Proof. The first claim ($f(a)^{f*} \leq f(a^{f*})$) is immediate by induction $(f2)$ and the assumed properties of f. Using μ-fusion together with $a^{f*} = \mu_x(1 + a \cdot f(x))$ and $f(a)^{f*} = \mu_x(1 + f(a) \cdot f(x))$, we get

$$f(1 + a \cdot f(x)) \leq 1 + f(a) \cdot f(f(x)) \Rightarrow f(a^{f*}) \leq a^{f*}.$$

The antecedent holds since f is a homomorphism. $\square$

4 Conclusion and Outlook

Sumarising, we have showed that the Kleene star is not the best in some situations. Therefore we introduced an f-generated Kleene star, which modifies the iterated element in each step. The new operator can be used e.g. in describing physical behaviours like a pendulum. We have also presented some basic properties of ^{f*}.

Since this research is still ongoing, there is a lot of further work. First it will be interesting to find more properties of the homomorphism f and the f-generated Kleene star. Especially the interaction of both should be discussed in more detail. Also the connection to functional programming (Example 2.2.2) will help to find more useful properties. This will lead to a better understanding of f-generated Kleene algebras.

In the paper we showed how to weaken the finite iteration operator of a Kleene algebra. But in the same way it should easily be possible to weaken the finite iteration of lazy Kleene algebra [6], the infinite iteration operator of omega algebras [2, 6] and the strong iteration of refinement algebra [7].

Additionally, as already mentioned, we give an algebra for hybrid systems, which is based on lazy Kleene algebra [4]. Using this approach the homomorphism f can be interpreted as changing behaviours in (continuous) time. Therefore the operator ^{f*} fits well in the development, specification and analysis of hybrid systems. As a first step, a case study will be done.

Acknowledgements. I am very grateful to Bernhard Möller and Kim Solin for valuable and fruitful discussions.

References

1. R. Bird. Lectures on constructive functional programming. In B. Manfred, editor, *Constructive Methods in Computing*, Science, volume F55 of NATO ASI Series, pages 151–216, 1989.
2. E. Cohen. Seperation and Reduction. In R. Backhouse and J. Olivera, editors, *Mathematics of Program Construction*, Lecture Notes in Computer Science 1837, pages 45–59, 2000.
3. J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
4. P. Höfner and B. Möller. Towards an Algebra of Hybrid Systems. In W. MacCaull, M. Winter, and I. Duentsch, editors, *Relational Methods in Computer Science*, Lecture Notes in Computer Science 3929, pages 121–133, 2006. (in press).
5. D. Kozen. Kleene Algebra with Tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.
6. B. Möller. Kleene getting lazy. *Science of Computer Programming, Special issue on MPC 2004*, 2006. (to appear)
   Previous version: B. Möller: Lazy Kleene algebra. In D. Kozen (ed.): Mathematics of program construction. LNCS 3125. Springer 2004, 252–273.
7. J. von Wright. From Kleene Algebra to Refinement Algebra. In E. Boiten and B. Möller, editors, *Proc. of 6th Int. Conf. on Mathematics of Program Construction, MPC 2002*, Lecture Notes in Computer Science 2386, pages 233–262, 2002.

Nomadic Time
(Extended Abstract)

Andrew Hughes[1]

Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK.
e-mail: a.hughes@dcs.shef.ac.uk

1 Introduction

CCS [1] is commonly used for modelling synchronous communication between two processes, where one sends a signal and the other receives it at the same time (a concept referred to as *local synchronization*). However, it cannot directly represent systems involving synchronization of a sender with an arbitrary number of recipient processes (known as *global synchronization*) in a *compositional* manner. Crucially, the semantics of a broadcast agent cannot suitably be represented using CCS. If the agent is defined as transmitting a signal to each of the recipients sequentially, through multiple local synchronizations, then its semantics will become non-compositional, because such behaviour depends upon the number of recipients. Each time a new recipient is introduced, or one of the existing ones is removed, the semantics will have to be changed.

A solution to this deficiency lies in providing a way of determining when all possible synchronizations have taken place. With this facility available, the broadcast agent can recurse, transmitting signals, until this condition holds. The family of abstract timed process calculi (including TPL[2] and CaSE[3]) allow this by extending CCS with *abstract clocks*. These don't represent real time, with units such as minutes and seconds, but are instead used to form synchronous cycles of internal actions followed by clock ticks. A concept known as *maximal progress* enforces the precedence of internal actions over clock ticks, allowing the possible synchronizations to be monitored. When a synchronization takes place, it appears to the system as an internal action. Thus, with maximal progress, synchronizations prevent the clock from ticking, and a result, the occurrence of a clock tick also indicates that there are no possible synchronizations.

However, the timed calculi mentioned above lack any notion of distribution or mobility. Thus, while they can adequately represent large static systems, involving both local and global synchronization, they fail to model more mobile systems, where the location of a process can change during execution. In contrast, the ambient calculus [4] includes both distribution (via structures known as *ambients*) and mobility (by allowing these structures to be moved, along with their constituent processes, during execution). But, it suffers from similar deficiencies to CCS when modelling global synchronization.

This extended abstract presents the calculus of *Typed Nomadic Time* (TNT), which combines the abstract timed calculus, CaSE, with notions of distribution and mobility from the ambient calculus and its variants ([5,6]). This allows the creation of a compositional semantics for mobile component-based systems, which utilise the notion of communication between arbitrary numbers of processes within a mobile framework. To extend the example of a broadcast agent given above, this extension allow broadcasts to be localised to a particular group of processes, which can change during execution. Section 2 provides a simple example, illustrating the use of the calculus, while section 3 concludes with a discussion of future work.

2 A Simple Example

Consider the familiar children's game of musical chairs. The conduct of the game can be divided into the following stages:

1. The players begin the game standing. The number of players is initially equal to the number of chairs.
2. The music starts.
3. A chair is removed from the game.
4. The music stops.
5. Each player attempts to obtain a chair.
6. Players that fail to obtain a chair are out of the game.
7. The music restarts. Any players who are still in the game leave their chairs and the next round begins (from stage three).

The winner is the last player left in the game. A model of this game can be created using the TNT process calculus.

The game environment is represented using named locations (commonly known as *localities* in the literature). These localities can be nested within each other and form a forest structure (due to the possibility of multiple localities occurring at the top level). In the musical chairs scenario, each chair is represented by a locality, as is the 'sin bin', to which players are moved when they are no longer in the game. These localities are all nested inside a further locality which represents the room itself. This is not a necessity, but makes for a cleaner solution; it allows multiple instances of the system to be nested inside some larger system, each performing its own internal interactions and entering into the synchronization cycle of the larger system.

The locality structure is represented in the calculus by the expression shown below. The *room* locality contains multiple *chair* localities, each of which contains $\mathbf{0}$, a process with no explicit behaviour[1]. The $|$ operator connecting the chair localities denotes parallel composition; each locality and its constituent processes runs concurrently. CB and the σ and ω symbols will be explained shortly.

$$room[chair[\mathbf{0}]_{\emptyset}^{CB} \mid chair[\mathbf{0}]_{\emptyset}^{CB}]_{\{\sigma\}}^{\omega}. \tag{1}$$

[1] It does exhibit contextual behaviour, due to transitions created by clock ticks.

The players themselves are represented by *processes*. This allows them both to interact and to move between localities. A gamesmaster process is also introduced. This doesn't play an active role in the game itself, but is instead responsible for performing the administrative duties of removing chairs from the game and controlling player movement. The process definitions are summarised in Table 1, along with the derived syntax used in this example.

Table 1. Summary of Processes and Derived Syntax for Musical Chairs

$$\omega \stackrel{\text{def}}{=} \mu X.(\overline{in}.X + \overline{out}.X + \overline{open}.X) \tag{2}$$

$$\sigma.P \stackrel{\text{def}}{=} \lceil \mathbf{0} \rceil \sigma(P) \tag{3}$$

$$CB \stackrel{\text{def}}{=} \mu X.(\overline{in}.\overline{out}.X + \overline{open}) \tag{4}$$

$$SB \stackrel{\text{def}}{=} \mu X.\overline{in}.X \tag{5}$$

$$GM2 \stackrel{\text{def}}{=} \sigma.GM3 \tag{6}$$

$$GM3 \stackrel{\text{def}}{=} open\ chair.GM5 \tag{7}$$

$$GM5 \stackrel{\text{def}}{=} \mu X.(\lceil in\ chair\ sit.X \rceil \sigma(GM6)) \tag{8}$$

$$GM6 \stackrel{\text{def}}{=} \mu X.(\lceil in\ sinbin\ leave.X \rceil \sigma(GM2)) \tag{9}$$

$$Player \stackrel{\text{def}}{=} \lceil sit.PInChair \rceil \sigma(Loser) \tag{10}$$

$$PInChair \stackrel{\text{def}}{=} \sigma.(out\ chair\ stand.0 | stand.Player) \tag{11}$$

$$Loser \stackrel{\text{def}}{=} leave.0 \tag{12}$$

The presence of music is signified by the ticks of a clock σ. A tick from σ is also used to represent the implicit acknowledgement that everyone who can obtain a chair has done so, and that the remaining player left in the room has lost. σ appears as part of a set of clocks on the bottom right of the locality definition to signify that its ticks are visible within the locality (including any nested localities), but not outside. Instead, ticks appear as silent actions outside the location boundaries.

The top right of a locality is used to specify a further property of the locality, the *bouncer*. This is essentially a process with a very limited choice of available actions. It has no real behaviour of its own, but instead performs the job of managing the locality. It dictates whether processes or other localities may enter or exit the locality, and whether the locality may be destroyed by a process in the parent locality. Within the musical chairs model, such protection is irrelevant for the room itself (a bouncer, ω (2), is used which ensures that all possible movements are allowed), but is essential for the chairs (4) and the sin bin (5).

It is the chair bouncer that enforces the implicit predicate that only one player may inhabit a chair at any one time, while the sin bin bouncer prevents players leaving the sin bin once in there.

To model stage one of the game, n player processes and n chair locations are placed in the room. The advantage of using TNT for this model is that the actual number of players or chairs is irrelevant. They only have to be equal at the start to accurately model the game. The calculus allows the creation of a compositional semantics, as discussed in section 1, which work with any n.

For the purposes of demonstration, n is assumed to be two to give the following starting state:

$$room[chair[\mathbf{0}]^{CB}_{\emptyset} \mid chair[\mathbf{0}]^{CB}_{\emptyset} \mid \sigma.\sigma.Player \mid \sigma.\sigma.Player \mid GM2]^{\omega}_{\sigma}. \tag{13}$$

The room and chairs appear as shown earlier. The processes of the form $\sigma.\sigma.Player$ simply wait until two clock cycles have passed, the end of each being signalled by a tick from σ. The intermittent period between the ticks (the second clock cycle) represents the playing of the music. This syntactic form, denoted more generally by $\sigma.P$ (P being some arbitrary process), is derived from the core syntax of TNT as shown in (3). Like most of the model, it relies on the stable timeout operator, $\lceil E \rceil \sigma(F)$, where F acts if E times out on the clock, σ. In this case, E, being $\mathbf{0}$, will always time out as it has no actions to perform.

The gamesmaster ($GM2$ (6)) also waits for the first clock tick (the music starting), but then evolves to $GM3$ (7) and uses the second cycle, prior to the music stopping, to remove a chair from the game. Maximal progress, as explained in section 1, ensures that this occurs before the next clock tick, as the removal emits a silent action.

The most interesting part of the model lies in the interaction with the chairs, which forms part of stages five to seven. The aim of stage five is to get as many player processes as possible inside chair localities. This is handled by again relying on maximal progress to essentially perform a form of broadcast that centres on mobile actions. Rather than sending a signal to a number of recipients, a request to move into a chair (see (8) and (10)) is delivered instead.

If a chair is available, then a player process will enter it (the actual chair and player chosen is non-deterministic). This will cause an internal action to occur, as illustrated by (14), and this will take precedence over the clock tick. Thus, when the clock eventually does tick, it is clear that no more players can enter chairs. Using clocks in this manner makes the system *compositional*; in contrast to other models, players and chairs can be added without requiring changes to the process definitions.

$$\begin{aligned} & GM5 \mid Player \mid chair[\mathbf{0}]^{CB}_{\emptyset} \\ \xrightarrow{\tau} \; & GM5 \mid chair[\mathbf{0} \mid PInChair]^{\overline{out}.CB}_{\emptyset} \end{aligned} \tag{14}$$

Stages six and seven proceed in a similar way. Stage six sees essentially the same broadcasting behaviour applied to the losing players (see (9) and (12)).

The difference is that stage six demonstrates something which wouldn't be possible without mobility: the broadcast is limited to those player processes which remain in the room. Communication between processes in different localities is disallowed in TNT, causing an implicit scoping of the broadcast. The broadcast is again terminated by a tick from σ, which, in this case, also signifies the music starting up again. The remaining players leave their chairs (11), and the system essentially returns to stage three, with $n-1$ chairs and $n-1$ players.

3 Conclusions and Future Work

This extended abstract outlines a calculus which provides a novel combination of features, allowing arbitrary numbers of agents both to synchronize with other agents and move around a dynamic topology, constructed from nested localities. Current work on this calculus focuses on the formalisation of an operational semantics and the creation of a type system to allow additional validity and security checks to be performed. The existing equivalence theory for CaSE will also require extension in order to encompass the new features found in TNT. In the longer term, further case studies will be considered, which go beyond the simple example presented here. In particular, the modelling of quorum sensing bacteria is of interest.

Acknowledgements

This work is supported by a grant from the Engineering and Physical Sciences Research Council (EPSRC). I would also like to thank my supervisor, Mike Stannett, as well as Simon Foster and Georg Struth, for their insightful discussions and support.

References

1. Milner, R.: Communication and Concurrency. Prentice-Hall, London (1989)
2. Hennessy, M., Regan, T.: A process algebra for timed systems. Information and Computation **117**(2) (1995) 221–239
3. Norton, B., Lüttgen, G., Mendler, M.: A compositional semantic theory for synchronous component-based design. In: Proceedings of the 14th International Conference on Concurrency Theory (CONCUR '03). Number 2761 in Lecture Notes in Computer Science, Springer-Verlag (2003) 461–476
4. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98). Volume 1378 of Lecture Notes in Computer Science., Springer-Verlag (1998) 140–155
5. Levi, F., Sangiorgi, D.: Mobile safe ambients. ACM Transactions on Programming Languages and Systems (TOPLAS) **25**(1) (2003) 1–69
6. Teller, D., Zimmer, P., Hirschkoff, D.: Using ambients to control resources. In Brim, L., Janar, P., Ketinsky, M., Kuera, A., eds.: Proceedings of the 13th International Conference on Concurrency Theory (CONCUR '02). Number 2421 in Lecture Notes in Computer Science, Springer-Verlag (2002) 288–303

Combining Relational Methods and Evolutionary Algorithms

Britta Kehden

Christian-Albrechts University of Kiel, 24098 Kiel, Germany
bk@informatik.uni-kiel.de

Abstract. We take a relation-algebraic view on the formulation of evolutionary algorithms in discrete search spaces. We show how individuals and populations can be represented as relations and how important moduls of evolutionary algorithms can be implemented using relational algebra. For many important problems, the evaluation of a population with respect to certain constraints is the most costly step in one generation of an evolutionary algorithm. We show that the evaluation process for a given population can be sped up by using relational methods.

1 Introduction

Evolutionary algorithms (EAs) have become quite popular in solving problems from combinatorial optimization in the recent years. The representation of possible solutions for a given problem has been widely discussed (see e.g. [1] and [2]) We study the question whether representations using relations can be useful. Another impotant issue in the area of evolutionary computation is hybridization, where one combines evolutionary algorithms with other approaches in order to get better results. We think that it may be useful to combine evolutionary algorithms with relational methods. A first step intop this direction was made in [3]. We consider evolutionary algorithms for the search space $\{0, 1\}^n$ and examine how the most important modules of an evolutionary algorithm can be implemented on the basis of relational operations. Relational algebra has been widely used in computer science. Especially in the case of NP-hard combinatorial optimization problems on graphs, a lot of algorithms have been developed. Relational algebra has a small, but efficiently to implement, set of operations and it allows a formal development of algorithms and expressions starting usually with a predicate logic description of the problem.

We represent a population, which is a set of search points, as one single relation and evaluate this population using relational algebra. It turns out that this approach can be implemented in a way that mainly relies on the relation-algebraic formulation of the specific modules. Considering the evaluation of a given population we show that this process can be made more efficient using relational algebra. After giving a brief introduction into evolutionary algorithms and relational algebra in sections 2 and 3, we discuss the relation-algebraic formulation of important modules of evolutionary algorithms in Section 4. After that we consider three well-known NP-hard combinatorial optimization problems, namely minimum vertex covers, maximum cliques, and maximum independent sets, and show how the whole population can be evaluated using relational algebra. It turns out that using this approach can reduce the runtime from $\Theta(n^3)$ to $O(n^{2.376})$ for a population of size n compared with a standard approach. A more detailed presentation of these results can be found in [4].

2 Evolutionary Algorithms

Evolutionary Algorithms (see e.g [5]) are randomized search heuristics that follow Darwin's principle of evolution, the survival of the fittest. Given a fitness function to be maximized (or minmized), a set of search points, called population, is evolved w.r.t the function until a stopping criterium is fulfilled. For example the algorithm stops after a given number of iterations or if the best individual in the population has not been improved for a certain number of generations. In each step, a parent population randomly generates an offspring population by applying different variation operators. Then a subset of individuals of both populations is selected for the next parent generation, so that the fitness of the population is increased in each step. Possible variation operators are mutation, where each parent individual generates one child, and crossover, where two individuals of the parent population create one child. The selection of the individuals for the next generation can be done in different ways. One can choose a subset of a certain size consisting the best individuals of both populations, i.e, the elements with the highest fitness values, or build tournaments of each one parent and one child and choose the better one for the next generation.

3 Representing populations as relations

We consider evolutionary algorithms working in the search space $\{0,1\}^n$. We want to represent each population as a relation P where each individual of P is stored in one single column. As we want to show how to use relational algebra in an evolutionary algorithm we have to start with some basic definitions. For a more detailed description of relational algebra see [6]. We write $R : X \leftrightarrow Y$ if R is a relation with domain X and range Y, i.e. a subset of $X \times Y$. In the case of finite carrier sets, we may consider a relation as a Boolean matrix. Since this Boolean matrix interpretation is well suited for many purposes, we often use matrix terminology and matrix notation in the following. Especially, we speak of the rows, columns and entries of R and write R_{xy} instead of $(x, y) \in R$. The basic operations on relations are $R^\top$ (transposition), $\overline{R}$ (negation), $R \cup S$ (union), $R \cap S$ (intersection), RS (composition), the special relations O (empty relation), L (universal relation), and I (identity relation). A relation $v : X \leftrightarrow \mathbf{1}$ is called vector, where $\mathbf{1} = \{\perp\}$ is a specific singleton set. We omit in such cases the second subscript, i.e. write v_i instead of $v_{i\perp}$. Such a vector can be considered as a Boolean matrix with exactly one column and describes the subset $\{x \in X : v_x\}$ of X. Note, that one search point of the considered search space can be represented as a vector of length n. A set of k subsets of X can be represented as a relation $P : X \leftrightarrow [1..k]$ with k columns. For $i \in [1..k]$ let $P^{(i)}$ be the i-th column of P. More formally, every column $P^{(i)}$ is a vector of the type $X \leftrightarrow \mathbf{1}$ with $P^{(i)}_x \iff P_{xi}$. We assume that we are always working with populations that have exactly n individuals, i.e., the relation P has exactly

n rows and n columns. Under the assumption that we work with $n \times n$ relations, the operations transposition, negation, union and intersection can be implemented in time $O(n^2)$. The standard implementation for the composition needs time $\Theta(n^3)$. Using the algorithm proposed by Coppersmith and Winograd (see [7]) for the multiplication of two $n \times n$ matrices we can reduce the runtime for the composition to $O(n^{2.376})$.

4 Relation-algebraic formulation of important modules

Variation operators are important to construct new solutions for a given problem. We assume that the current population is represented by a relation P and present a relation-algebraic formulation for some well-known variation-operators. In addition we formulate an important selection method based on relational algebra. It turns out that the runtimes for our general framework are of the same magnitude as in a standard approach.

4.1 Mutation

An evolutionary algorithm that uses only mutation as variation operator usually flips each bit of each individual with a certain probability p. To model the mutation operator with relational algebra, we assume that we have constructed a relation M randomly, that gives the mask which entries are flipped in the next step. In this case each entry of M is set to 1 with probability p. Then we can construct the relation C for the children of P using the symmetric difference of P and M.

$$C = (P \cap \overline{M}) \cup (\overline{P} \cap M).$$

4.2 Crossover

A crossover operator for the current population P takes two individuals of P to produce one child. To create the population of children C by this process, we assume that we have in addition created a relation P' by permuting the columns of P. Then we can decide which entry to use for the relation C by using a mask M.

$$C = (M \cap P) \cup (\overline{M} \cap P')$$

To implement different crossover operators we have to use different masks in this expression.

4.3 Selection

We focus on tournament selection, and assume that we have a parent population P and a child population C both of size n. To establish n tournaments of size 2 we use a random bijective mapping that assigns each individual of P to an individual of C. This can be done by permuting the columns of C randomly. Due to the evaluation process we assume that we have a decision vector d that tells us to take the individual of P or the individual of C for the new population N. Let $P, C : X \leftrightarrow [1..n]$ and $d : [1..n] \leftrightarrow \mathbf{1}$, where we assume that the columns of C have already been permuted randomly. We want to construct a new population N, such that for each $i \in [1..n]$ either $P^{(i)}$ or $C^{(i)}$ is the i-th column of N. The vector d specifies which columns should be adopted in the new population N.

$$d_i \iff P^{(i)} \text{ should be adopted} \quad \text{and} \quad \overline{d}_i \iff C^{(i)} \text{ should be adopted.}$$

The new population N is determined by

$$N = (P \cap \mathsf{L}d^\top) \cup (C \cap \overline{\mathsf{L}d^\top}).$$

It is easy to see that the presented determinations can be done in time $O(n^2)$, which is the same magnitude as in a standard approach. In the following section we will give an example to show how the decision vector d can be determined.

5 Testing properties of solutions for some graph problems

Assume that we have a relation P that represents a population. One important issue is to test which of the individuals of the population fulfill given constraints which means that they are feasible solutions. Given a graph $G = (V, E)$ with n vertices represented as an adjacency relation R we want to test each individual to fulfill a given property. We concentrate on the constraints for some well-known combinatorial optimization problems for graphs, namely minimum vertex covers, maximum cliques, and maximum independent sets. A vertex cover of a given graph is a set of vertices $V' \subseteq V$ such that $e \cap V' \neq \emptyset$ holds for each $e \in E$. For a clique $C \subseteq V$ the property that R_{uv} for all $u, v \in C$ with $u \neq v$ has to be fulfilled and in an independent set $I \subseteq V$, $\overline{R}_{uv}$ has to hold for all $u, v \in I$ with $u \neq v$. It is well known that computing a vertex cover of minimum cardinality or cliques and independent sets of maximal cadinality are NP-hard optimization problems (see e.g.[8]). It is easy to see that it affords a runtime of $\Theta(n^2)$ to test whether one individual fulfills one of the stated properties with a standard approach. Working with a population of size n this means that we need time $\Theta(n^3)$ for evaluating each of these properties. We want to show that the runtime for evaluating a population that is represented as a relation can be substantially smaller using relation-algebraic expressions. Note, that the size of the solutions, which means the number of ones in the associated column, can be determined for the whole population P in time $O(n^2)$. Therefore, the most costly part of the evaluation process for the three mentioned problems seems to be the test whether the given constraints are fulfilled. Given the two relations R and P we can compute a vector that marks all individuals of the population that are vertex covers. The $i-th$ column in P represents a vertex cover, if the following condition holds.

$$\forall\, u, v : R_{uv} \rightarrow (P_u^{(i)} \vee P_v^{(i)}).$$

This predicate logic expression can be transformed into a relation-algebraic expression and we achieve the vector

$$\overline{\mathsf{L}(R\overline{P} \cap \overline{P})}^\top$$

that specifies all vertex cover in P. For the case of independent sets we can use the fact that $P^{(i)}$ is an independent set iff $\overline{P}^{(i)}$ is a vertex cover. We obtain the vector

$$\overline{\mathsf{L}(RP \cap P)}^\top$$

that represents all independent sets in the population. Since a set of vertices is a clique of G if and only if it is an independent set of the complement graph with adjacency relation $\overline{R} \cap \overline{\mathsf{I}}$, we can determine the vector

$$\overline{\mathsf{L}((\overline{\mathsf{I} \cup R})P \cap P)}^{\top}$$

that specifies all columns of P that represent cliques of R. Considering the different expressions, the most costly operation that has to be performed is the composition of two $n \times n$ relations. Therefore the evaluation process for a given population P and a relation R can be implemented in time $O(n^{2.376})$ by adapting the algorithm of Coppersmith and Winograd (see [7]) for the multiplication of two $n \times n$ matrices to relations, which beats the lower bound of $\Omega(n^3)$ for the standard implementation.

6 Conclusions

We have taken a relation-algebraic view on evolutionary algorithms for some graph problems. It turns out that the evaluation of a population can be sped up by using relation-algebraic expressions to test whether the solutions of the population fulfill given constraints. In the case of the three considered graph problems the computation time for one generation can be reduced from $\Theta(n^3)$ to $O(n^{2.376})$.

References

1. Michalewicz, Z. (2004). How to solve it: Modern heuristics. 2nd edition, Springer-Verlag, Berlin.
2. Raidl, G.R. and Julstrom, B.A. (2003). Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans. on Evolutionary Computation 7, 225–239.
3. Kehden B., Neumann F., Berghammer R. (2005): Relational Implementation of Simple Parallel Evolutionary Algorithms In: Proc. of the 8th International Conference on Relational Methods in Computer Science (RelMiCS 8), LNCS 3929, Springer, Berlin, Germany
4. Kehden, B., Neumann F. (2006): A Relation-Algebraic View on Evolutionary Algorithms for Some Graph Problems In: Gottlieb and Raidl (Eds.): EvoCop 2006, LNCS 3906, Springer, Berlin, pages 147 - 158.
5. Eiben, A.E., Smith, J.E.(2003). Introduction to Evolutionary Computing. Springer
6. Schmidt, G., and Ströhlein, T. (1993). Relations and graphs. Discrete mathematics for computer scientists, EATCS Monographs on Theoret. Comp. Sci., Springer.
7. Coppersmith, D., and Winograd, S. (1990). Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251–280.
8. Garey, M. R., Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, New York.

A topographical analysis of event structures

José Juan Palacios Pérez*

The University of Manchester

Abstract In the thesis [PP06] we introduce the notion of a *topographical space* to carry out an analysis of *event structures*, which is a model for concurrent computation. We show how the notion of a topographical space produces a clear and uniform presentation of several kinds of event structures and related structures in the literature.

1 Introduction

In the field of mathematical models of concurrent computation, the model known as an *event structure* has a distinguished rôle. Introduced by Nielsen, Plotkin and Winskel in [NWP81] and later extended by Winskel in [Win85,Win89], event structures are based in the following basic ideas.

1. The behaviour of a concurrent process is expressed by the *occurrence of events*. An event is an atomic entity, on the same level of abstraction as that of a point in geometry.
2. There is a notion of *consistency* among events, that is, whether it is the case that two (or more) events can occur without any conflict. The notion of consistency is related to the notion of non-determinism which pervades concurrent computation.
3. The occurrence of an event may *causally depend* on the occurrence of previous events. Such a relation of causal dependency can be given by a preorder on events or something more elaborated which is sometimes called an *enabling relation*.

A *state* or *configuration* of a concurrent system is defined as a coherent set of (the occurrence of) events which is closed under the causal dependency relation. If two events inside a configuration are independent of each other, then they can occur simultaneously (that is, in parallel).

A number of relations between event structures and several models of concurrency, such as Petri nets, labelled transition systems and trace languages has been accomplished [WN95]. Event structures first appeared about 25 years ago. Since then they have re-appeared in various forms, sometimes using different terminology, and sometimes with the same terminology meaning something

* Thanks to Dr. Andrea Schalk my supervisor, and Dr. Harold Simmons, my adviser for their guidance and support.

different. Part of the aim of the thesis is to clarify and generalise the relationships between these various notions, hence providing a clearer and more uniform account of these various patterns.

First, we review the original definition of event structures and point out the importance of the family of configurations. Then, we introduce the notion of topographical spaces and describe some properties as well as different kinds of topographies, each one associated with a particular class of event structure.

2 Topographies from configurations

The following definition is taken from [Win86].

Definition 1. *An event structure* $\boldsymbol{S}$ *is a triple* $(S, \mathcal{S}_\bullet, \vdash)$ *which consists of*

- *a set* S *of events, its carrier,*
- *a non-empty family* $\mathcal{S}_\bullet$ *of finite subsets of* S*, called the consistency predicate, which satisfies being downwards closed under inclusion, that is if* $X \in \mathcal{S}_\bullet$ *and* $Y \subseteq X$ *then* $Y \in \mathcal{S}_\bullet$*,*
- *an enabling relation* $\vdash \subseteq \mathcal{S}_\bullet \times S$*, which is upwards closed under inclusion, that is if* $Y \vdash s$ *and* $Y \subseteq X$ *then* $X \vdash s$ *(for* $X, Y \subseteq_f S$*).*

Example 1. Consider the event structure $\boldsymbol{S}_1 = (S, \mathcal{S}_\bullet, \vdash)$ with carrier $S = \{a, b, c, d\}$, consistency predicate given by $\mathcal{S}_\bullet = \mathcal{P}_f - \{\{a, b\}, \{a, b, c\}, \{a, b, c, d\}, \{a, b, d\}\}$ and $\vdash$ given by $\emptyset \vdash a, b, d$, $\{a\} \vdash c$, and $\{b\} \vdash c$. The first sentence of the enabling says that events a, b, d are independent each other, the second sentence says that event c depends on the occurrence of either event a or b[1]. Note that the events a, b are incompatible each other.

In Figure 1 we show the family of configurations $(\mathfrak{S}, \subseteq)$ of $\boldsymbol{S}_1$, which forms a partial order under inclusion.

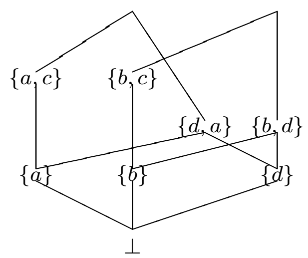


Figure1. Family of configurations of the event structure $\boldsymbol{S}_1$.

It is precisely the family of configurations of an event structure that models a concurrent system. Such family of configurations forms a special kind of domain

[1] Nothing forbid us to have something like $\{a, d, c\} \vdash c$ or $\{b, d\} \vdash b$.

(that is, a poset with certain completeness properties), and is naturally described in terms of topographical spaces.

Definition 2. *Let S be a set. A* topography *on S is a family $\mathfrak{S}$ of subsets with the following properties.*

(i) $\emptyset \in \mathfrak{S}$ *(ii)* $\bigcup \mathfrak{S} = S$

(iii) For each subfamily $\mathcal{X} \subseteq \mathfrak{S}$ which is locally bounded[2] *in $\mathfrak{S}$, we have $\bigcup \mathcal{X} \in \mathfrak{S}$.*

A topographical space *$(S, \mathfrak{S})$ is a set S equipped with a topography $\mathfrak{S}$. A* region *of the space is a set $X \in \mathfrak{S}$. For each region $X \in \mathfrak{S}$ the family $\mathfrak{S}X = \{Y \in \mathfrak{S} \mid Y \subseteq X\}$ is the* down family below *X* in *$\mathfrak{S}$.*

Indeed, a region of a topographical space captures a configuration of an event structure. It is easy to show that the poset $(\mathfrak{S}, \subseteq)$ associated to the family of configurations of the event structure from Example 1 is a topographical space. As the name suggest, this is related to the standard notion of a topological space, but more suited to our investigation[3].

A topographical space $(S, \mathfrak{S})$ is locally topological if for all regions $X \in \mathfrak{S}$, we have $Y, Z \in \mathfrak{S}X \Longrightarrow Y \cap Z \in \mathfrak{S}X$. In particular, the space is locally Alexandroff if for each region $X \in \mathfrak{S}$ the family $\mathfrak{S}X$ is closed under arbitrary intersections. The space is locally discrete if $\mathfrak{S}X = \mathcal{P}X$.

Each topographical space induces two kinds of comparison, a global and a local comparison. Both are important in describing properties among and inside regions, respectively.

Each topography $\mathfrak{S}$ has a associated a canonical family, its cover given by $X \in C(\mathfrak{S}) \iff (\forall Y \subseteq_f X)(\exists Z \in \mathfrak{S})[Y \subseteq Z]$ (for $X \subseteq S$). Such a family is the smallest topography that contains $\mathfrak{S}$ and is downwards closed under inclusion.

In the following section we consider

Capital Country Commonwealth

spaces, each of which generates a topography which is at least locally topological.

3 From capitals to commonwealth

We think of $\mathcal{S}_\bullet$ as a consistency predicate *in the small*, since it deals only with finite sets. In contrast, a consistency predicate *in the large* deals with arbitrary sets. Hence the name capital.

Definition 3. *A* Consistency Predicate in the Large (capital) *on a set S is a family $\mathcal{S}$ of subsets of S with the following properties.*

(i) $\emptyset \in \mathcal{S}$ *(ii)* $\bigcup \mathcal{S} = S$

[2] In any poset $(S, \leq)$ a subset $X \subseteq S$ is locally bounded iff every finite set $Y \subseteq_f X$ is bounded in S.

[3] In fact, each topological space gives an example of a topographical space.

(iii) For each subfamily $\mathcal{X} \subseteq \mathcal{S}$ which is directed, we have $\bigcup \mathcal{X} \in \mathcal{S}$.
(iv) The family $\mathcal{S}$ is downwards closed under inclusion, that is $Y \subseteq X \in \mathcal{S} \Longrightarrow Y \in \mathcal{S}$ for subsets X, Y of S.

A capital space $(S, \mathcal{S})$ is a set S equipped with a capital $\mathcal{S}$.

It is immediate to show that each consistency predicate $\mathcal{S}_\bullet$ is isomorphic to its capital $\mathcal{S}$. Intuitively, a capital space can be seen as an event structure where all events are independent to each other, that is, the causality relation is just equality. Note that the cover $C(\mathfrak{S})$ of a topography $\mathfrak{S}$ forms a capital space $(S, C(\mathfrak{S}))$, and for any other capital $\mathcal{S}$ we have $C(\mathfrak{S}) \subseteq \mathcal{S}$. That is, the cover is the minimal capital associated to the parent topography.

Each capital space produces a locally discrete topography. Capital spaces have appeared in the literature as qualitative domains and coherence spaces (the latter using a binary conflict relation).

We now consider the case when the causality relation is given by a partial order.

Definition 4. *Let $(S, \leq)$ be a poset. A capital $\mathcal{S}$ on S is* compatible *with the carried comparison $\leq$ if for all $X \in \mathcal{P}S$ we have $X \in \mathcal{S} \Longrightarrow \downarrow X \in \mathcal{S}$. A* country space

$$(S, \mathcal{S}, \leq)$$

is a poset with a compatible capital.

The topography $\mathfrak{S}$ of a country space $(S, \mathcal{S}, \leq)$ is given as

$$\mathfrak{S} = \mathcal{S} \cap \mathcal{L}S$$

that is, each region $X \in \mathfrak{S}$ is a consistent lower section. Such topographies are locally separated, locally Alexandroff, the partial order $\leq$ coincides with the comparison induced by $\mathfrak{S}$, and $C(\mathfrak{S}) = \mathcal{S}$. For each region $X \in \mathfrak{S}$ we have $x \leq_X y \Longrightarrow x \leq y$ (for all $x, y \in S$). As consequence, for $Y \in \mathcal{L}_X X$ we have $Y = \downarrow Y \cap X$ for each region $X \in \mathfrak{S}$.

For practical purposes, one can assume that each event in a country space can occur once a number of events have occurred. A special kind of country spaces are Winskel's prime event structures $(S, \mathcal{S}_\bullet, \leq)$ [Win86] where the occurrence of each event is dependent only on a finite number of events.

In a country space each event depends on (at most) a set of events. We extend the causal dependency relation by allowing an event to depend of several sets. For this we replace the partial order of a country space by a tree.

Definition 5. *(a) Let S be a set, let $\mathcal{S}$ be a capital on S and let $\mathbb{S}$ be a tree over S. We say $(\mathcal{S}, \mathbb{S})$ is a* compatible pair *if*

$$\boldsymbol{u} \in \mathbb{S} \Longrightarrow \lfloor \boldsymbol{u} \rfloor \in \mathcal{S}_\bullet$$

That is, if for each node of $\mathbb{S}$ its underlying set is in the finite part of the capital.

(b) A commonwealth space *is a structure*

$$\boldsymbol{S} = (S, \mathcal{S}, \mathbb{S})$$

where $(S, \mathcal{S})$ *is a capital space and* $\mathbb{S}$ *is a tree over* S *such that the pair* $(\mathcal{S}, \mathbb{S})$ *is compatible.*

From an event structure $\boldsymbol{S} = (S, \mathcal{S}_\bullet, \vdash)$ it is easy to obtain a commonwealth $\mathfrak{k}\boldsymbol{S} = (S, \mathcal{S}, \mathbb{S})$ by constructing the tree $\mathbb{S}$ over S as follows.

$$\begin{aligned} \mathbb{L}_0 &= \{\perp\} \\ \mathbb{L}_1 &= \{\perp s \mid \emptyset \vdash s\} \\ \mathbb{L}_{i+1} &= \{\boldsymbol{u}s \mid \boldsymbol{u} \in \mathbb{L}_i,\ \lfloor \boldsymbol{u} \rfloor \vdash s,\ s \notin \lfloor \boldsymbol{u} \rfloor\} \end{aligned}$$

for all $s \in S$. Each $\mathbb{L}_i$ denotes the tree $\mathbb{S}$ up to level i. The converse is also immediate.

We now define the topography $\mathfrak{S}$ induced by a commonwealth $\boldsymbol{S} = (S, \mathcal{S}, \mathbb{S})$. For each $s \in X \subseteq S$ we say that the membership $s \in X$ is witnessed by the node $\boldsymbol{u} \in \mathbb{S}$ if $s \in \lfloor \boldsymbol{u} \rfloor \subseteq X$.

The harvest $\mathcal{H}\mathbb{S}$ of $\mathbb{S}$ is the family

$$X \in \mathcal{H}\mathbb{S} \iff (\forall s \in X)(\exists \boldsymbol{u} \in \mathbb{S})[s \in \lfloor \boldsymbol{u} \rfloor \subseteq X]$$

for $X \subseteq S$.

The harvest $\mathcal{H}\mathbb{S}$ is indeed a topography on S, but is not necessarily closed under intersections. We define the topography $\mathfrak{S}$ of the commonwealth $\boldsymbol{S}$ as follows

$$\mathfrak{S} = \mathcal{S} \cap \mathcal{H}\mathbb{S}$$

which is the family of those $X \in \mathcal{S}$ such that each member $s \in X$ is witnessed by a node of $\mathbb{S}$. Such topography is locally separated with very finite character.

For any commonwealth $\boldsymbol{S} = (S, \mathcal{S}, \mathbb{S})$ with topography $\mathfrak{S}$ the pair $(C(\mathfrak{S}), \mathbb{S})$ is compatible and induces $\mathfrak{S}$.

References

[NWP81] Mogens Nielsen, Glynn Winskel, and Gordon Plotkin. Petri nets, Event structures and Domains, part 1. *Theoretical Computer Science*, 13:85–108, 1981.

[PP06] José Juan Palacios-Pérez. *The categorical analysis of Event structures*. PhD thesis, Computer Science, The University of Manchester, 2006.

[Win85] Glynn Winskel. Petri Nets, Algebras and Morphisms. Technical Report 79, Univ. of Cambridge Computer Laboratory, 1985.

[Win86] Glynn Winskel. Event structures : Lecture notes for the Advanced course on Petri nets. Technical Report UCAM-CL-TR-95, University of Cambridge, 1986.

[Win89] G. Winskel. An introduction to event structures. *Lecture Notes in Computer Science: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency.*, 354:364–397, 1989. NewsletterInfo: 34.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1995.

Relational Kleene Algebras and their compilation to modular applicative transducers

Benoît Razet

Université Paris 7 - INRIA Rocquencourt

1 Introduction

Finite-state methods are commonly used in computational linguistic and particularly at the morphology level. Gérard Huet[8] uses those structures to treat the sanskrit language segmentation problem which deals with the sandhi relation inversion. The method uses a technology based on annotated lexicons implemented in the *Zen* toolkit[6]. A more general transducer representation method is inspired from this application and leads the introduction of *aums*[7]. We plan to study potential algebraic operators, inspired from Kleene algebras, to describe rational relations useful in the natural language processing field.
In the following we suppose that the reader is familiar with rational languages or relations, automata, transducers and regular expressions.

2 Modular transducers

Let remind that rational sets or rational relations are completely described with Kleene algebra models. Basic Operators of the Kleene algebras like $\cdot$(concatenation), $+$(union) and $*$(Kleene's star) are often used to describe modular automata constructions. The automata constructions associated to thoses operations give no guaranty to produce a deterministic or mininimal automaton. Since determinization and miminization have exponentional costs, automata constructions must not do those operations too frequently.

An originality of *aums* for representing automata or transducers is that they are completely applicative data structures, loops are coded with a virtual address system. The recognition or transduction is done using a reactive process over *aums*. In [8] lexicons (finite sets of words) are coded as DAGs with maximal sharing using *aums* data structure. A reactive engine is introduced to perform the recognition of L^* for a lexicon L, and it uses continuations to perform the non-deterministic search. This engine is highly configurable since additional parameters could affect the search and then extensions can be made easily.

We have done one such extension for modular transducers[9]. Let now describe modular automata which is based on the *aum* technology, modular transducers are a direct extension from modular automata using decorated *aums*. Let consider a finite set of lexicons indexed L_i, one would like to modify the reactive engine to perform the recognition of the language defined by a regular expression over the L_i which is still a rational language. Regular expressions are elements of the following Kleene algebra :

$$\begin{array}{lc} \text{regexp} ::= & 1 \\ \quad\quad\quad\quad \mid & L_i \\ \quad\quad\quad\quad \mid & regexp + regexp \\ \quad\quad\quad\quad \mid & regexp \cdot regexp \\ \quad\quad\quad\quad \mid & regexp^* \end{array}$$

As *aums* can code with the maximal sharing for lexicons, we do not want to change those *aums* during the regular expression compilation. We choose to keep separated lexicons from the geometry over those lexicons, the geometry refering the regular expression. Let compile the regular expression into an automaton over a new phase alphabet, each phase linked to an *aum*, we now have modular automata description with two levels.
In the second version of the *Zen* toolkit, description of modular transducers is possible using system's of regular expressions. For example the definition of the sanskrit word morphology could be expressed that way :

```
INVAR  = prev.abso | unde
CONJUG = prev? . root
SUBST  = iic* .noun | iic+ .ifc
VERBAL = CONJUG | iiv.auxi
WORD   = SUBST | VERBAL | INVAR
```

`WORD`, `CONJUG`, `SUBST`, `VERBAL` and `INVAR` are names for equations. `iiv`, `auxi`, `noun`, `iic`, `ifc`, `prev`, `root`, `abso` and `unde` are lexicons for sanskrit lexical categories.

The way such regular expressions are compiled using the Berry-Sethi algorithm[2] into a modular transducers is described in [9]. In this article we introduce modular transducers incrementally using three various reactive engines. They differ from the way the power of *aums* are in use. Firstly *aums* code for lexicons then for automata and finally for transducers. But the modularity notion is the same for the three engines. And at the end of the article we present the way it is compiled (using the Berry-Sethi algorithm) into a macro-generated dispatching module used by the various reactive engines. The Berry-Sethi algorithm is described in purely functional code guaranted having the theoretical complexity.
More generally, the design of all those algorithms exploits and justifies the functional programming methodology in which algreabric closure operations are easily described, formal proofs are amenable, concise expression of powerful control paradigms is possible and the resulting tool is efficient for a concrete linguistic application.

3 Ongoing work

We have used the Berry-Sethi algorithm because it produces efficiently a compact non-deterministic automaton. Such an automaton has the property that for every state, every edge pointing that state have the same label. This property is due to a linearization step of the regular expression which leeds to the definition of a local language[5, 3]. This linearization cannot treat regular expressions with additional useful operators like complement and intersection. Then we have studied other automata constructions as defined by Brzozowski[4], Raymond[11] and Antimirov[1]. As a short summary :

- **Brzozowski**'s algorithm treats regular expressions with additional operators and produce a deterministic automaton, this construction is exponential but practically possible.
- **Raymond**'s algorithm is very efficient since it produces a non-determnistic with ϵ-transition automaton of the linear length of the regular expression and in linear time. But it does not extend to additional operators and the presence of ϵ-transitions could be problematic, looping the reactive engine.
- **Antimirov**'s algorithm produces a more compact non-deterministic automaton than the Berry-Sethi does, but the algorithm is not as efficient. Antimirov also indicates as a possible further research the question if his algorithm can be adapted to extended regular expressions.

We plan to extend our regular expression language in the same spirit of Kaplan and Kay[10] which presented a way to define phonological rewriting rules as rational relations adding some macro operators over the basic ones of regular expressions. We then aim to present efficient algorithms to compile our extended language, inspired from relational Kleene algebras, in a *Zen* style.

References

1. V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.
2. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(1):117–126, 1986.
3. J. Berstel and J.-E. Pin. Local languages and the berry-sethi algorithm. *Theor. Comput. Sci.*, 155(2):439–446, 1996.
4. J. A. Brzozowski. Derivatives of regular expressions. *J. Assoc. Comp. Mach.*, 11(4):481–494, October 1964.
5. S. Eilenberg. *Automata, Languages, and Machines, Volume A.* Academic Press, Inc., Orlando, FL, USA, 1974.
6. G. Huet. The zen computational linguistics toolkit. *ESSLLI 2002 Lectures, Trento, Italy*, 2002.
7. G. Huet. Automata mista. *verification: theory and practice: essays dedicated to zohar manna on the occasion of his 64th birthday. Springer-Verlag LNCS*, 2772:359–372, 2003.
8. G. Huet. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *J. Functional programming*, 15, 2005.

9. G. Huet and B. Razet. the reactive engine for modular transducers. *In Algebra, Meaning and Computation, Festschrift in Honor of Prof. Joseph Goguen*, to appear 2006.
10. R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics 20(3):331-378*, 1994.
11. P. Raymond. Recognizing regular expressions by means of dataflows networks. In *23rd International Colloquium on Automata, Languages, and Programming, (ICALP'96)*, Paderborn, Germany, July 1996. LNCS 1099, Springer Verlag.

Resolution Based Natural Deduction For Modal Logic

David Robinson

University Of Manchester

1 Introduction

In resolution based natural deduction (RND) [1], Andrzej Indrzejczak introduces a system called RND that mixes some features of two of the most well known systems, natural deduction and resolution. This combination of the two methods gives a proof system that produces proofs that are easily understood, unlike resolution proofs which can make little sense to understand individual steps, and fairly straight forward to produce, unlike some natural deduction proofs such as proof by contradicition.

The following will introduce and develop the system of RND for modal logic and then introduce a method which makes it possible to automatically generate RND proofs using resolution theorem provers. To save space, only the rules and encodings required for an example will be presented.

2 Modal Logic and RND

The syntax and semantics of modal logic used is that of Goré [2].

In this section I will present the resolution based natural deduction system, RND, introduced by Indrzejczak [1], but presented in a more recognisable style as introduced in [3].

Figure 1 shows a selection of the rules of RND for modal logic K. The α rules use the compact notation of Smullyan [4]. The [Sub] rule allows the making and discharging of assumptions. Formulae enclosed in square brackets are assumptions which are discharged when the rule is applied. The α rules allow manipulation of connectives and the introduction and elimination forms are the mirror of each other. The R' rule is the resolution rule, and is used here instead of the less general resolution rule given in the original presentation [1] for simplicity. It is easy to show that the R' resolution rule is derivable in the original system [1]. The $\Box$ rules are standard.

$$[Sub]\ \dfrac{\begin{array}{c}[s:\neg\varphi_1]\quad[...]\quad[s:\neg\varphi_k]\\ \vdots\\ s:\gamma\end{array}}{s:\gamma\vee\varphi_1\vee...\vee\varphi_k}\qquad(\alpha I)\ \dfrac{s:\mathit{\Gamma}\vee\alpha_1\quad s:\mathit{\Gamma}\vee\alpha_2}{s:\mathit{\Gamma}\vee\alpha}\qquad(\Box I)\ \dfrac{\begin{array}{c}[R(s,t)]\\ \vdots\\ t:\varphi\end{array}}{s:\Box\varphi}$$

$$(R')\ \dfrac{s:\mathit{\Gamma}\vee\varphi\quad s:\mathit{\Delta}\vee\neg\varphi}{s:\mathit{\Gamma}\vee\mathit{\Delta}}\qquad(\alpha E)\ \dfrac{s:\mathit{\Gamma}\vee\alpha}{s:\mathit{\Gamma}\vee\alpha_1\quad s:\mathit{\Gamma}\vee\alpha_2}\qquad(\Box E)\ \dfrac{s:\Box\varphi\quad R(s,t)}{t:\varphi}$$

Fig. 1. RND rules for Modal Logic K

3 Encodings

Any formula can be structurally transformed using the transformations shown in figure 2. These are applied to connectives present in the formula in turn to get an equivalent set of clauses which are suitable for resolution. The two tables show the encoding depending upon whether the connective appears positively or negatively in the formula being encoded.

φ	*negative*
$\psi_1 \vee \psi_2$	$Q_\varphi(x) \vee \neg Q_{\psi_1}(x)$ $Q_\varphi(x) \vee \neg Q_{\psi_2}(x)$
$\psi_1 \wedge \psi_2$	$Q_\varphi(x) \vee \neg Q_{\psi_1}(x) \vee \neg Q_{\psi_2}(x)$
$\neg\psi$	$Q_\varphi(x) \vee Q_\psi(x)$
$\neg(\psi_1 \vee \psi_2)$	$Q_\varphi(x) \vee Q_{\psi_1}(x) \vee Q_{\psi_2}(x)$
$\Box\psi$	$Q_\varphi(x) \vee R(x, f(x))$ $Q_\varphi(x) \vee \neg Q_\psi(f(x))$
$\Diamond\psi$	$Q_\varphi(x) \vee \neg R(x, y) \vee \neg Q_\psi(y)$

φ	*positive*
$\psi_1 \vee \psi_2$	$\neg Q_\varphi(x) \vee Q_{\psi_1}(x) \vee Q_{\psi_2}(x)$
$\psi_1 \wedge \psi_2$	$\neg Q_\varphi(x) \vee Q_{\psi_1}(x)$ $\neg Q_\varphi(x) \vee Q_{\psi_2}(x)$
$\neg\psi$	$\neg Q_\varphi(x) \vee \neg Q_\psi(x)$
$\neg(\psi_1 \vee \psi_2)$	$\neg Q_\varphi(x) \vee \neg Q_{\psi_1}(x)$ $\neg Q_\varphi(x) \vee \neg Q_{\psi_2}(x)$
$\Box\psi$	$\neg Q_\varphi(x) \vee \neg R(x, y) \vee Q_\psi(y)$
$\Diamond\psi$	$\neg Q_\varphi(x) \vee R(x, f(x))$ $\neg Q_\varphi(x) \vee Q_\psi(f(x))$

Fig. 2. Structural Transformations for Connectives

Figure 3 shows encodings for some of the RND rules for modal logic K. C, D represent sets of formulae, possibly empty. A clause containing C or D will be called a *derived clause*, as it will be a clause that has been derived from previous steps and does not form part of the encoding of the original problem, except in the first application of a rule at the beginning of a proof where C, D will be empty. The clause under the line is the *resolvent* of the clauses above possibly with factoring performed.

$$(\alpha E)\,\frac{\begin{array}{l} C \vee Q_\alpha(t) \\ \neg Q_\alpha(x) \vee Q_{\alpha_i}(x) \end{array}}{C \vee Q_{\alpha_i}(t)} \qquad \alpha I)\,\frac{\begin{array}{l} C \vee Q_{\alpha_1}(t) \\ D \vee Q_{\alpha_2}(t) \\ Q_\alpha(x) \vee \neg Q_{\alpha_1}(x) \vee \neg Q_{\alpha_2}(x) \end{array}}{C \vee D \vee Q_\alpha(t)}$$

$$(\Box E)\,\frac{\begin{array}{l} C \vee Q_{\Box\varphi}(s) \\ D \vee R(s,t) \\ \neg Q_{\Box\varphi}(x) \vee \neg R(x,y) \vee Q_\varphi(y) \end{array}}{C \vee D \vee Q_\varphi(t)} \qquad (\Box I)\,\frac{\begin{array}{l} [q_\Box(s) \vee R(s,t)] \\ \quad\vdots \\ C \vee q_R(s) \vee Q_\varphi(s) \\ Q_{\Box\varphi}(x) \vee \neg q_R(x) \\ Q_{\Box\varphi}(x) \vee \neg Q_\varphi(f(x)) \end{array}}{C \vee Q_{\Box\varphi}(t)}$$

$$[Sub]\,\frac{\begin{array}{l} C \vee q_{\neg\beta_i}(s) \vee Q_{\beta_j}(s) \\ \neg q_{\neg\beta_i}(x) \vee Q_{\beta_i}(x) \\ Q_\beta(x) \vee \neg Q_{\beta_i}(x) \\ Q_\beta(x) \vee \neg Q_{\beta_j}(x) \end{array}}{C \vee Q_\beta(x)} \qquad (Res)\,\frac{\begin{array}{l} C \vee Q_\psi(t) \vee Q_\theta(t) \\ D \vee \neg Q_\theta(t) \vee Q_\varphi(t) \end{array}}{C \vee D \vee Q_\psi(t) \vee Q_\varphi(t)}$$

Fig. 3. Encodings of RND Rules

All other clauses are *definitional clauses* that appear in the encoding of the problem if the rule can be applied.

In $\Box I$ the square brackets indicate a clause that is not used in the rule application but is a clause that appears somewhere in the generation of one of the derived clauses, and relates exactly to the subproof in the RND $\Box I$ rule.

The encoding will also include splitting on any positive clauses generated with q's introduced into clauses generated by the splitting. These q's correspond to assumptions in the RND, and this can be seen in the structural encoding, where the only positive clauses generated are for $\Box$ and $\neg$ formulae, corresponding to the two RND rules that require assumptions.

Theorem 1. *It is possible to simulate any RND derivation using the above structural transformation and rule encodings.*

Conjecture 1. I strongly believe that it is possible to automatically generate any RND proof using the same encodings.

4 A Simple $\Box$ Example

The simple example in figure 4 shows the resolution proof on the left hand side, beside the RND proof on the right. The first 9 lines of the resolution proof are the encoding of the formula, with the connective numberings indicated at the top. Lines 10-13 are the result of splitting on positive definitional clauses. The RND rule to which some of the resolution steps relates is also indicated for some lines in brackets.

$s : \neg\Box(\varphi \wedge \psi) \vee \Box\varphi$
$s : 2\ 4(\ \ 5\ \ \ )\ 1\ 3$

Resolution Proof

1	$\neg Q_1(s)$	
2	$Q_1(x) \vee \neg Q_2(x)$	
3	$Q_1(x) \vee \neg Q_3(x)$	
4	$Q_2(x) \vee Q_4(x)$	
5	$Q_3(x) \vee R(x, f(x))$	
6	$Q_3(x) \vee \neg Q_\varphi(f(x))$	
7	$\neg Q_4(x) \vee \neg R(x, y) \vee Q_5(y)$	
8	$\neg Q_5(x) \vee Q_\varphi(x)$	
9	$\neg Q_5(x) \vee Q_\psi(x)$	
10	$q_4(x) \vee Q_4(x)$	Split 4
11	$\neg q_4(x) \vee Q_2(x)$	Split 4
12	$q_R(x) \vee R(x, f(x))$	Split 5
13	$\neg q_R(x) \vee Q_3(x)$	Split 5

14	$q_R(x) \vee q_4(x) \vee Q_5(f(x))$	$12, 10, 7$ ($\Box E$)
15	$q_R(x) \vee q_4(x) \vee Q_\varphi(f(x))$	$14, 8$ (αE)
$15a$	$q_R(x) \vee q_4(x) \vee Q_\psi(f(x))$	$14, 9$ (αE)
16	$q_4(x) \vee Q_3(x) \vee Q_\varphi(f(x))$	$15, 13$ ($\Box I$)
17	$q_4(x) \vee Q_3(x) \vee Q_3(x)$	$16, 6$ ($\Box I$)
18	$q_4(x) \vee Q_3(x)$	*fact.*
19	$Q_2(x) \vee Q_3(x)$	$18, 11$
20	$Q_1(x) \vee Q_2(x)$	$19, 3$
21	$Q_1(x) \vee Q_1(x)$	$20, 2$
22	$Q_1(x)$	*fact.*
23	$\perp$	$22, 1$

RND Proof

1	$s : \Box(\varphi \wedge \psi)$	*ass.*
2	$R(s, t)$	*ass.*
3	$t : \varphi \wedge \psi$	$\Box E$ 1
4	$t : \varphi$	αE 3
$4a$	$t : \psi$	αE 3
5	$s : \Box\varphi$	$\Box I$ $2, 4$
6	$s : \neg\Box(\varphi \wedge \psi) \vee \Box\varphi$	[*Sub*] $1, 5$

Fig. 4. A Simple $\Box$ Example

Line 14 is the resolvent of lines 12, 10 and 7, using hyperresolution which gives $q_R(x) \vee q_4(x) \vee Q_5(f(x))$. The corresponding inference steps in the RND proof are steps 1-3. 1 is the assumption of $s : \Box(\varphi \wedge \psi)$ which corresponds to clause 10 in the resolution proof, 2 is the assumption of $R(s, t)$ which corresponds to clause 12. The derivation of $t : \varphi \wedge \psi$ in step 3 using the $\Box E$ rule (and 1 and 2) corresponds to the hyperresolution step producing clause 14. We see that $Q_5(f(x))$ corresponds to $t : \varphi \wedge \psi$ because 5 is the number of $\varphi \wedge \psi$ in the original formula.

Line 15 and 15a are the resolvents of lines 14 and 8, and 14 and 9 respectively, giving $q_R(x) \vee q_4(x) \vee \alpha_i$ for $i \in 1, 2$. The corresponding steps in the RND proof are steps 4 and 4a. These two steps are the derivations of the α_i's of $t : \varphi \wedge \psi$ using the αE rule and are still dependent upon the two assumptions $s : \Box(\varphi \wedge \psi)$ and $R(s, t)$. This corresponds to lines 15 and 15a which still contain the q's indicating these lines still depend on the assumptions, and which now contain $Q_\psi(f(x))$ and $Q_\varphi(f(x))$, the two formulae derived in the RND proof.

Line 18 is the resolvent of line 15 with line 13 and 6 and with factoring applied. This is done in a number of steps at lines 16, 17 and 18 for clarity but could have been done in one hyperresolution step. The corresponding step in the RND proof is step 5, the derivation of $s : \Box\varphi$ using the $\Box I$ rule and discharging the assumption $R(s,t)$. We see that $q_R(x)$ no longer appears in the clause at line 18, and that $Q_3(x)$ corresponds to $s : \Box\varphi$ since 3 is the number of $\Box\varphi$ in the original formula.

Line 22 of the proof is the resolvent of line 18 with 11,3 and 2. This can again be combined into one single hyperresolution step, but the steps are performed separately for clarity. The corresponding step in the RND proof is step 6, the derivation of the initial formula by the [Sub] rule and discharging the remaining assumption $s : \Box(\varphi \wedge \psi)$. We again see that $q_4(x)$ which corresponds to $s : \Box(\varphi \wedge \psi)$ is no longer present in the clause at line 22, and that $Q_1(x)$ corresponds to the initial formula to be proved.

Line 23 completes the resolution proof by deriving $\perp$ as required.

5 Conclusion

The example in figure 4 shows that it is possible to simulate RND steps using clauses from the encoding of the initial formula. It is also possible to establish what information is required in the clauses in order that an RND rule can be applied. The rule encodings given in figure 3 provide the clauses that are required in order that the RND rules can be simulated, and the clause that would be derived given this information. I believe therefore that it is possible to generate an RND proof automatically, using these encodings by only ever resolving clauses that map exactly to an RND rule encoding. A proof of this type could then be generated using modern resolution theorem provers, and could then easily be translated into the corresponding RND proof.

This should give easily and automatically generated proofs which can be easily read and understood by a human reader. I hope to have developed a proof of this before the conference.

References

1. Andrzej Indrzejczak. Resolution Based Natural Deduction. *Bulletin of the Section of Logic*, pages 159–170, 2002.
2. Reiner Hahnle Marcello D'Agostino, Dov M. Gabbay and Joachim Posegga. *Handbook of Tableau Methods.* Kluwer Academic Publishers, 1999.
3. David Robinson. Resolution based natural deduction. Master's thesis, Manchester University, Mathematics, 2004.
4. Raymond M. Smullyan. *First-Order Logic.* Dover Publications, Inc., 1995.

Some Notes on Duality in Refinement Algebra

Kim Solin*

TUCS (Turku Centre for Computer Science)
Lemminkäinengatan 14 A, FIN-20520 Åbo, Finland
kim.solin@utu.fi

Abstract. We formulate a duality principle for refinement algebras. We first consider the dual of Kleene star: angelic iteration. Angelic iteration was introduced by Back, Mikhajlova and von Wright in a predicate-transformer setting, here we propose an abstract-algebraic characterisation. This allows us to formulate the duality principle. We conclude by considering iterative choice and by introducing the dual of action systems.

1 Introduction

Duality in different structures has often been used to simplify proofs, to give two-in-one theorems and has inspired the conception of new and useful structures. In the program refinement tradition, duality has proved a useful technical tool and seems to have inspired new structures as well [1]. The purpose of this paper is to cast the duality of some program-refinement concepts in the form of the recent abstract refinement algebras.

Refinement algebras are abstract algebras (simply a set equipped with operators) for reasoning about program refinement [6, 7, 5, 4]. It can be argued that by reasoning on a more abstract level one obtains a more perspicuous view than provided by the classical model-theoretic frameworks: due to the abstraction there are not so many details that clutter the view. Thanks to this perspicuity, seeing common features amongst already established frameworks is also made easier.

2 A refinement algebra with angelic iteration

In [2] Back, Mikhajlova and von Wright introduced the angelic iteration operator $^{\phi}$ in a predicate-transformer setting. Angelic iteration can be seen as a finite repetition of a program statement of any length determined by the user (as opposed to being determined by the system). It is not hard to consider angelic iteration abstract-algebraically and this is what we shall do in this section of the paper. An intuition and a model for the operators involved in the definition below will be given in the next section. We shall introduce the operator into a

* Work done while visiting Institut für Informatik, Universität Augsburg.

refinement algebra having a signature containing all the operators that (as of yet) have been considered. However, by varying the signature and/or by having the conjunctivity condition to hold for all elements, we can get other refinement algebras, such as the ones in [7] and [4], of which the later one also could harbour angelic iteration.

A *full refinement algebra* is a structure over the signature $(\sqcap, \sqcup, \neg, ;, ^*, ^\phi, ^\omega, ^\dagger, \bot, \top, 1)$ such that $(\sqcap, \sqcup, \neg, \bot, \top)$ is a Boolean algebra, $(;, 1)$ is a monoid, and the following equations hold (the operator $\neg$ binds stronger than the equally strong $^*, ^\omega, ^\phi$ and †, which in turn bind stronger than ;, which, finally, binds stronger than the equally strong $\sqcap$ and $\sqcup$; $x \sqsubseteq y \Leftrightarrow_{df} x \sqcap y = x$; and ; is left implicit):

$$\begin{array}{rcl} \neg xy = \neg(xy), & & \\ \top x = \top, & & \bot x = \bot, \\ (x \sqcap y)z = xz \sqcap yz & \text{and} & (x \sqcup y)z = xz \sqcup yz. \end{array}$$

Moreover, if an element x satisfies $y \sqsubseteq z \Rightarrow xy \sqsubseteq xz$ we say that x is *isotone*, and if x and y are isotone, then

$$\begin{array}{lll} x^* = xx^* \sqcap 1, & & z \sqsubseteq xz \sqcap y \Rightarrow z \sqsubseteq x^*y, \\ x^\phi = xx^\phi \sqcup 1, & & xz \sqcup y \sqsubseteq z \Rightarrow x^\phi y \sqsubseteq z, \\ x^\omega = xx^\omega \sqcap 1, & & xz \sqcap y \sqsubseteq z \Rightarrow x^\omega y \sqsubseteq z, \\ x^\dagger = xx^\dagger \sqcup 1 & \text{and} & z \sqsubseteq xz \sqcup y \Rightarrow z \sqsubseteq x^\dagger y \end{array}$$

hold. If x satisfies $x(y \sqcap z) = xy \sqcap xz$ and $x(y \sqcup z) = xy \sqcup xz$ we say that x is *conjunctive* and *disjunctive*, respectively. Of course, conjunctivity or disjunctivity implies isotony. If an element is both conjunctive and disjunctive, then we say that it is *functional*. If x and y are conjunctive, then * and $^\omega$ are assumed to satisfy

$$x^\omega = x^* \sqcap x^\omega \top \qquad \text{and} \qquad z \sqsubseteq zx \sqcap y \Rightarrow z \sqsubseteq yx^*,$$

and if x and y are disjunctive then

$$x^\dagger = x^\phi \sqcup x^\dagger \bot \qquad \text{and} \qquad zx \sqcup y \sqsubseteq z \Rightarrow yx^\phi \sqsubseteq z$$

are assumed to hold.[1]

Guards and *assertions* are special elements of the carrier set. An element g is a *guard* if it is is functional, it has a complement $\bar{g}$ satisfying $g\bar{g} = \top$ and $g \sqcap \bar{g} = 1$, and for any g' also satisfying the two first conditions it holds that $gg' = g \sqcup g'$. An element p is an *assertion* if it is functional, it has a complement $\bar{p}$ satisfying $p\bar{p} = \bot$ and $p \sqcup \bar{p} = 1$, and for any p' also satisfying the two first conditions it holds that $pp' = p \sqcap p'$. If G is the set of guards and A is the set of assertions, then $(G, \sqcap, ;, ^-, 1, \top)$ and $(A, ;, \sqcup, ^-, \bot, 1)$ are Boolean algebras. We will use $g, g_1, g_2, \ldots$ to denote guards and $p, p_1, p_2, \ldots$ to denote assertions.

[1] Let C and D be the sets of conjunctive and disjunctive elements, respectively. When the operators are interpreted as above, the structures $(C, ;, \sqcap, ^*, \bot, 1)$ and $(D, ;, \sqcup, ^\phi, \top, 1)$ satisfy all the axioms of a Demonic Algebra [3] except right annihilation for $\bot$ and $\top$, respectively.

The *enabledness operator* ϵ is a mapping from the set of isotone elements to the set of guards defined by $\epsilon x =_{df} x\bot \sqcup 1$. The *termination operator* τ is a mapping from isotone elements to the set of assertions defined by $\tau x =_{df} x\top \sqcap 1$.

3 Intuition and a model

The elements of the carrier set can be seen as program statements. The operators should be understood so that $\sqcap$ is *demonic choice* (a choice we cannot affect, a choice by the system), $\sqcup$ is *angelic choice* (a choice we can affect, a choice made by the user), ; is *sequential composition*, $\neg x$ terminates from any state where x would not terminate and the other way around. The constant $\bot$ is abort, an always aborting program statement; $\top$ is magic, a program statement that establishes any postcondition; and 1 is skip. If y establishes anything that x does and possibly more, then x is refined by y: $x \sqsubseteq y$. *Weak iteration* * (Kleene star) can be seen as an iteration of any finite length determined by the system. The *(weak) angelic iteration* $^\phi$ can be seen as a finite repetition of a program statement in which the length of the iteration is determined by the user. *Strong iteration*, $^\omega$, is an iteration that either terminates *or* goes on infinitely, in which case it aborts, and †, the *strong angelic iteration* [4], is an iteration that terminates *or* goes on infinitely, in which case a miracle occurs. The difference between the operators can be displayed by the fact that $1^* = 1, 1^\omega = \bot, 1^\phi = 1$ and $1^\dagger = \top$.

A conjunctive element can be seen as facilitating demonic nondeterminism, but not angelic, whereas a disjunctive element can have angelic nondeterminism, but not demonic. An isotone element permits both kinds of nondeterminism.

Guards should be thought of as programs that check if some predicate holds, skip if that is the case, and otherwise a miracle occurs. Assertions are similar to guards, but instead of performing a miracle when the predicate does not hold, they abort. The enabledness operator maps any program to a guard that skips in those states in which the program is enabled, that is, in those states from which the program will not terminate miraculously. The termination operator applied to a program denotes an assertion that skips in those states from which the program is guaranteed to terminate, that is, states from which it will not abort.

The operators, the guards and the assertions can all be given an interpretation such that the set of predicate transformers over a fixed state space forms a full refinement algebra.

4 A duality principle

Given a statement Φ about a refinement algebra, we formulate the order-dual statement Φ^∂ by replacing occurrences of symbols according to the following rules: $\sqsubseteq$ is replaced by $\sqsupseteq$ and vice versa, $\sqcap$ is replaced by $\sqcup$ and vice versa, * is replaced by $^\phi$ and vice versa, $^\omega$ is replaced by † and vice versa, $\top$ is replaced by $\bot$ and vice versa, any arbitrary guard g is replaced by an arbitrary assertion

p and vice versa, and ϵ is replaced by τ and vice versa. We assume that a given signature that contains an opertor $\mathbf{o}$ also contains the dual operator $\mathbf{o}^{\partial}$ (its replacement operator according to the above). By this, we can now formulate a duality principle for refinement algebras: *Given statement Φ which holds true in a refinement algebra, the dual statement Φ^{∂} also holds true in the refinement algebra.*

Note that although ϵx and τx are a guard and an assertion respectively, their duality does not follow from that fact, but from the duality of their respective definitions. Moreover, from the properties of Boolean algebra it directly follows that $\sqcap$ and $\sqcup$ are de Morgan dual with respect to the negation.

5 Iterative choice and dual action systems

The *iterative choice* construct [1, 2], $\mathsf{bo}\ p_1 :: x_1 \between \dots \between p_n :: x_n\ \mathsf{ob}$, can be seen as an iteration done by the user and in every iteration step the user can choose either to execute one of the statements $x_1, \dots, x_n$, provided the related assertion holds, and continue the iteration – or choose to skip and end the iteration. The user will be assumed to choose a statement for which the guard holds. The iterative choice statement has been used to reason about interactive programs [2]. An intuitive example of an iterative choice is an interactive dialog box (a menu).

By expressing iterative choice in an abstract refinement algebra we would have the possibility to reason about interactive programs also on a more abstract level – and, perhaps, to more easily see connections between different frameworks intended for reasoning about interaction. It is easy to give iterative choice an abstract formulation. In fact, in a concrete predicate-transformer algebra it was formulated in [2], so the only thing we need to do is to translate this into the abstract algebra: $(p_1 x_1 \sqcup \dots \sqcup p_n x_n)^{\phi}$. By the duality principle and known results for weak iteration and demonic choice, this directly yields useful properties such as decomposition, $(p_1 x_1 \sqcup p_2 x_2)^{\phi} = (p_1 x_1)^{\phi}(p_2 x_2 (p_1 x_1)^{\phi})^{\phi}$, and leapfrog, $(p_1 x_1 x_2)^{\phi} p_1 x_1 \sqsubseteq p_1 x_1 (x_2 p_1 x_1)^{\phi}$. In the concrete predicate-transformer algebra, these and other results were noted to arise from duality already in [2].

In [4, 5] action systems were considered abstract-algebraically. We now consider the dual of an action system. To the best of our knowledge, this structure has not earlier appeared in the literature and constitutes an example of how duality considerations can give rise to new interesting structures. A *dual action system* $\mathsf{bo}\ x_1 \between \dots \between x_n\ \mathsf{ob}$ is abstract-algebraically defined by $(x_1 \sqcup \dots \sqcup x_n)^{\dagger}\overline{\tau x_1} \dots \overline{\tau x_n}$, where $x_1, \dots x_n$ are disjunctive *actions*.

An intuition is that the user iterates and for every iteration the user should choose one of the terminating actions. The user should choose to end the iteration when none of the actions are any longer guaranteed to terminate. If the user would end the iteration prematurely, the statement $\overline{\tau x_1} \dots \overline{\tau x_n}$ would ensure that the dual action system would abort and if the user would continue to iterate although no action would be guaranteed to terminate (by choosing a nonterminating action), the dual action system would of course also abort. (One can also look at it as if the iteration ends automatically when none of the actions will terminate.) The user can also attempt to continue the iteration forever (*e.g.* by in turn increasing and decreasing a value of variable). This kind of behaviour could be called angelic nontermination or *perpetuum mobile behaviour*. If the user succeeds in doing this, a miracle is brought about (so here we equal perpetuum mobile behaviour and miraculous behaviour).

Many interesting properties of dual action systems follow from duality and results on classical action systems. One such property is leapfrog for dual action systems: $(x_1x_2)^\dagger\overline{\tau(x_1x_2)}x_1 \sqsubseteq x_1(x_2x_1)^\dagger\overline{\tau(x_2x_1)}$.

A more concrete example of a dual action system is a code lock consisting of a numerical pad and an enter button. The lock works on a state space and the numerical keys act on this space. Opening the lock corresponds to the action system terminating, entering a wrong code and thus setting off the alarm corresponds to the action system aborting, and being able to click the numbers for ever corresponds to having found an *evighetsmaskin*. Entering a digit corresponds to choosing one of the actions, choosing the wrong digit might set off the alarm. Hitting the enter button corresponds to ending the iteration. (One can also look at it as if the iteration automatically ends when a correct code is entered.) There are several codes – possibly of different length – that produce a state so that none of the actions will terminate and thus open the lock.

6 Ending remarks

We considered angelic iteration abstract-algebraically which enabled us to formulate a duality principle. The duality principle can be used for proving two-in-one theorems and also inspires the conception of new structures, as exemplified by the dual action-system construct. The dual action systems were here only sketched and deserve further investigation to determine if they, in combination with classical constructs, could be employed for reasoning about more elaborate interactive systems.

Acknowledgements. Peter Höfner and Bernhard Möller both provided constructive critique that substantially improved the paper.

References

1. R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.
2. R.-J. Back, A. Mikhajlova and J. von Wright. Modeling Component Environments and Interactive Programs Using Iterative Choice. Technical Report 200, TUCS, 1998.
3. J.-L. de Carufel and J. Desharnais. Demonic algebra with domain. Accepted to RelMiCS/AKA 2006.
4. K. Solin. On Two Dually Nondeterministic Refinement Algebras. Accepted to RelMiCS/AKA 2006.
5. K. Solin and J. von Wright. Refinement Algebra with Operators for Enabledness and Termination. In *Math. of Progr. Constr.,* vol. 4014 of *LNCS*, Springer, 2006.
6. J. von Wright. From Kleene Algebra to Refinement Algebra. In *Math. of Prog. Constr.,* vol. 2386 of *LNCS*, Springer, 2002.
7. J. von Wright. Towards a Refinement Algebra. *Sci. of Comp. Prog.*, 51, 2004.