Rudolf Berghammer
Bernhard Möller
Georg Struth (eds.)

Relations and Kleene Algebra in Computer Science

PhD Programme at RelMiCS10/AKA5

Frauenwörth, Germany
April 7 – April 11, 2008

Proceedings

Technical Report 2008-04, Institut für Informatik, Universität Augsburg, Germany

Preface

This volume contains the tutorial materials and the contributed extended abstracts of the PhD Programme at the *10th International Conference on Relational Methods in Computer Science (RelMiCS10)* and the *5th International Conference on Applications of Kleene Algebra (AKA5)*. The programme has been organised for the second time in association with RelMiCS/AKA. It took place in Frauenwörth on an Island in Lake Chiem in Bavaria, from April 7 to April 11, 2008, and included invited tutorials, a student session and attendance at the conference.

Eight extended abstracts by students were selected for the programme by the organisers due to the relevance and quality of their submissions. They nicely reflect the diverse applications of relations and Kleene algebras in computing. The student session allowed the participants to present and discuss their own work. In addition there were three invited tutorials: *Basics of Relation Algebra* by Jules Desharnais (Université Laval, Quebec, Canada), *Basics of Modal Kleene Algebra* by Georg Struth (University of Sheffield, UK) and *Basics of Preference and Fuzzy Preference Modeling* by Susanne Saminger (Universität Linz, Austria).

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. This year's proceedings, published as volume 4988 of the Springer LNCS series, contain 28 contributions by researchers from all over the world. Next to 26 regular papers there are the invited papers *Formal Methods and the Theory of Social Choice* by Marc Pauly (Stanford University, USA) and *Relations Making Their Way from Logics to Mathematics and Applied Sciences*, by Gunther Schmidt (University of the Armed Forces Munich, Germany). The papers show that relational and Kleene Algebra methods have wide-ranging impact and applicability in theory and practice.

The organisers would warmly like to thank all those who contributed to the success of the programme: the tutorial speakers Susanne Saminger and Jules Desharnais for accepting our invitation, the students for their interest in the programme and the local organisers at the Universities of Augsburg and Kiel, notably Roland Glück, Peter Höfner, Iris Kellner and Ulrike Pollakowski, for their dedicated help; they made organising this meeting a pleasant experience. Finally, we want to thank our sponsors ARIVA.DE AG (Kiel), CrossSoft (Kiel), HSH Nordbank AG (Kiel) and the Deutsche Forschungsgemeinschaft DFG for their financial support.

April 2008

Rudolf Berghammer
Bernhard Möller
Georg Struth

Table of Contents

Invited Tutorials

Contributed Extended Abstracts

Basics of Relation Algebra

Jules Desharnais

Départment d'informatique et de génie logociel
Université Laval, Québec, Canada
Jules.Desharnais@ift.ulaval.ca

Basics of Relation Algebra

Jules Desharnais
Département d'informatique et de génie logiciel
Université Laval, Québec, Canada

Plan

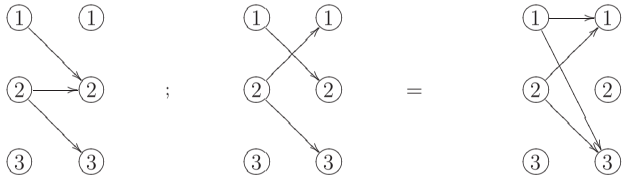
1 Relations on a set

1. A relation on a set S is a subset of $S \times S$
2. The set of relations on S is $\mathcal{P}(S \times S)$
3. Notation: $sRt \;\Leftrightarrow\; (s,t) \in R, \qquad V = S \times S$
4. Operations on relations
   - Set-theoretical operations: $\cup$, $\cap$, $\overline{}$, $\emptyset$, V
   - Relational operations
     - Composition (relative product): $sQ;Ru \Leftrightarrow (\exists t : sQt \wedge tRu)$
     - Converse: $sR^{\smile}t \Leftrightarrow tRs$
     - Identity relation: $sIt \Leftrightarrow s = t$

Representations of relations: sets of ordered pairs, graphs, matrices

Let $S \stackrel{\text{def}}{=} \{1,2,3\}$.

Q	;	R	=	$Q;R$
$\{(1,2),(2,2),(2,3)\}$	;	$\{(1,2),(2,1),(2,3)\}$	=	$\{(1,1),(1,3),(2,1),(2,3)\}$



$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad ; \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad = \quad \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Relations on a set satisfy many laws

Increasing priority: $(\cup, \cap)$, $;$, $^\smile$

- $Q \cup R = R \cup Q$
- $P \cap (Q \cap R) = (P \cap Q) \cap R$
- $\overline{(Q \cup R)} = \overline{Q} \cap \overline{R}$
- $I;R = R$
- $P;(Q \cup R) = P;Q \cup P;R$
- $P;(Q \cap R) \subseteq (P;Q);R$
- $(Q;R)^\smile = R^\smile;Q^\smile$
- $R \neq \emptyset \;\Rightarrow\; V;R;V = V$
- $P;Q \subseteq R \;\Leftrightarrow\; P^\smile;\overline{R} \subseteq \overline{Q} \;\Leftrightarrow\; \overline{R};Q^\smile \subseteq \overline{P}$
- $P;Q \cap R \subseteq (P \cap R;Q^\smile);(Q \cap P^\smile;R)$
- ...

2 Relation algebra

Relation algebra (RA): Aims at "characterizing" relations on a set by means of simple equational axioms. It is a structure

$$\mathcal{A} = \langle A, \sqcup, \overline{}, ;, ^\smile, \mathbb{I} \rangle$$

such that

(1) $Q \sqcup R = R \sqcup Q$
(2) $P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R$
(3) $\overline{\overline{Q} \sqcup \overline{R}} \sqcup \overline{\overline{Q} \sqcup R} = Q$

} Boolean algebra axioms

2 Relation algebra

Relation algebra (RA): Aims at "characterizing" relations on a set by means of simple equational axioms. It is a structure

$$\mathcal{A} = \langle A, \sqcup, \overline{}, ;, ^\smile, \mathbb{I} \rangle$$

such that

(1) $Q \sqcup R = R \sqcup Q$
(2) $P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R$
(3) $\overline{\overline{Q} \sqcup \overline{R}} \sqcup \overline{\overline{Q} \sqcup R} = Q$

Another axiomatisation of Boolean algebra:
Add $\sqcap, \bot, \top$ to the signature, replace Huntington's axiom (3) by
$Q \sqcap R = R \sqcap Q$
$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$
$Q \sqcup (Q \sqcap R) = Q$
$Q \sqcap (Q \sqcup R) = Q$
$P \sqcup (Q \sqcap R) = (P \sqcup Q) \sqcap (P \sqcup R)$
$R \sqcup \bot = R$
$R \sqcap \top = R$
$R \sqcup \overline{R} = \top$
$R \sqcap \overline{R} = \bot$

2 Relation algebra

Relation algebra (RA): Aims at "characterizing" relations on a set by means of simple equational axioms. It is a structure

$$\mathcal{A} = \langle A, \sqcup, \overline{}, ;, ^\smile, \mathbb{I} \rangle$$

such that

(1) $Q \sqcup R = R \sqcup Q$
(2) $P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R$
(3) $\overline{\overline{Q} \sqcup \overline{R}} \sqcup \overline{\overline{Q} \sqcup R} = Q$

} Boolean algebra axioms

(4) $P;(Q;R) = (P;Q);R$
(5) $(P \sqcup Q);R = P;R \sqcup Q;R$
(6) $R;\mathbb{I} = R$
(7) $R^{\smile\smile} = R$
(8) $(Q \sqcup R)^\smile = Q^\smile \sqcup R^\smile$
(9) $(Q;R)^\smile = R^\smile;Q^\smile$
(10) $Q^\smile;\overline{Q;R} \sqcup \overline{R} = \overline{R}$

2 Relation algebra

Relation algebra (RA): Aims at "characterizing" relations on a set by means of simple equational axioms. It is a structure

$$\mathcal{A} = \langle A, \sqcup, \overline{\ }, ;, ^{\smile}, \mathbb{I} \rangle$$

such that

(1) $Q \sqcup R = R \sqcup Q$
(2) $P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R$
(3) $\overline{\overline{Q} \sqcup \overline{R}} \sqcup \overline{\overline{Q} \sqcup R} = Q$

} Boolean algebra axioms

(4) $P;(Q;R) = (P;Q);R$
(5) $(P \sqcup Q);R = P;R \sqcup Q;R$
(6) $R;\mathbb{I} = R$
(7) $R^{\smile\smile} = R$
(8) $(Q \sqcup R)^{\smile} = Q^{\smile} \sqcup R^{\smile}$
(9) $(Q;R)^{\smile} = R^{\smile};Q^{\smile}$
(10) $Q^{\smile};\overline{Q;R} \sqcup \overline{R} = \overline{R}$

Ordering $\sqsubseteq$

Define

$$Q \sqsubseteq R \;\Leftrightarrow\; Q \sqcup R = R.$$

Then (10) can be written

$$Q^{\smile};\overline{Q;R} \sqsubseteq \overline{R}.$$

2 Relation algebra

Relation algebra (RA): Aims at "characterizing" relations on a set by means of simple equational axioms. It is a structure

$$\mathcal{A} = \langle A, \sqcup, \overline{\ }, ;, ^{\smile}, \mathbb{I} \rangle$$

such that

(1) $Q \sqcup R = R \sqcup Q$
(2) $P \sqcup (Q \sqcup R) = (P \sqcup Q) \sqcup R$
(3) $\overline{\overline{Q} \sqcup \overline{R}} \sqcup \overline{\overline{Q} \sqcup R} = Q$

} Boolean algebra axioms

(4) $P;(Q;R) = (P;Q);R$
(5) $(P \sqcup Q);R = P;R \sqcup Q;R$
(6) $R;\mathbb{I} = R$
(7) $R^{\smile\smile} = R$
(8) $(Q \sqcup R)^{\smile} = Q^{\smile} \sqcup R^{\smile}$
(9) $(Q;R)^{\smile} = R^{\smile};Q^{\smile}$
(10) $Q^{\smile};\overline{Q;R} \sqcup \overline{R} = \overline{R}$

Derived operators $\sqcap, \bot, \top$

$$Q \sqcap R = \overline{\overline{Q} \sqcup \overline{R}}$$
$$\bot = \overline{\mathbb{I}} \sqcap \mathbb{I}$$
$$\top = \overline{\mathbb{I}} \sqcup \mathbb{I}$$

Laws that can be proved from the axioms

Relation algebra	Corresponding laws, relations on sets
$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$	$P \cap (Q \cap R) = (P \cap Q) \cap R$
$\overline{(Q \sqcup R)} = \overline{Q} \sqcap \overline{R}$	$\overline{(Q \cup R)} = \overline{Q} \cap \overline{R}$
$\mathbb{I};R = R$	$I;R = R$
$\mathbb{I}^{\smile} = \mathbb{I}$	$I^{\smile} = I$
$\top^{\smile} = \top$	$V^{\smile} = V$
$\top;\top = \top$	$V;V = V$
$P;(Q \sqcup R) = P;Q \sqcup P;R$	$P;(Q \cup R) = P;Q \cup P;R$
$P;Q \sqsubseteq R \;\Leftrightarrow\; P^{\smile};\overline{R} \sqsubseteq \overline{Q}$	$P;Q \subseteq R \;\Leftrightarrow\; P^{\smile};\overline{R} \subseteq \overline{Q}$
$P;Q \sqcap R \sqsubseteq (P \sqcap R;Q^{\smile});(Q \sqcap P^{\smile};R)$	$P;Q \cap R \subseteq (P \cap R;Q^{\smile});(Q \cap P^{\smile};R)$
???	$R \neq \emptyset \;\Rightarrow\; V;R;V = V$
⋮ ⋮ ⋮ ⋮	⋮ ⋮ ⋮ ⋮

Properties of the equational axiomatisation

Because RAs are defined by equations, the class of RAs is a **variety**: it is closed under **products**, **homomorphic images** and **subalgebras**.

Example. Consider the relations on $S_2 \stackrel{\text{def}}{=} \{1,2\}$ and $S_3 \stackrel{\text{def}}{=} \{1,2,3\}$ or, equivalently, the subsets of

$$V_2 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad V_3 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

1. **Products** The set of pairs of relations

$$A_{2,3} = \{(R_2, R_3) \mid R_2 \subseteq V_2 \;\wedge\; R_3 \subseteq V_3\}$$

is a RA with identity (I_2, I_3). Operations are defined pointwise. E.g., $(Q_2, Q_3);(R_2, R_3) = (Q_2;R_2, Q_3;R_3)$ and $(R_2, R_3)^{\smile} = (R_2^{\smile}, R_3^{\smile})$. The top relation is (V_2, V_3) or, on an isomorphic matrix form,

$$V_{2,3} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

2. **Homomorphic images** Let $f : A_{2,3} \to \mathcal{P}(S_2 \times S_2)$ be defined by

$$f((R_2, R_3)) = R_2.$$

Then f is a homomorphism.

An RA homomorphism is defined by the following properties.

$f : \mathcal{A} \to \mathcal{B}$			$f : A_{2,3} \to \mathcal{P}(S_2)$		
$f(Q \sqcup_{\mathcal{A}} R)$	$=$	$f(Q) \sqcup_{\mathcal{B}} f(R)$	$f((Q_2, Q_3) \cup (R_2, R_3))$	$=$	$Q_2 \cup R_2$
$f(\overline{R}^{\mathcal{A}})$	$=$	$\overline{f(R)}^{\mathcal{B}}$	$f(\overline{(R_2, R_3)})$	$=$	$\overline{R_2}$
$f(Q ;_{\mathcal{A}} R)$	$=$	$f(Q) ;_{\mathcal{B}} f(R)$	$f((Q_2, Q_3) ; (R_2, R_3))$	$=$	$Q_2 ; R_2$
$f(R^{\smile \mathcal{A}})$	$=$	$(f(R))^{\smile \mathcal{B}}$	$f((R_2, R_3)^{\smile})$	$=$	$(R_2)^{\smile}$
$f(\mathbb{I}_{\mathcal{A}})$	$=$	$\mathbb{I}_{\mathcal{B}}$	$f((I_2, I_3))$	$=$	I_2

The image of an RA homomorphism is an RA.

3. **Subalgebras**

- An RA
$$\mathcal{B} = \langle B, \sqcup, \overline{\ }, ;, ^{\smile}, \mathbb{I} \rangle$$
is a subalgebra of an RA
$$\mathcal{A} = \langle A, \sqcup, \overline{\ }, ;, ^{\smile}, \mathbb{I} \rangle$$
if $A \subseteq B$ (note: the operations are the same).
- For instance,
$$\langle \{\bot\!\!\!\bot, \mathbb{I}, \bar{\mathbb{I}}, \top\!\!\!\top\}, \sqcup, \overline{\ }, ;, ^{\smile}, \mathbb{I} \rangle$$
is a subalgebra of $\mathcal{A}$.
- Given $B \subseteq A$, a subalgebra can be generated by closing B under the operations of $\mathcal{A}$.

Models of the axioms

1. Relations on a set S where **the universal relation V is an equivalence relation**. For instance, all relations included in

$$V_{2,3} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

form a relation algebra. Now let

$$R_{2,3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and consider the composition

$$V_{2,3} ; R_{2,3} ; V_{2,3}\,.$$

$$\begin{aligned} V_{2,3} ; R_{2,3} ; V_{2,3} &= \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} ; \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} ; \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ &\neq \emptyset_{2,3} \end{aligned}$$

Thus, RAs do not in general satisfy the **Tarski rule**

$$R \neq \bot\!\!\!\bot \;\Leftrightarrow\; \top\!\!\!\top ; R ; \top\!\!\!\top = \top\!\!\!\top.$$

Those that do are **simple RAs** (with only two homomorphic images: themselves and the trivial RA with one element; they are not closed under products). For concrete relations on a set S, they are those with

$$V = S \times S.$$

Models of the axioms

2. Let $\mathcal{A} = \langle A, \sqcup, \overline{\ }, ;, ^\smile, \mathbb{I} \rangle$ be an RA. Let $\mathcal{M}_n$ be the set of $n \times n$ matrices with elements of A as entries. Define the following (red) operations on $\mathcal{M}_n$.

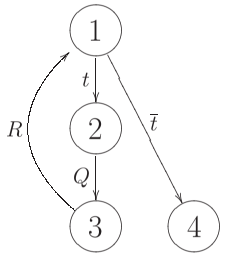
Operation	Definition
$\sqcup$	$(\mathbf{M} \sqcup \mathbf{N})[i,j] \stackrel{\text{def}}{=} \mathbf{M}[i,j] \sqcup \mathbf{N}[i,j]$
$\overline{\ }$	$\overline{\mathbf{M}}[i,j] \stackrel{\text{def}}{=} \overline{\mathbf{M}[i,j]}$
$;$	$(\mathbf{M};\mathbf{N})[i,j] \stackrel{\text{def}}{=} (\bigsqcup k : \mathbf{M}[i,k];\mathbf{N}[k,j])$
$^\smile$	$\mathbf{M}^\smile[i,j] \stackrel{\text{def}}{=} (\mathbf{M}[i,j])^\smile$
$\mathbb{I}$	$\mathbb{I}[i,j] \stackrel{\text{def}}{=} \begin{cases} \mathbb{I} & \text{if } i = j \\ \bot & \text{if } i \neq j \end{cases}$

Then

$$\langle \mathcal{M}_n, \sqcup, \overline{\ }, ;, ^\smile, \mathbb{I} \rangle$$

is an RA.

This model can be used for the description of programs.



$$\begin{pmatrix} \bot & t & \bot & \overline{t} \\ \bot & \bot & Q & \bot \\ R & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

The matrix (graph) represents the control structure. The entries of the matrix (labels of the graph) are relations describing how the state changes by a transition.

Expressing properties of relations

Property	Definition	Set-theoretical expresion
R total	$R;\top = \top$	$(\forall x : (\exists y : xRy))$
	$\mathbb{I} \sqsubseteq R;R^\smile$	
	$\overline{R} \sqsubseteq R;\overline{\mathbb{I}}$	
R univalent	$R^\smile;R \sqsubseteq \mathbb{I}$	$(\forall x,y,z : xRy \wedge xRz \Rightarrow y = z)$
(or functional)	$R;\overline{\mathbb{I}} \sqsubseteq \overline{R}$	
R reflexive	$\mathbb{I} \sqsubseteq R$	$(\forall x : xRx)$
R antisymmetric	$R \sqcap R^\smile = \mathbb{I}$	$(\forall x,y : xRy \wedge yRx \Rightarrow x = y)$
R transitive	$R;R \sqsubseteq R$	$(\forall x,y,z : xRy \wedge yRz \Rightarrow xRz)$
$\vdots$	$\vdots$	$\vdots$

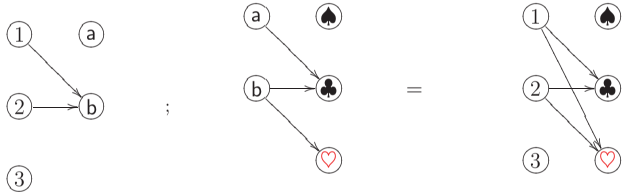
Using the relational instead of the set-theoretical definitions leads to equational proofs that are more compact and easier to verify.

3 Heterogeneous relation algebra

Relations between different sets

Let $S = \{1,2,3\}$, $T = \{\mathsf{a},\mathsf{b}\}$ and $U = \{\spadesuit, \clubsuit, \heartsuit\}$.

$$\begin{array}{ccccc} Q & ; & R & = & Q;R \\ \{(1,\mathsf{b}),(2,\mathsf{b})\} & ; & \{(\mathsf{a},\clubsuit),(\mathsf{b},\clubsuit),(\mathsf{b},\heartsuit)\} & = & \{(1,\clubsuit),(1,\heartsuit),(2,\clubsuit),(2,\heartsuit)\} \end{array}$$



$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \quad ; \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \quad = \quad \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Composition is possible if sizes match.

Axiomatising heterogeneous relation algebra

Same axioms, but add typing and typing rules (the definition can also be based on category theory).

1. $R_{\mathsf{S},\mathsf{T}} : \mathsf{S} \leftrightarrow \mathsf{T}$ (relation $R_{\mathsf{S},\mathsf{T}}$ has type $\mathsf{S} \leftrightarrow \mathsf{T}$).
2. $Q_{\mathsf{S},\mathsf{T}} \sqcup R_{\mathsf{S},\mathsf{T}} : \mathsf{S} \leftrightarrow \mathsf{T}$.
3. $\overline{R_{\mathsf{S},\mathsf{T}}} : \mathsf{S} \leftrightarrow \mathsf{T}$.
4. $Q_{\mathsf{S},\mathsf{T}} ; R_{\mathsf{T},\mathsf{U}} : \mathsf{S} \leftrightarrow \mathsf{U}$.
5. $R^{\smile}_{\mathsf{S},\mathsf{T}} : \mathsf{T} \leftrightarrow \mathsf{S}$.
6. $Q_{\mathsf{S},\mathsf{T}} \sqcup R_{\mathsf{U},\mathsf{V}}$ is defined only if $\mathsf{S} = \mathsf{U}$ and $\mathsf{T} = \mathsf{V}$.
7. $Q_{\mathsf{S},\mathsf{T}} ; R_{\mathsf{U},\mathsf{V}}$ is defined only if $\mathsf{T} = \mathsf{U}$.
8. There are constants $\mathbb{I}_{\mathsf{S},\mathsf{S}}$, $\bot_{\mathsf{S},\mathsf{T}}$, $\top_{\mathsf{S},\mathsf{T}}$ for each S and T.

Laws that can be derived

Most laws derivable in the homogeneous setting are also derivable in the heterogenous setting, but there are exceptions.

Example

- Homogenous RA law: $\top ; \top = \top$.

  Proof

  $$\top \;=\; \mathbb{I} ; \top \;\subseteq\; \top ; \top \;\subseteq\; \top$$

- Heterogenous RA: $\top_{\mathsf{S},\mathsf{T}} ; \top_{\mathsf{T},\mathsf{U}} = \top_{\mathsf{S},\mathsf{U}}$ cannot be derived. The above proof cannot be used:

  $$\top_{\mathsf{S},\mathsf{U}} \;=\; \mathbb{I}_{\mathsf{S},\mathsf{S}} ; \top_{\mathsf{S},\mathsf{U}} \;\not\subseteq\; \top_{\mathsf{S},\mathsf{T}} ; \top_{\mathsf{T},\mathsf{U}} \;\subseteq\; \top_{\mathsf{S},\mathsf{U}}.$$

  And there is a counterexample. The only relation between sets S and $\emptyset$ is

  $$\top_{S,\emptyset} = \bot_{S,\emptyset}$$

  Thus $\top_{S,\emptyset} ; \top_{\emptyset,S} = \bot_{S,\emptyset} ; \bot_{\emptyset,S} = \bot_{S,S} \neq \top_{S,S}$ unless $S = \emptyset$.

4 References

[1] C. Brink, W. Kahl and G. Schmidt (eds.). *Relational Methods in Computer Science*, Springer, 1997.

[2] R. D. Maddux. *Relation Algebras*, Studies in Logic and the Foundations of Mathematics. Elsevier, 2006.

[3] G. Schmidt and T. Ströhlein. *Relations and Graphs*, EATCS Monographs in Computer Science (Springer-Verlag, Berlin, 1993).

[4] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.

Basics of Modal Kleene Algebra

Georg Struth

Department of Computer Science
University of Sheffield, United Kingdom
G.Struth@dcs.shef.ac.uk

Basics of Modal Kleene Algebra

Georg Struth

University of Sheffield

based on joint work with Jules Desharnais, Bernhard Möller and others

Motivation

program/system analysis requires formalisms that balance

- expressive interoperable modelling languages
- powerful proof procedures

modelling languages: e.g.

- relations used in Z or B
- functions/quantales used in refinement calculi
- modal logics/process algebras used for reactive/concurrent systems

proof procedures dominated by

- interactive proof checking
- model checking

Motivation

questions: is there formalism that offers better balance

- unifies/integrates relational, functional, modal reasoning?
- allows off-the-shelf automated theorem provers?

Motivation

questions: is there formalism that offers better balance

- unifies/integrates relational, functional, modal reasoning?
- allows using off-the-shelf automated theorem provers?

answer: modal Kleene algebras (maybe)

benefits of algebraic approach:

- simple first-order equational calculus
- rich class of computationally meaningful models
- mechanisms for abstraction and (de)composition
- suitable for ATP systems

This Tutorial

goal: introduce modal Kleene algebras as computational tools for modelling and analysing discrete dynamical systems

outline:

1. surveys foundations of modal Kleene algebras
2. discusses some computationally interesting models
3. sketches connection with computational logics
4. presents some (automation) examples

dual rôle of ATP: a new approach to

- computer mathematics: develop/analyse algebraic structures
- formal methods: develop/analyse programs and systems

apology: highly subjective and incomplete picture

Semirings, Actions and Propositions

semirings: $(S, +, \cdot, 0, 1)$ "ring without minus"

$$x + (y + z) = (x + y) + z \qquad x + y = y + x \qquad x + 0 = x$$
$$x(yz) = (xy)z \qquad x1 = x \qquad 1x = x$$
$$x(y + z) = xy + xz \qquad (x + y)z = xz + yz$$
$$x0 = 0 \qquad 0x = 0$$

interpretation: S represents actions of some discrete dynamical system

- $+$ models nondeterministic (angelic) choice (cf. next slide)
- $\cdot$ models sequential composition
- 0 models abortive action
- 1 models ineffective action

Semirings, Actions and Propositions

remarks:

- swapping multiplication yields opposite semiring
- semiring is idempotent if $x + x = x$
- idempotent semirings are naturally ordered by $x \leq y \Leftrightarrow x + y = y$ hence $(S, +, 0)$ is upper semilattice with least element 0
- idempotency turns addition into choice

questions:

- how can the state space of the system be included?
- how can the "limit behaviour" of the system be described?

Semirings, Actions and Propositions

task: include the state space

test algebras: [ManesArbib] "Boolean centre"

- Boolean subalgebra $(\mathsf{test}(S), +, \cdot, \neg, 0, 1)$ embedded into $[0, 1]$ of S
- $+$ coincides with Boolean join
- $\cdot$ coincides with Boolean meet

remarks:

- Boolean algebra $\mathsf{test}(S)$ captures the main intuition behind state spaces
- elements of $\mathsf{test}(S)$ are sets of states
- alternative interpretations as propositions of a system or tests of a program

notation: $x, y, z \ldots$ for actions; $p, q, r, \ldots$ for tests/propositions

Kleene Algebras

task: describe "limit behaviour"

Kleene algebras: [Kozen] idempotent semiring with star satisfying

- unfold axiom $1 + xx^* \leq x^*$
- induction axiom $y + xz \leq z \Rightarrow x^*y \leq z$
- and their opposites

aside: KAs capture while-programs/guarded commands in various semantics

$$\ldots$$
$$\text{if } p \text{ then } x \text{ else } y = px + \neg py$$
$$\text{while } p \text{ do } x = (px)^*\neg p$$

Adding Modalities

motivation:

- many applications require different approach to actions/propositions
- systems dynamics is often modelled via state transitions;
  i.e. mappings from states to states
- various logics "use" Kleene algebras, but what is the precise connection?

idea: modal approach

- actions/propositions via Kripke frames
- system dynamics via images/preimages $\langle x|p \,/\, |x\rangle p$
- preimages via axiomatisation of domain
- images via axiomatisation of codomain

Domain Semirings

general idea:

- axiomatise domain as mapping $d : S \to S$ on semiring S
- $d(x)$ models states at which action x is enabled
- $d(x)$ should be
  - ≤ 1
  - least left preserver of x: $\quad x \leq px \Leftrightarrow d(x) \leq p$
    where px models restriction of action x to states in p
- equational axioms would be nice

question: what would be the type of p?

Domain Semirings

domain semiring: semiring with mapping $d : S \to S$ that satisfies

$$x + d(x)x = d(x)x \qquad d(xy) = d(xd(y)) \qquad d(x + y) = d(x) + d(y)$$
$$d(x) + 1 = 1 \qquad d(0) = 0$$

some intuition:

- axiom 1: $x \leq d(x)x$ means that domain is a left preserver
- axiom 2: $d(xy)$ is local on y through its domain
- axiom 3: enabling a choice means enabling one alternative or the other
- axiom 4: domain is smaller than 1 (cf. next slide)
- axiom 5: the abortive action is never enabled

Domain Semirings

property: every domain semiring is automatically idempotent

further properties: the axioms

- are irredundant (use model generator Mace4)
- cannot be weakened to inequalities (Mace4)
- imply least left preservation
- imply many "natural properties" (cf. next slides)

domain elements: $d(x) = x$ says "x is domain element"

Properties of Domain

fact: Let S be a domain semiring. Let $x, y \in S$ and let $p \in d(S)$. Then

- $d(x)x = x$ (domain is a left invariant)
- $d(p) = p$ (domain is a projection)
- $d(xy) \leq d(x)$ (domain increases for prefixes)
- $x \leq 1 \Rightarrow x \leq d(x)$ (domain expands subidentities)
- $d(x) = 0 \Leftrightarrow x = 0$ (domain is very strict)
- $d(1) = 1$ (domain is co-strict)
- $x \leq y \Rightarrow d(x) \leq d(y)$ (domain is isotone)
- $d(px) = pd(x)$ (domain elements can be exported)
- $d(x)d(x) = d(x)$ (domain elements are multiplicatively idempotent)
- $d(x)d(y) = d(y)d(x)$ (domain elements commute)
- $x \leq px \Leftrightarrow d(x) \leq p$ (domain elements are least left-preservers)
- $xy = 0 \Leftrightarrow xd(y) = 0$ (domain is weakly local)

Domain Algebra

question: how can we relate domain elements with tests?

property: for every domain semiring S, the sub-structure $(d(S), +, \cdot, 0, 1)$ is a bounded distributive lattice

proof: (with ATP)

1. check closure properties, $d(1) = 1$ and $d(0) = 0$
2. this gives sub-semiring
3. $d(x) \leq 1$ is axiom and $d(x)d(x) = d(x)$
4. but semirings satisfying these two properties are distributive lattices [Birkhoff]

notation:

- $(d(S), +, \cdot, 0, 1)$ is called domain algebra of S
- $p, q, r \ldots$ for domain elements

Domain Algebra

question: how can we enrich the domain algebra?

answer: (examples)

1. Heyting algebra: add Galois connection (and closure condition for $\rightarrow$)

$$pq \leq r \Leftrightarrow p \leq q \rightarrow r$$

2. Boolean algebra: add antidomain operation $a : S \rightarrow S$ with axioms

$$d(x) + a(x) = 1 \qquad d(x)a(x) = 0$$

Boolean Domain Algebra

assume: semiring that satisfies the domain/antidomain axioms

consequence: $d(S)$ is the largest Boolean subalgebra of S, so

$$d(S) = \mathsf{test}(S)$$

properties: (ATP)

- $a^2(x) = d(x)$
- $a(x)$ is greatest left annihilator of x: $px = 0 \Leftrightarrow p \leq a(x)$

consequence:

- d can be replaced by a^2
- many domain/antidomain axioms become redundant
- axiomatisation can be simplified
- this yields. . .

Boolean Domain Semirings

Boolean domain semiring: semiring S with mapping $a : S \to S$ that satisfies

$$a(x)x = 0 \qquad a(xy) \leq a(xa^2(y)) \qquad a^2(x) + a(x) = 1$$

remarks:

- ATP/model search is very helpful in this development
- simple axioms induce rich modal calculus. . .

Modal Semirings

idea: define forward/backward diamonds as preimages/images

$$|x\rangle p = d(xp) \qquad \langle x|p = d^\circ(px)$$

where antidomain operation d° is dual of domain

consequence:

- we have $|x\rangle 0 = 0$ and $|x\rangle(p + q) = |x\rangle p + |x\rangle q$
- this yields
  - distributive lattices with operators
  - Heyting algebras with operators
  - Boolean algebras with operators

convention: we will call KAs with Boolean domain modal KAs (MKAs)

Modalities, Symmetries, Dualities for Boolean Domain

demodalisation: $|x\rangle p \leq q \Leftrightarrow \neg q x p \leq 0 \qquad \langle x|p \leq q \Leftrightarrow px\neg q \leq 0$

dualities:

- de Morgan: $|x]p = \neg|x\rangle\neg p \qquad [x|p = \neg\langle x|\neg p$
- opposition: $\langle x|, [x| \Leftrightarrow |x\rangle, |x]$

symmetries:

- conjugation: $(|x\rangle p)q = 0 \Leftrightarrow p(\langle x|q) = 0$
- Galois connection: $|x\rangle p \leq q \Leftrightarrow p \leq [x|q$

benefits: rich calculus (automatically verified)

- symmetries as theorem generators
- dualities as theorem transformers

Kleene Modules

Kleene module: [Leiß06] structure $(K, L, :)$ with

$$(x+y)p = xp + xp \qquad x(p+q) = xp + xq \qquad (xy)p = x(yp)$$
$$1p = p \qquad x0 = 0 \qquad xp + q \leq p \Rightarrow x^*q \leq p$$

remark: scalar product : omitted

fact: modal Kleene algebras are Kleene modules with $: = \lambda x \lambda p.|x\rangle p$

consequence: close relationship with computational logics

Models

trace: alternating sequence $p_0 a_0 p_1 a_1 p_2 \dots p_{n-2} a_{n-1} p_{n-1}$, $p_i \in P$, $a_i \in A$

trace product: $\sigma.p \cdot p.\sigma' = \sigma.p.\sigma'$ $\qquad$ $\sigma.p \cdot q.\sigma'$ undefined

fact: power-set algebra $2^{(P,A)^*}$ forms (full trace) MKA

$$T_0 \cdot T_1 = \{\tau_0 \cdot \tau_1 : \tau_0 \in T_0, \tau_1 \in T_1 \text{ and } \tau_0 \cdot \tau_1 \text{ defined}\}$$
$$T^* = \{\tau_0 \cdot \tau_1 \cdot \dots \cdot \tau_n : n \geq 0, \tau_i \in T \text{ and prods defined}\}$$
$$|T\rangle Q = \{p : p.\sigma.q \in T \text{ and } q \in Q\}$$

trace MKA: complete subalgebra of full trace MKA

Models

special cases: essentially by forgetting structure in trace MKA

- path/language MKAs forget actions/propositions
- relation MKAs forget sequences between endpoints

property: (equational) properties are inherited by (relations), paths, languages

further models:

- functions/predicate transformers form weaker Kleene algebras
- matrices over Kleene algebras [Conway/Kozen]

MKAs and Propositional Dynamic Logic

fact: MKAs are dynamic/test algebras

proof:

- dynamic algebras are almost Kleene modules
- main task is to show equivalence of
  - module induction law $|x\rangle p + q \leq p \Rightarrow |x^*\rangle q \leq p$
  - Segerberg axiom $|x^*\rangle p - p \leq |x^*\rangle(|x\rangle p - p)$

extensionality: $(\forall p.|x\rangle p = |y\rangle p) \Rightarrow x = y$

intuition: extensionality forces Kripke-style models

corollary: extensional MKAs are essentially propositional dynamic logics

MKAs and Propositional Dynamic Logic

benefits: MKA offers

- simpler/more modular axioms
- richer model class (beyond Kripke frames)
- more flexible setting

perspective:

- simple automated reasoning about programs and systems with off-the-shelf ATP systems
- easily extendable to the automation of first-order variants, e.g.,

$$\exists x \forall p \exists q.(|x\rangle f(p) \leq |x\rangle g(q) \rightarrow |x]h(p,q) = 0)$$

- some temporal logics and Hoare logics subsumed

MKAs and Linear Temporal Logic

encoding:

- temporal operators (use one single action x)

$$Xp = |x\rangle p \qquad Fp = |x^*\rangle p \qquad Gp = |x^*]p \qquad pUq = |(px)^*\rangle q$$

- initial state $\mathsf{init}_x = [x|0$ "there's nothing before the beginning"
- validity of temporal implications $\sigma \models p \rightarrow q \Leftrightarrow \mathsf{init}_x \cdot p = q$
- tests now model sets of traces and x models the abstract tail relation

MKAs and Linear Temporal Logic

LTL axioms: von Karger's variant of [Manna/Pnueli]

$$|(px)^*\rangle q = q + p|x\rangle|(px)^*\rangle q \qquad \langle(xp)^*|q = q + p\langle(xp)^*|\langle x|q$$
$$|(px)^*\rangle 0 \leq 0 \qquad \langle x|0 = 1$$
$$|x^*](p \rightarrow q) \leq |x^*]p \rightarrow |x^*]q \qquad [x^*|(p \rightarrow q) \leq [x^*|p \rightarrow [x^*|q$$
$$|x^*]p \leq p|x]|x^*]p \qquad |x^*](p \rightarrow |x]p) \leq |x^*](p \rightarrow |x^*]p)$$
$$p \leq [x||x\rangle p \qquad p \leq |x]\langle x|p$$
$$\mathsf{init}_x \leq |x^*](p \rightarrow [x|q) \rightarrow |x^*](p \rightarrow [x^*|q) \qquad \mathsf{init}_x \leq |x^*]p \rightarrow |x^*][x|p$$
$$|x](p \rightarrow q) = |x]p \rightarrow |x]q \qquad [x|(p \rightarrow q) = [x|p \rightarrow [x|q$$
$$\langle x|p \leq [x|p \qquad |x\rangle p = |x]p$$

MKAs and Linear Temporal Logic

fact:

1. blue axioms are theorems of MKA
2. axioms express linearity of models (in MKA)

benefits:

- reasoning about infinite-state systems possible
- first-order temporal reasoning
- trace model available

remark:

- CTL also subsumed
- CTL* needs additional fixed points (and quantale-based setting)

MKAs and Hoare Logic

fact: MKA subsumes (propositional) Hoare logic

explanation: this is Hoare logic without the assignment rule

example: validity of while rule $\vdash_{\mathsf{MKA}} \langle x|pq \leq q \Rightarrow \langle (px)^*\neg p|q \leq \neg pq$

benefits:

- weakest liberal precondition semantics for free in MKA ($\mathsf{wlp}(x,p) = |x]p$)
- soundness and completeness of Hoare logic are easy in MKA
- formalism of Hoare logic is dissolved in modal setting
- relative completeness not an issue. . .

MKAs and Hoare Logic

example: validity of while rule

$$\begin{aligned} \langle a|pq \leq q &\Leftrightarrow \langle pa|q \leq q && (\text{ contravariance }) \\ &\Rightarrow \langle (pa)^*|q \leq q && (\text{ induction }) \\ &\Rightarrow \langle \neg p|\langle (pa)^*|q \leq \langle \neg p|q && (\text{ isotonicity }) \\ &\Leftrightarrow \langle (pa^*)\neg p|q \leq \neg pq && (\text{ contravariance }) \end{aligned}$$

perspective:

- full automation of Hoare logic seems possible
- assignment rule requires formalising substitution
- handling numbers or data types is so far difficult for ATP systems
- approach extends to total correctness

Divergence and Termination

∇**-Kleene module:** Kleene module $(K, L, :)$ with divergence $^\nabla : K \to L$ satisfying

- ∇-unfold $\quad x^\nabla \leq xx^\nabla$
- ∇-coinduction $\quad p \leq xp + q \Rightarrow p \leq x^\nabla + x^*q$

remark: scalar product symbol omitted

interpretation:

1. for modal Kleene algebra, x^∇ denotes those states from which infinite behaviour may start
2. if K models finite actions and L infinite actions, then x^∇ is the infinite iteration of finite action x

Divergence and Termination

fact: if L is Boolean algebra, then ∇-coinduction is equivalent to

$$p \leq xp \Rightarrow p \leq x^\nabla$$

final part: $\mathsf{max}_x(p) = p - xp$ models final part of p w.r.t. x

termination: action x terminates if $x^\nabla = 0$

property: if L is Boolean algebra, then x terminates iff

$$\mathsf{max}_x(p) = 0 \Rightarrow p = 0$$

remark: this captures set-theoretic notion of Noethericity

Divergence and Termination

trace model:

- let K be a trace Kleene algebra
- let L be a set of infinite traces under union
- define, for $\tau \in K$ and $\pi \in L$ the scalar product $\tau : \pi$ like product of finite traces
- lift that product to sets of traces
- define $x^\nabla = \{\pi \in L : \pi = \tau_0 \cdot \tau_1 \cdot \ldots \text{ with } \tau_i \in K \text{ for } i \geq 0\}$

Then $(K, L, :, ^\nabla)$ is a (full trace) ∇-Kleene module

special cases: path and language ∇-Kleene modules

consequence: ∇-Kleene modules useful for integrated finite/infinite behaviour

Divergence and Termination

fact: divergence and termination can be equationally axiomatised

- $p \leq x^\nabla + x^* \mathsf{max}_x(p)$ is equivalent to ∇-coinduction
- $p \leq x^* \mathsf{max}_x(p)$ is equivalent to termination

remark: L must be Boolean algebra

intuition: p either leads to divergence or to final states after a finite iteration

perspective:

- characterisation dual to Segerberg's axiom
- equational approach to finite and infinite behaviours of discrete dynamical systems
- very suitable for ATP systems (see below)

Domain on Sub-Semirings

near-semiring: structure $(S, +, \cdot)$ such that

- $(S, +)$ and $(S, \cdot)$ are semigroups
- right distributivity law $(x + y)z = xz + yz$ holds

pre-semiring: left pre-isotone near-semiring $x + y = y \Rightarrow zx + zy = zy$

units: 0, 1 or

- deadlock $x + \delta = x \quad \delta x = \delta$.
- silent action $x\tau = x$

Domain on Sub-Semirings

basic process algebra: idempotent near-semiring $(S, +, \cdot, ^*)$ or $(S, +, \cdot, ^*, \delta, \tau)$

game algebra: idempotent pre-semiring $(S, +, \cdot, 0, 1)$

probabilistic Kleene algebra: idempotent pre-semiring $(S, +, \cdot, ^*, 0, 1)$

demonic refinement algebra: idempotent semiring $(S, +, \cdot, ^*, ^\infty, \delta, 1)$

Domain on Sub-Semirings

	$\mathsf{NS}^{\tau}_{\delta}$	NS^{1}_{δ}	PS^{1}_{δ}
$a(x)x = \delta$		$\surd$	$\surd$
$a(xy) \leq a(xa^2(y))$		$\surd$	$\surd$
$a^2(x) + a(x) = 1$	$\surd$	$\surd$	$\surd$
$a(x+y) = a(x)a(y)$		$\surd$	
$x = d(x)x$	$\surd$		
$d(xy) = d(xd(y))$	$\surd$		
$d(x+y) = d(x) + d(y)$	$\surd$		
$d(\delta) = \delta$	$\surd$		
$d(x)d(y) = d(y)d(x)$	$\surd$		
$d(a(x)) = a(x)$	$\surd$		

NS: near-semiring, PS: pre-semiring

Domain on Sub-Semirings

conclusion:

- domain can still be defined on sub-semirings
- this models enabledness conditions for games, processes and actions in protocols
- semiring domain axioms suffice for probabilistic Kleene algebras and demonic refinement algebras
- domain does not induce modal operators

Automation Examples

observation: ATP system have rather been neglected in formal methods

idea: combine MKAs with ATPs and counter example generators

results: experiments with various ATPs (Prover9, SPASS, Waldmeister,. . .)

- ~ 500 theorems automatically proved
- successful case studies in program refinement, termination analysis

benefits:

- special-purpose calculi made redundant
- generic flexible library of lemmas
- new style of verification

Automating Bachmair and Dershowitz's Termination Theorem

theorem: [BachmairDershowitz86] *termination of the union of two rewrite systems can be separated into termination of the individual systems if one rewrite system quasicommutes over the other*

formalisation: ∇-Kleene module over semilattice

encoding:

- quasicommutation $yx \leq x(x+y)^*$
- separation of termination $(x+y)^{\nabla} = 0 \Leftrightarrow x^{\nabla} + y^{\nabla} = 0$

statement: termination of x and y can be separated if x quasicommutes over y

remark: posed as challenge by Ernie Cohen in 2001

Automating Bachmair and Dershowitz's Termination Theorem

results: SPASS finds an extremely short proof in < 5min

$$\begin{aligned}
(x+y)^\nabla &= y^\nabla + y^* x (x+y)^\nabla && \text{(sum unfold)}\\
&\leq y^\nabla + x(x+y)^*(x+y)^\nabla && \text{(strong quasicommmutation)}\\
&= y^\nabla + x(x+y)^\nabla && \text{(since } z^\omega = z^* z^\omega\text{)}\\
&\leq x^\nabla + x^* y^\nabla && \text{(coinduction)}\\
&= 0 && \text{(assumption } x^\nabla = y^\nabla = 0\text{)}
\end{aligned}$$

Automating Bachmair and Dershowitz's Termination Theorem

surprise: proof reveals new refinement law

$$yx \leq x(x+y)^* \Rightarrow (x+y)^\nabla = x^\nabla + x^* y^\nabla$$

for separating infinite loops

remarks:

- reasoning essentially coinductive
- theorem holds in large class of models
- translation safe since relations form ∇-Kleene modules

Automating the DBW-Theorem

lazy commutation: $yx \leq x(x+y)^* + y$

theorem: [Doornbos/Backhouse/van der Woude]
if x lazily commutes over y then termination of x and y can be separated

comment: this generalisation is much more difficult

lemma: x lazily commutes over y iff

$$yx^* \leq x(x+y)^* + y$$

proof: $44.23s$ by Prover9.

Automating the DBW-Theorem

proof: (non-trivial direction of DBW-theorem)

1. abbreviate $\nabla = (x+y)^\nabla$
2. assume that x and y terminate
3. for $\nabla = 0$ it suffices to show $\mathsf{max}_y(\mathsf{max}_x(\nabla)) = 0$
4. this is equivalent to $\mathsf{max}_x(\nabla) \leq y\mathsf{max}_x(\nabla)$
5. we calculate

$$\begin{aligned}
\nabla &= x\nabla + y\nabla \leq x\nabla + yx^*\mathsf{max}_x(\nabla) \leq x\nabla + x(x+y)^*\mathsf{max}_x(\nabla) + y\mathsf{max}_x(\nabla)\\
&\leq x\nabla + x(x+y)^*\nabla + y\mathsf{max}_x(\nabla) = x\nabla + y\mathsf{max}_x(\nabla)
\end{aligned}$$

6. the claim now follows from the Galois connection for complementation and the definition of max_x

remark: the second step uses the equational characterisation of termination

Automating the DBW-Theorem

remarks:

- proof is much more compact than previous approaches
- for the first time in first-order setting
- theorem holds again in large model class
- main calculation could again be automated
- full automation remains a challenge

Automating a Modal Correspondence Result

modal logic: Löb's formula $\Box(\Box p \to p) \to \Box p$

translation to MKA/Kleene modules: $xp \leq x(p - xp) = x\mathsf{max}_x(p)$

intuition: all states with transitions into p are states from which no further transitions are possible

remark: this would correspond to Noethericity if x is transitive ($xx \leq x$)

reminder: two equivalent characterisations of Noethericity

- $p \leq x^*\mathsf{max}_x(p)$ (x pre-Löbian)
- $\mathsf{max}_x(p) = 0 \Rightarrow p = 0$ (x Noetherian)

Automating a Modal Correspondence Result

property: for every x in some ∇-Kleene module

(i) x Löbian $\Rightarrow$ x Noetherian
(ii) x Noetherian $\Leftrightarrow$ x pre-Löbian (see above)
(iii) x pre-Löbian and $x = xx$ $\Rightarrow$ x Löbian

proofs: with Prover9 in ∇-Kleene algebra

(i) $\leq 4s$
(ii) $\leq 4s$ and $\leq 20s$ (hypothesis learning)
(iii) $\leq 1s$ (hypothesis learning)

remark: this is a modal correspondence result

- Noethericity corresponds to frame property
- proof is calculational and automated
- model theory is normally used

Automating Hoare Logic

algorithm: integer division n/m

```
fun DIV = k:=0;l:=n;
          while m<=l do k:=k+1;l:=l-m;
```

precondition: $0 \leq n$

postconditions: $n = km + l$ $0 \leq l$ $l < m$

proof goal: $\langle x_1x_2(ry_1y_2)^*\neg r|p \leq q_1q_2\neg r$

Automating Hoare Logic

proof: two phases coupled by assignment rule $p[e/x] \leq |\{x := e\}]p$

1. MKA: goal follows from $p \leq |x_1]|x_2](q_1 q_2) \qquad q_1 q_2 r \leq |y_1]|y_2](q_1 q_2)$
   (automated with Prover9)
2. arithmetics: subgoals must still be manually verified, e.g.,

$$\begin{aligned} |x_1]|x_2](q_1 q_2) &= |\{k := 0\}]\,|\{l := n\}](q_1 q_2) \geq (\{n = km + l\}\{0 \leq l\})[k/0][l/n] \\ &= \{n = 0m + n\}\{0 \leq n\} = \{0 \leq n\} \\ &= p \end{aligned}$$

remark:

- reasoning essentially inductive
- domain specific solvers should be integrated ATPs
- try SPASS+T?

Newman's Lemma: A Proof Challenge

Newman's lemma: *A term rewriting system is confluent if it is locally confluent and terminating.*

generalisation and translation:

- x commutes over $y \quad y^* x^* \leq x^* y^*$
- x locally commutes over $y \quad yx \leq x^* y^*$

theorem: In ∇-Kleene algebra, if $x + y$ terminates and y locally commutes over y, then y commutes over y

Newman's Lemma: A Proof Challenge

proof: (so far)

- one page of semi-calculational arguments
- main calculation

$$\begin{aligned} \langle y^*|\langle y|\langle p\rangle|x\rangle|x^*\rangle &\leq \langle y^*|\langle p_y\rangle\langle y||x\rangle\langle p_x\rangle|x^*\rangle \\ &\leq \langle y^*|\langle p_y\rangle|x^*\rangle\langle y^*|\langle p_x\rangle|x^*\rangle \\ &\leq \langle y^*|\langle p_y\rangle|x^*\rangle|x^*\rangle\langle y^*| \\ &\leq \langle y^*|\langle p_y\rangle|x^*\rangle\langle y^*| \\ &\leq |x^*\rangle\langle y^*|\langle y^*| \\ &\leq |x^*\rangle\langle y^*| \end{aligned}$$

- $p_x = \langle x|p$ and $p_y = \langle y|p$
- proof lifted to level of modal operators

Newman's Lemma: A Proof Challenge

question: can you automate this?

remarks:

- Newman's lemma seems to require a mix of inductive and coinductive reasoning
- the main calculation mimics precisely the traditional diagrammatic proof
- more generally, Kleene algebras give an algebraic semantics to (some) rewrite diagrams

A Non-Modal Example: Back's Atomicity Refinement Law

demonic refinement algebra: [von Wright04] Kleene algebra

- with axiom $x0 = 0$ dropped
- extended by strong iteration $^\infty$ encompassing finite and infinite iteration

remark: abstracted from refinement calculus [BackvonWright]

atomicity refinement law for action systems

- complex theorem first published by Back in 1989
- long proof in set theory analysing infinite sequences
- proof by hand in demonic refinement algebra still covers 2 pages
- automated analysis reveals some glitches and yields generalisation

first task: build up library of verified basic refinement laws for proof

A Non-Modal Example: Back's Atomicity Refinement Law

theorem: if (i) $s \leq sq$ (ii) $a \leq qa$ (iii) $qb = 0$ (iv) $rb \leq br$
(v) $(a + r + b)l \leq l(a + r + b)$ (vi) $rq \leq qr$ (vii) $ql \leq lq$
(viii) $r^* = r^\infty$ (ix) $q \leq 1$

then

$$s(a + r + b + l)^\infty q \leq s(ab^\infty q + r + l)^\infty$$

two-step proof with "hypothesis learning"

1. assumptions imply $s(a + r + b + l)^\infty q \leq sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty$
   wait $60s$ for 75-step proof with Prover9
2. $q \leq 1$ implies $sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty \leq s(ab^\infty q + r + l)^\infty$
   wait $< 1s$ for 30-step proof

remark: full proof succeeds for $l = 0$ ($1013s$ for 46-step proof)

A Non-Modal Example: Back's Atomicity Refinement Law

equational proof can be reconstructed

$$\begin{aligned}
s(a + b + r + l)q &= sl^\infty(a + b + r)^\infty q \\
&= sl^\infty(b + r)^\infty(a(b + r)^\infty)^\infty q \\
&= sl^\infty b^\infty r^\infty(ab^\infty r^\infty)^\infty q \\
&\leq sl^\infty b^\infty r^\infty(qab^\infty r^\infty)^\infty q \\
&= sl^\infty b^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sql^\infty b^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sl^\infty qb^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sl^\infty qr^\infty q(ab^\infty r^\infty q)^\infty \\
&= sl^\infty qr^\infty q(ab^\infty r^* q)^\infty \\
&\leq sl^\infty qr^\infty q(ab^\infty qr^*)^\infty \\
&= sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty.
\end{aligned}$$

Conclusion

this tutorial: modal Kleene algebras offer

- simple equational calculus including some (co)induction
- rich model class (traces, paths, languages, relations, functions,. . .)
- easy automation
- interesting applications in program analysis/verification
- relevant for modelling discrete dynamical systems

related work:

- automation of relation algebras similarly successful
- code at `www.dcs.shef.ac.uk/~georg/ka`
- results will be integrated into TPTP library

Conclusion

general conclusion: ATP systems + computational algebras motivates

verification challenge

- off-the-shelf ATP with domain-specific algebras
- promising alternative to conventional approaches (model checking, HOL)
- light-weight formal methods with heavy-weight automation

Seek Simplicity and Distrust It.

[Whitehead]

Some References

1. *Prover9 and Mace4*, `http://www.cs.unm.edu/~mccune/prover9`.
2. *Spass 3.0*, `http://spass.mpi-inf.mpg.de/`.
3. G. Struth. Modal Tools for Separation and Refinement Technical Report CS-08-07, Department of Computer Science; University of Sheffield, 2008.
4. J. Desharnais and G. Struth. Enabledness Conditions for Action Systems, Probabilistic Systems, and Processes Technical Report CS-08-06, Department of Computer Science; University of Sheffield, 2008.
5. J. Desharnais and G. Struth. Modal Semirings Revisited. (Accepted for MPC 2008).
6. B. Möller. Kleene getting lazy. *Science of Computer Programming*, 65(2):195–214, 2007.
7. L. Meinicke and K. Solin. Refinement algebra for probabilistic programs. In E. Boiten, J. Derrick, and G. Smith, editors, *Refine 2007*, ENTCS, 2007. To appear.
8. G. Struth. Reasoning Automatically about Termination and Refinement. In S. Ranise, editor, *6th International Workshop on First-Order Theorem Proving*, Technical Report ULCS-07-018, Department of Computer Science, University of Liverpool, pages 36–51. 2007.
9. P. Höfner and G. Struth. Automated Deduction in Kleene Algebra. In P. Pfenning, editor, *21th International Conference on Automated Deduction (CADE21)*, volume 4603 of *LNAI*, pages 279–294. Springer, 2007.
10. J.-L. De Carufel and J. Desharnais. Demonic algebra with domain. In R. A. Schmidt, editor, *Relations and Kleene Algebras in Computer Science*, volume 4136 of *LNCS*, pages 120–134. Springer, 2006.
11. K. Solin and J. von Wright. Refinement algebra with operators for enabledness and termination. In T. Uustalu, editor, *Mathematics of Program Construction*, volume 4014 of *LNCS*, pages 397–415. Springer, 2006.
12. P. Höfner, B. Möller and G. Struth. Quantales and temporal logics. In M. Johnson and V. Vene, editors, *Algebraic Methodology and Software Technology, 11th International Conference*, volume 4019 of *LNCS*, pages 262–277. Springer, 2006.
13. J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transations on Computational Logic*, 7(4):798–833, 2006.
14. B. Möller, and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.
15. B. Möller and G. Struth. wp is wlp. In I. Düntsch, W. MacCaull and M. Winter, editors, *Relational Methods in Computer Science*, volume 3929 of *LNCS*, pages 200–211. Springer, 2005.

16. J. Desharnais, B. Möller, and G. Struth. Modal Kleene algebra and applications—a survey. *Journal on Relational Methods in Computer Science*, 1:93–131, 2004.(Invited contribution).
17. J. Desharnais, B. Möller, and G. Struth. Termination in modal Kleene algebra. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *IFIP TCS2004*, pages 647–660. Kluwer, 2004. Revised version: *Algebraic Notions of Termination*. Technical Report 2006-23, Institut für Informatik, Universität Augsburg, 2006.

Basics of Preference and Fuzzy Preference Modeling

Susanne Saminger-Platz

Department of Knowledge-Based Mathematical Systems
Johannes Kepler University, Linz, Austria
susanne.saminger-platz@jku.at

Basics of Preference and Fuzzy Preference Modeling

Susanne Saminger-Platz

[1]Department of Knowledge-Based Mathematical Systems
Johannes Kepler University Linz

http://www.flll.jku.at
susanne.saminger-platz@jku.at

RelMiCS10, AKA5, April 07, 2008

Motivation

Decision problems (Vincke, 1992)

A multicriteria decision problem is a situation in which, having defined a set of alternatives A and a consistent family of criteria G on A, one wishes

1. to determine a subset of alternatives considered to be best w.r.t. G (*choice problem*),
2. to divide A into subsets according to some norm (*sorting problem*),
3. to rank the alternatives of A from best to worst (*ranking problem*),
4. or a mixture of all these.

Motivation (cont'd)

Criterion

A criterion G is a mapping from A to some (quantitative or qualitative) valuation set E.

It represents the decision maker's preference over the alternatives w.r.t. to some viewpoint or objective.

However, it might be that decision maker is not able to provide some criterion him/herself, but is able

- to compare all alternatives pairwise and
- to provide some information about his/her personal preferences between each single pair of alternatives.

Preference modeling

Preference modeling aims at

- providing a formal representation of the decision maker's internal preference structure and
- determining conditions under which this structure can be represented numerically, i.e., by some criterion.

The decision maker's preferences between two alternatives are represented by

- binary relations or

  ⇢ Preference modeling

- valued binary relations.

  ⇢ Fuzzy preference modeling

Part I

Preference Modeling

Binary relations — basics

A binary relation S on the set A is a subset of the Cartesian product $A \times A$. If two elements $a, b \in A$ are in relation S we will denote that by $(a, b) \in S$ or aSb.

Let R, S be binary relations on A, then the following are also binary relations on A:

- Complement S^C: $(a, b) \in S^C :\Leftrightarrow (a, b) \notin S$,
- Converse S^t: $(a, b) \in S^t :\Leftrightarrow (b, a) \in S$,
- Intersection $R \cap S$: $(a, b) \in R \cap S :\Leftrightarrow (a, b) \in S \wedge (a, b) \in R$,
- Composition $R \circ S$:

$$(a, b) \in R \circ S :\Leftrightarrow \exists c \in A : (a, c) \in R \wedge (c, b) \in S.$$

Basic properties of binary relation

Let S be a binary relation on A.

- S is *reflexive*: $\forall a \in A : (a, a) \in S$,
- S is *irreflexive*: $\forall a \in A : (a, a) \notin S$.
- S is *symmetric*: $\forall a, b \in A : (a, b) \in S \Rightarrow (b, a) \in S$,
- S is *antisymmetric*: $\forall a, b \in A : (a, b) \in S \wedge (b, a) \in S \Rightarrow a = b$,
- S is *asymmetric*: $\forall a, b \in A : (a, b) \in S \Rightarrow (b, a) \notin S$.

Basic properties of binary relation (cont'd)

Let S be a binary relation on A.

- S is *transitive*: $(S \circ S \subseteq S)$

$$\forall a, b, c \in A : (a, b) \in S \wedge (b, c) \in S \Rightarrow (a, c) \in S,$$

- S is *negatively transitive*:

$$\forall a, b, c \in A : (a, c) \in S \Rightarrow (a, b) \in S \vee (b, c) \in S,$$

$$\forall a, b, c \in A : (a, b) \notin S \wedge (b, c) \notin S \Rightarrow (a, c) \notin S,$$

- S is *semitransitive*:

$$\forall a, b, c, d \in A : (a, b) \in S, (b, c) \in S \Rightarrow (a, d) \in S \vee (d, c) \in S.$$

Basic properties of binary relation (cont'd)

Let S be a binary relation on A.

- S is an *equivalence* relation if it is reflexive, symmetric and transitive.
- S is a *strict partial order* if it is asymmetric and transitive.
- S is a *partial order* if it is reflexive, antisymmetric and transitive.
- S is a *partial preorder (or quasi order)* if it is reflexive and transitive.

Basic properties of binary relation (cont'd)

Let S be a binary relation on A.

- S is *complete*: $\forall a \neq b \in A : (a,b) \in S \vee (b,a) \in S$,
- S is *strongly complete*: $\forall a, b \in A : (a,b) \in S \vee (b,a) \in S$.
- S fulfills the *Ferrers relation*:

$$\forall a,b,c,d \in A : (a,b) \in S, (c,d) \in S \Rightarrow (a,d) \in S \vee (c,b) \in S.$$

Basic properties of binary relation (cont'd)

Let S be a binary relation on A.

- S is a *strict total order* if it is asymmetric, complete and transitive.
- S is a *total order* if it is antisymmetric, strongly complete and transitive.
- S is a *strict weak order* if it is asymmetric and negatively transitive.
- S is a *total preorder (or weak order)* if it is strongly complete and transitive.

Preference relations

Pairwise comparison of elements a, b of a finite set of alternatives A.
The decision maker either

- clearly prefers one alternative to the other,
  $\dashrightarrow$ (strict) preference relation P
- feels indifferent about them, or
  $\dashrightarrow$ indifference relation I
- considers the two elements to be incomparable.
  $\dashrightarrow$ incomparability relation J

Preference structures (P, I, J)

Definition

A preference structure on a set A is a triplet (P, I, J) of binary relations on A where

- P is (irreflexive and) asymmetric,
- I is reflexive and symmetric,
- J is irreflexive and symmetric,
- $\{P, P^t, I, J\}$ forms a partition of $A \times A$.

Large preference relation R

Each preference structure (P, I, J) on a set A is characterized by a unique (reflexive) relation R defined by

$$(a, b) \in R :\Leftrightarrow (a, b) \in P \cup I.$$

The relation P is called (strict) preference relation, whereas R is called large or weak preference structure, or also characteristic relation.

$$aPb :\Leftrightarrow (a, b) \in R \wedge (b, a) \notin R$$
$$aIb :\Leftrightarrow (a, b) \in R \wedge (b, a) \in R$$
$$aJb :\Leftrightarrow (a, b) \notin R \wedge (b, a) \notin R$$
$$(P, I, J) = (R \cap (R^t)^C, R \cap R^t, R^C \cap (R^t)^C).$$

Properties of R

Consider a preference structure (P, I, J) and its corresponding large preference structure R.

Properties of R

- R is reflexive.
- R is symmetric if and only if $P = \emptyset$.
- R is not asymmetric.
- R is antisymmetric if and only if $I = \{(a, a) \mid a \in A\}$.
- R is strongly complete if and only if $J = \emptyset$.
- R is transitive if and only if P, I are transitive and if $P \circ I \cup I \circ P \subset P$.
- If R is strongly complete, then it is transitive if and only if P, I are transitive.

Particular preference structures

- Preference structures without incomparability ($J = \emptyset$).
  - The 'traditional model' — total order resp. preoder structure ⇢ true criterion
  - The 'threshold model' — total semi-order structure ⇢ semi-criterion
  - The 'interval order structures' ⇢ interval criterion.
- Preference structures with incomparabilities ($J \neq \emptyset$): traditional model ⇢ partial order resp. preorder structure.

Total order structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a total order structure if $I = \{(a, a) \mid a \in A\}$, P is transitive and $J = \emptyset$.

The following are equivalent:

- The preference structure (P, I, J) is a total order structure.
- The large preference relation R is a total order.
- There exists $g : A \to \mathbb{R}$ with $\begin{cases} aPb \Leftrightarrow g(a) > g(b), \\ g(a) = g(b) \Rightarrow a = b. \end{cases}$

  $\dashrightarrow$ true criterion

Total preorder structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a total preorder structure if I and P are transitive, and $J = \emptyset$.

The following are equivalent:

- The preference structure (P, I, J) is a total preorder structure.
- The large preference relation R is a total preorder,i.e., strongly complete and transitive.
- There exists a real-valued function g on A such that

$$\begin{cases} aPb & \Leftrightarrow g(a) > g(b), \\ aIb & \Leftrightarrow g(a) = g(b). \end{cases} \dashrightarrow \text{true criterion, utility}$$

Total preorder structure

Consider a total preorder struture (P, I, J). Then:

- I is an equivalence relation.
- P is a strict weak order.
- $P \circ I \cup I \circ P \subseteq P$.

- Utilities are unique up to strictly increasing transformations, i.e., if g represents (P, I, J) then also $t \circ g$ with t some strictly increasing mapping.
- Since A is finite, $|A| = m$, the large preference relation R can be entirely represented as path

$$a_1 R a_2 R \ldots, R a_m$$

$\dashrightarrow$ utilities from ranks.

Fixed indifference threshold

Total semiorder structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a total semiorder structure if

- $P \circ I \circ P \subseteq P$,
- $P \circ P \circ I \subseteq P$,
- $J = \emptyset$.

Properties

- P is semitransitive and Ferrers,
- $P \circ P \cap I \circ I = \emptyset$.

Fixed indifference threshold
Total semiorder structure

Consider some finite set A. A preference structure (P, I, J) is a total semiorder structure if $P \circ I \circ P \subseteq P$, $P \circ P \circ I \subseteq P$, and $J = \emptyset$.

The following are equivalent:

- The preference structure (P, I, J) is a total semiorder structure
- The large preference relation R is strongly complete and semitransitive.
- There exists a real-valued function g on A and a positive threshold q such that

$$\begin{cases} aPb & \Leftrightarrow g(a) > g(b) + q, \\ aIb & \Leftrightarrow |g(a) - g(b)| < q. \end{cases} \quad \dashrightarrow \text{semi criterion}$$

Variable indifference thresholds
Total interval order structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a total interval order structure if

- $P \circ I \circ P \subseteq P$,
- $J = \emptyset$.

Properties

- P is transitive and Ferrers.

Variable indifference thresholds
Total interval order structure

Consider some finite set A. A preference structure (P, I, J) is a total interval order structure if $P \circ I \circ P \subseteq P$ and $J = \emptyset$.

The following are equivalent:

- The preference structure (P, I, J) is a total interval order structure.
- The large preference relation R is a strongly complete Ferrers relation.
- There exists two real-valued functions g, q on A such that

$$\begin{cases} aPb & \Leftrightarrow g(a) > g(b) + q(b), \\ aIb & \Leftrightarrow g(a) \leq g(b) + q(b) \wedge g(b) \leq g(a) + q(a). \end{cases}$$

$\dashrightarrow$ interval criterion

Partial order structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a partial order structure if P is transitive and $I = \{(a, a) \mid a \in A\}$

The following are equivalent:

- The preference structure (P, I, J) is a partial order structure.
- The large preference relation R is a partial order.
- There exists a real-valued function g on A such that

$$aPb \Rightarrow g(a) > g(b).$$

Partial preorder structure

Definition

Consider some finite set A. A preference structure (P, I, J) is a partial preorder structure if I and P are transitive, and that $P \circ I \subseteq P$ and $I \circ P \subseteq P$.

The following are equivalent:

- The preference structure (P, I, J) is a partial preorder structure.
- The large preference relation R is a partial preorder.
- There exists a real-valued function g on A such that

$$\begin{cases} aPb & \Rightarrow g(a) > g(b), \\ aIb & \Rightarrow g(a) = g(b). \end{cases}$$

Part II

Fuzzy preference modeling

Classical preference structures revisited

A preference structure on a set A is a triplet (P, I, J) of binary relations on A where

- P, J are irreflexive, I is reflexive,
- P is asymmetric, I, J are symmetric,
- $P \cap I = \emptyset$, $P \cap J = \emptyset$, $I \cap J = \emptyset$,
- $P \cup P^t \cup I \cup J = A^2$.

A preference structure on a set A is a triplet (P, I, J) of binary relations on A where

- I is reflexive and symmetric,
- for all $a, b \in A$: $P(a, b) + P^t(a, b) + I(a, b) + J(a, b) = 1$.

From crisp to fuzzy ...

Main idea: Replacing presence/absence of relationship (0 or 1) by a degree of relationship ($a \in [0, 1]$)

$$\dashrightarrow \text{valued/fuzzy binary relations}$$

Challenges:

- How to define operations on binary fuzzy relations?
- How to formulate relevant properties of binary fuzzy relations?
- What is a fuzzy preference structure?

Definition

A binary fuzzy relation R on A is a fuzzy subset of the Cartesian product A^2, i.e., R is a function from A^2 into $[0,1]$, $R\colon A^2 \to [0,1]$.

Let R_1, R_2 be two binary fuzzy relations on A. Then we define, for all $a, b \in A$,

- Complement: $R_1^C(a,b) = n(R_1(a,b))$,
- Converse: $R_1^t(a,b) = R_1(b,a)$,
- Intersection: $R_1 \cap_T R_2(a,b) = T(R_1(a,b), R_2(a,b))$,
- Union: $R_1 \cup_S R_2(a,b) = S(R_1(a,b), R_2(a,b))$,

with $n\colon [0,1] \to [0,1]$ some negation, $T, S\colon [0,1]^2 \to [0,1]$ some t-norm resp. t-conorm.

Negation

Definition

A function $n\colon [0,1] \to [0,1]$ which is decreasing and fulfills

$$n(0) = 1 \quad \text{and} \quad n(1) = 0$$

is called a negation. A negation is called strict if it is strictly decreasing and continuous. A strict negation is called strong if it is involutive, $n(n(x)) = x$ for all $x \in [0,1]$.

The standard negation N is given by $N(x) = 1 - x$.

Negation (cont'd)

Theorem (Fodor, 1993)

A function $n\colon [0,1] \to [0,1]$ is a strict negation if and only if there exist two automorphisms $\varphi, \psi\colon [0,1] \to [0,1]$ such that

$$n(x) = \varphi(1 - \psi(x)).$$

Theorem (Trillas, 1979)

A function $n\colon [0,1] \to [0,1]$ is a strong negation if and only if there exists an automorphism $\varphi\colon [0,1] \to [0,1]$ such that

$$n(x) = \varphi^{-1}(1 - \varphi(x)).$$

Triangular norms

Definition

A binary operation $T\colon [0,1]^2 \to [0,1]$ is a triangular norm (t-norm) if for all $x, y, z \in [0,1]$:

- $T(x,1) = x$,
- $T(x,z) \leq T(y,z)$ whenever $x \leq y$,
- $T(x,y) = T(y,x)$,
- $T(x, T(y,z)) = T(T(x,y), z)$.

Triangular norms

Minimum: $T_{\mathbf{M}}(x,y) = \min(x,y)$

Product: $T_{\mathbf{P}}(x,y) = x \cdot y$

Łukasiewicz t-norm: $T_{\mathbf{L}}(x,y) = \max(x+y-1, 0)$

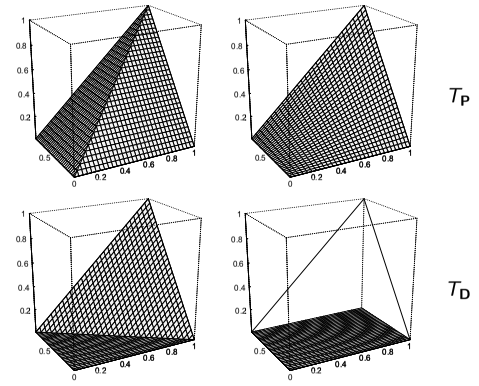
Drastic product: $T_{\mathbf{D}}(x,y) = \begin{cases} \min(x,y), & \text{if } \max(x,y) = 1, \\ 0, & \text{otherwise.} \end{cases}$

Frank t-norms: $T^{\mathbf{F}}_{\lambda}(x,y) = \log_{\lambda}\left(1 + \frac{(\lambda^x - 1)(\lambda^y - 1)}{\lambda - 1}\right)$

$\lambda \in]0,1[\cup]1,\infty[$

Basic t-norms

$T_{\mathbf{M}}$ $T_{\mathbf{P}}$



$T_{\mathbf{L}}$ $T_{\mathbf{D}}$

Basic properties of t-norms

Let $\varphi\colon [0,1] \to [0,1]$ be an increasing bijection. Then $T_{\varphi}\colon [0,1]^2 \to [0,1]$ defined by

$$T_{\varphi}(x,y) = \varphi^{-1}(T(\varphi(x), \varphi(y)))$$

is again a t-norm which is isomorphic to T.

- Nilpotent t-norms are isomorphic transforms of $T_{\mathbf{L}}$.
- Strict t-norms are isomorphic transforms of $T_{\mathbf{P}}$.

Triangular co-norms

Definition

A triangular co-norm (t-conorm) S is a binary operation on $[0,1]$ which is commutative, associative, increasing in each argument and has neutral element 0.

To t-conorm S corresponds a t-norm T via

$$S(x,y) = 1 - T(1-x, 1-y).$$

- Maximum: $S_{\mathbf{M}}(x,y) = \max(x,y)$,
- Probabilistic sum: $S_{\mathbf{P}}(x,y) = x + y - xy$,
- Łukasiewicz t-conorm: $S_{\mathbf{L}}(x,y) = \min(x+y, 1)$,
- Drastic sum $S_{\mathbf{D}}(x,y) = \begin{cases} \max(x,y), & \text{if } \min(x,y) = 0, \\ 1, & \text{otherwise.} \end{cases}$

Definition

Let T be a t-norm, S a t-conorm, and n a strict negation.
The triple (T, S, n) is called a De Morgan triple if
$n(S(x, y)) = T(n(x), n(y))$ for all $x, y \in [0, 1]$.
The triple is called continuous if T and S are continuous.

⇢ Sufficient tools for determining a fuzzy preference structure from a fuzzy large preference relation?

Properties of fuzzy relations

Let R be a fuzzy binary relation on A, T a t-norm and S a t-conorm.

- R is *reflexive*: $\forall a \in A : R(a, a) = 1$,
- R is *irreflexive*: $\forall a \in A : R(a, a) = 0$.
- R is *symmetric*: $\forall a, b \in A : R(a, b) = R(b, a)$,
- R is *S-complete*: $\forall a, b \in A : a \neq b \Rightarrow S(R(a, b), R(b, a)) = 1$,
- R is *strongly S-complete*: $\forall a, b \in A : S(R(a, b), R(b, a)) = 1$,
- R is *strongly complete*: $\forall a, b \in A : \max(R(a, b), R(b, a)) = 1$,
- R is *weakly complete*: $\forall a, b \in A : R(a, b) + R(b, a) \geq 1$,

Properties of fuzzy relations (cont'd)

Let R be a fuzzy binary relation on A, T a t-norm and S a t-conorm.

- R is *T-transitive*:

  $$\forall a, b, c \in A : T(R(a, b), R(b, c)) \leq R(a, c),$$

- R is *negatively S-transitive*:

  $$\forall a, b, c \in A : R(a, c) \leq S(R(a, b), R(b, c)),$$

- R is *T-S-semitransitive*:

  $$\forall a, b, c, d \in A : T(R(a, b), R(b, c)) \leq S(R(a, d), R(d, c)).$$

- R is *T-S-Ferrers*:

  $$\forall a, b, c, d \in A : T(R(a, b), R(c, d)) \leq S(R(a, d), R(c, b)).$$

Properties of fuzzy relations (cont'd)

Let R be a fuzzy binary relation on A, T a t-norm and S a t-conorm.

- R is *T-antisymmetric*:
  $\forall a, b \in A : a \neq b \Rightarrow T(R(a, b), R(b, a)) = 0$,
- R is *T-E-antisymmetric*:
  $\forall a, b \in A : T(R(a, b), R(b, a)) \leq E(b, a)$,

  with E some T-equivalence relation, i.e., a reflexive, symmetric and T-transitive relation,
- R is *T-asymmetric*: $\forall a, b \in A : T(R(a, b), R(b, a)) = 0$.

Large preference relation (the crisp case)

Let R be a reflexive binary relation on A.
Then (P, I, J) given by

$$(P, I, J) = (R \cap (R^t)^C, R \cap R^t, R^C \cap (R^t)^C)$$

is a preference structure such that

$$R = P \cup I.$$

Large preference relation — a first attempt

Let R be a reflexive fuzzy binary relation on A, (T, S, n) be a continuous De Morgan triple with strict negation n.
Is (P, I, J) given by

$$\begin{aligned} P(a,b) &= T(R(a,b), n(R(b,a))), \\ I(a,b) &= T(R(a,b), R(b,a)), \\ J(a,b) &= T(n(R(a,b)), n(R(b,a))), \end{aligned}$$

a preference structure such that

$$R(a,b) = S(P(a,b), I(a,b))?$$

(Alsina, 1985)

No, there is no such De Morgan triple.

Fuzzy preference structure — second approach

Definition

Consider a continuous De Morgan triple (T, S, n) with some strong negation n.
A fuzzy preference structure on A is a triple (P, I, J) of binary fuzzy relations such that

- P is irreflexive, I is reflexive and J is irreflexive,
- P is T-asymmetric, I, J are symmetric,
- $P \cap_T I = \emptyset$, $P \cap_T J = \emptyset$, $I \cap_T J = \emptyset$,
- $(P \cup_S I)^C = P^t \cup_S J$, $(P \cup_S P^t \cup_S I)^C = J$, or $(P \cup_S P^t \cup_S I \cup_S J) = A^2$.

Assignment principle

Assignment principle

For any pair (a, b) of alternatives the decision maker is allowed to assign at least one of the degrees $P(a, b)$, $P(b, a)$, $I(a, b)$ and $J(a, b)$ freely.

Additive fuzzy preference structure

Definition (De Baets, Fodor (2003))

Let φ be an automorphism on $[0,1]$ and consider the continuous De Morgan triple $(T_{\mathbf{L}_\varphi}, S_{\mathbf{L}_\varphi}, N_\varphi)$.
A φ-fuzzy preference structure (P, I, J) on A is a triple of binary fuzzy relations on A such that

- P is irreflexive, I is reflexive and J is irreflexive,
- P is $T_{\mathbf{L}_\varphi}$-asymmetric, I, J are symmetric,
- $P \cap_{T_{\mathbf{L}_\varphi}} I = \emptyset$, $P \cap_{T_{\mathbf{L}_\varphi}} J = \emptyset$, $I \cap_{T_{\mathbf{L}_\varphi}} J = \emptyset$,
- $(P \cup_{S_{\mathbf{L}_\varphi}} I)^C = P^t \cup_{S_{\mathbf{L}_\varphi}} J$.

Additive fuzzy preference structure

(De Baets, Fodor (2003))

A triple (P, I, J) is a φ-fuzzy preference structure if and only if I is reflexive and symmetric, and for all $a, b \in A$:

$$\varphi(P(a,b)) + \varphi(P(b,a)) + \varphi(I(a,b)) + \varphi(J(a,b)) = 1$$

An axiomatic approach (Fodor,Roubens (1994))

... for determining a fuzzy preference structure (P, I, J) from some large preference relation R.

Axioms

(IA) Independence of irrelevant alternatives:
For any two alternatives a, b the values of $P(a,b)$, $I(a,b)$, and $J(a,b)$ depend only on the values $R(a,b)$ and $R(b,a)$, i.e., there exist three functions $p, i, j\colon [0,1]^2 \to [0,1]$ such that

$$P(a,b) = p(R(a,b), R(b,a)),$$
$$I(a,b) = i(R(a,b), R(b,a)),$$
$$J(a,b) = j(R(a,b), R(b,a)).$$

An axiomatic approach (Fodor,Roubens (1994))

Axioms

(PA) Positive association principle:
The functions $p(x, n(y))$, $i(x,y)$, $j(n(x), n(y))$ are increasing in both arguments.

(S) Symmetry:
The functions i, j are symmetric functions.

Axiomatic approach (Fodor,Roubens (1994))

Consider functions p, i, j fulfilling (IA), (PA), (S) and let (T, S, n) be a continuous De Morgan triple with a strict negation n.

If

$$S(p(x,y), i(x,y)) = x \qquad (P \cup I = R),$$
$$S(p(x,y), j(x,y)) = n(y) \quad (P \cup J = (R^t)^C),$$

for all $x, y \in [0,1]$, then there exists some automorphism φ such that

$$T = (T_{\mathbf{L}})_\varphi \quad \text{and} \quad n = (N)_\varphi.$$

Therefore, $(T, S, n) = (T_{\mathbf{L}\varphi}, S_{\mathbf{L}\varphi}, N_\varphi)$ for some automorphism φ.
Upper and lower bounds for p, i, j are imposed.

Axiomatic approach (Fodor,Roubens (1994))

Let p, i, j fulfil (IA), (PA), (S) and consider $(T_{\mathbf{L}\varphi}, S_{\mathbf{L}\varphi}, N_\varphi)$ for some automorphism φ. Assume that, for all $x, y \in [0,1]$,

$$S_{\mathbf{L}\varphi}(p(x,y), i(x,y)) = x$$
$$S_{\mathbf{L}\varphi}(p(x,y), j(x,y)) = N_\varphi(y),$$

then

$$T_{\mathbf{L}\varphi}(x, N_\varphi(y)) \le p(x,y) \le \min(x, N_\varphi(y)),$$
$$T_{\mathbf{L}\varphi}(x,y) \le i(x,y) \le \min(x,y),$$
$$T_{\mathbf{L}\varphi}(N_\varphi(x), N_\varphi(y)) \le j(x,y) \le \min(N_\varphi(x), N_\varphi(y)).$$

Axiomatic approach (De Baets, Fodor (2003))

Let p, i, j fulfil (IA), (PA), (S) and consider $(T_{\mathbf{L}\varphi}, S_{\mathbf{L}\varphi}, N_\varphi)$ for some automorphism φ. Assume that, for all $x, y \in [0,1]$,

$$S_{\mathbf{L}\varphi}(p(x,y), i(x,y)) = x$$
$$S_{\mathbf{L}\varphi}(p(x,y), j(x,y)) = N_\varphi(y).$$

Then, for every reflexive binary fuzzy relation R on A, the triplet (P, I, J) defined by

$$P(a,b) = p(R(a,b), R(b,a))$$
$$I(a,b) = i(R(a,b), R(b,a))$$
$$J(a,b) = j(R(a,b), R(b,a))$$

is a φ-fuzzy preference structure such that $R = P \cup_{S_{\mathbf{L}\varphi}} I$ and $R^c = P^t \cup_{S_{\mathbf{L}\varphi}} J$.

Generator triple (De Baets, Fodor (2003))

Definition

A triple (p, i, j) of functions from $[0,1]^2$ into $[0,1]$ is called a generator triple compatible with a continuous t-conorm S and a strong negation n if for every reflexive binary fuzzy relation R it holds that the triple (P, I, J) defined by

$$P(a,b) = p(R(a,b), R(b,a))$$
$$I(a,b) = i(R(a,b), R(b,a))$$
$$J(a,b) = j(R(a,b), R(b,a))$$

is a fuzzy preference structure such that $P \cup_S I = R$ and $P^t \cup_S J = R^C$.

Generator triple (De Baets, Fodor (2003))

Theorem

A generator triple (p, i, j) is compatible with $S_{\mathbf{L}}$ and N if and only if, for all $x, y \in [0, 1]$

- $i(1, 1) = 1$,
- $i(x, y) = i(y, x)$,
- $p(x, y) + p(y, x) + i(x, y) + j(x, y) = 1$,
- $p(x, y) + i(x, y) = x$.

It then follows that $p(x, y) = x - i(x, y)$ and $j(x, y) = i(x, y) - (x + y - 1)$.

Generator triple (De Baets, Fodor (2003))

Theorem

A generator triple (p, i, j) compatible with $S_{\mathbf{L}}$ and N is monotone (fulfills conditions corresponding to (PA)) if and only if i is a commutative quasi-copula, i.e., a commutative, increasing function from $[0, 1]^2$ to $[0, 1]$ which is 1-Lipschitz and has neutral element 1.

Theorem

Consider a generator triple (p, i, j) such that i is a t-norm, then the following statements are equivalent:

- $j(1 - x, 1 - y)$ is a t-norm,
- $p(x, 1 - y)$ is symmetric,
- i is an ordinal sum of Frank t-norms.

Related topics and problems

- Transitivity concepts based on (quasi-)copulas, conjunctors
- Decomposition of transitivity properties of R, I, and P
- Reciprocal relations and related transitivity concepts
- Group preferences
  - aggregation of preference structures,
  - ranking procedures
- Different types of scales for valuations
- ...

C. Alsina, M. Frank, and B. Schweizer. *Associative Functions: Triangular Norms and Copulas*. World Scientific Publishing Company, Singapore, 2006.

D. Bouyssou, T. Marchant, M. Prilot, P. Perny, A. Tsoukiàs, and P. Vincke. *Evaluation and Decision Models — A Critical Perspective*, volume 32 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Boston, London, Dordrecht, 2000.

B. De Baets, J. Fodor. Twenty years of fuzzy preference structures (1978–1997). *Riv. Mat. Sci. Econom. Social.*, 20:45–66, 1997.

B. De Baets, J. Fodor. Additive fuzzy preference structure: the next generation. In: *Principles of Fuzzy Preference Modelling and Decision Making*, B. De Baets, J. Fodor (Eds.), Academia Press, p.15–27, 2003.

J. Fodor. A new look on fuzzy connectives. *Fuzzy Sets and Systems*, 57:141–148, 1993.

J. Fodor and M. Roubens. *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer Academic Publishers, Dordrecht, 1994.

R. L. Keeney and H. Raifa. *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, New York, Santa Barbara, London, Sydney, Toronto, 1976.

E. P. Klement and R. Mesiar, editors. *Logical, Algebraic, Analytic, and Probabilistic Aspects of Triangular Norms*. Elsevier, Amsterdam, 2005.

E.P. Klement, R. Mesiar, and E. Pap. *Triangular Norms*, volume 8 of *Trends in Logic. Studia Logica Library*. Kluwer Academic Publishers, Dordrecht, 2000.

E.P. Klement, R. Mesiar, and E. Pap. Triangular norms. Position paper I: basic analytical and algebraic properties. *Fuzzy Sets and Systems*, 143:5–26, 2004.

E.P. Klement, R. Mesiar, and E. Pap. Triangular norms. Position paper II: general constructions and parameterized families. *Fuzzy Sets and Systems*, 145:411–438, 2004.

E.P. Klement, R. Mesiar, and E. Pap. Triangular norms. Position paper III: continuous t-norms. *Fuzzy Sets and Systems*, 145:439–454, 2004.

J.-C. Pomerol and S. Barba-Romero. *Multicriterion Decision in Management: Principles and Practice*. Kluwer Academic Publishers, Boston, Dordrecht, London, 2000.

M. Roubens and P. Vincke. *Preference Modeling*, volume 250 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1985.

B. Roy. *Méthodologie multicritère d'aide à la décision*. Economica, Paris, 1985.

E. Trillas. Sobre funciones de negación en la teoría de conjuntos difusos. *Stochastica*, III:47–60, 1979.

B. Van De Walle, B. De Baets, E. Kerre. Characterizable fuzzy preference structures. *Annals of Operational Research*, 80:105–136, 1998.

P. Vincke. *Multicriteria Decision-Aid*. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1992.

Relation Algebraic Aspects of Semantics, Visualization and Implementation for Functional Logic Languages

Bernd Braßel

Institut für Informatik
Christian-Albrechts-Universität zu Kiel, Germany
bbr@informatik.uni-kiel.de

The main topic of my PhD-thesis in computer science is approaches to debugging functional logic languages. This concerns both the formal base and the implementation of such debugging tools, especially for the language Curry. In my search for a better formal framework for reasoning about functional logic programs I have come across relation algebra. Developing a relation algebraic semantics of Curry has already proven most interesting and fruitful, the results will be presented at RelMiCS10. As discussed in the corresponding paper, the resulting relation algebraic framework has very promising advantages over previous approaches. While former approaches are either too abstract or too technically detailed, the new framework might just have the right level of detail. Accordingly, one of my aims in the near future is to employ this framework to clarify several topics hotly discussed in the functional logic community and to provide interesting solutions to the corresponding problems.

More concretely connected to the topic of debugging functional logic programs is the problem of program visualization. The ideal debugging tools would give you only relevant information at a glance. Up to now, fruitful approaches to visualize functional logic programs and their execution are scarce. It has often been observed, however, that relation algebraic terms have a straightforward and intuitive visualization comparable to circuit layout. Therefore, the developed relation algebraic framework for Curry might enable such a visualization technique.

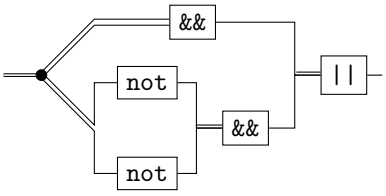
For example, the Boolean if-and-only-if could be defined in Curry as follows, where `&&` is Boolean conjunction, `||` is disjunction and `not` is negation.

```
(<=>) :: Bool -> Bool -> Bool
x <=> y = (x && y) || (not x && not y)
```

As a first step we employ a transformation on this program, introducing tupling `fork` and parallel and sequential composition `/` and `*`.

```
(<=>) :: (Bool,Bool) -> Bool
(<=>) = fork * ((&&) / ((not / not) * (&&))) * (||)
```

The resulting declaration can be visualized quite concisely.



In the middle term I would like to find out how such visualizations behave on a larger scale. Techniques for scaling, coloring and filtering might yield interesting tools for debugging.

In the long term an algebraic approach to programming languages is promising to enable a high level treatment of advanced compilation techniques like abstract interpretation, type and effect systems and partial evaluation. Some ideas to develop an operational semantics which conforms to the algebraic semantics presented at RelMiCS could even help to improve existing implementations of functional logic languages.

The chance to discuss the above ideas with international experts while at the same time learning more about the state of the art would be highly appreciated.

Program Inversion and Relation Algebra

Jan Christiansen

Department of Computer Science, CAU Kiel, Germany
jac@informatik.uni-kiel.de

1 Introduction

The programming language Curry provides a feature called function patterns [1]. This feature is defined only by a concrete implementation in the programming environment PAKCS [2]. We want to provide a deeper understanding of this feature by relating it with program inversion. There are many applications that can profit from this relationship. First of all we hope to define a denotational semantics for function patterns for the first time by employing this relationship. Furthermore there are approaches to automatic program inversion which can serve as a more efficient implementation for function patterns. This way the compiler could transform a function pattern into an application of a standard Curry operation. The current implementation uses a new primitive to implement function patterns which is not available in all Curry implementations. Program inversion of strict programs can easily be formalized in an relation algebraic semantics. Inversion in a lazy setting and in particular function patterns seem to break some of the algebraic laws of inversion. At the end we give an outlook on an implementation of program inversion for lazy programs by employing partial evaluation.

2 Program Inversion

The goal of program inversion is to find an inverse program $f^{-1} :: B \to A$ for a program $f :: A \to B$ such that $y = f(x) \Leftrightarrow f^{-1}(y) = x$. There are a couple of approaches to invert programs by hand. These approaches are often used to derive a program from a program that solves the inverse problem. In our context we will focus on approaches to automatic inversion of programs. A recent approach to an automatic inversion of strict functional programs can be found in [3] and [4]. This approach performs a local inversion of a program. The result of this local inversion is possibly non-deterministic. To dissolve this non-determinism a transformation is used that uses techniques of LR parser construction. As Curry is non-deterministic we do not have to dissolve the non-determinism. Nevertheless the method presented in [3] and [4] could be used to improve the performance of the inverse programs because it reduces the amount of non-determinism in a program. Note that this approach to automatic program inversion is only concerned with strict programs. We are not aware of any approach to inversion in a lazy language.

In a strict setting we can define the semantics of program inversion by the inversion of the corresponding relation algebraic semantics. In [5] we have presented a transformation of arbitrary Curry programs into point-free Curry programs which directly correspond to relation algebraic expressions. This correspondence yields an relation algebraic semantics.

Note that we use the term *operation* instead of function in the context of Curry because Curry provides non-determinism. Consider the data type of polymorphic lists `List a` and the operation `null` that takes a list and checks whether the list is empty or not. In this section we assume all programs to be strict and address only strict semantics.

```
data List a = Nil | Cons a (List a)

null :: List a -> Bool
null Nil        = True
null (Cons _ _) = False
```

This operation can be expressed in point-free style as follows.

```
null :: List a -> Bool
null = invNil * True
     ? invCons * unit * False
```

There is a very close correspondence between this point-free program and its relation algebraic semantics.

$$[\![\texttt{null}]\!] = [\![\texttt{Nil}]\!]^{\mathsf{T}} \circ [\![\texttt{True}]\!] \cup [\![\texttt{Cons}]\!]^{\mathsf{T}} \circ \mathsf{L} \circ [\![\texttt{False}]\!]$$

We just interpret the point-free primitives by relation algebraic operations. For example, the primitive `(?)` corresponds to the relation algebraic union and the primitive `(*)` to the multiplication. The interpretation of `unit` is the all-relation, i.e., it is just some kind of type cast here. While in [3] no formal justification for the used transformation rules for local inversion are given this relation algebraic model provides a semantic background. Therefore in a strict language we can easily define a denotational semantics for program inversion on the basis of a relation algebraic semantics. Furthermore the transformation steps that are performed in the automatic inversion directly correspond to the relation algebraic laws of inversion. That is, the following laws are used to invert the semantics of an operation.

$$(R \circ S)^{\mathsf{T}} = S^{\mathsf{T}} \circ R^{\mathsf{T}} \qquad (R \cup S)^{\mathsf{T}} = R^{\mathsf{T}} \cup S^{\mathsf{T}} \qquad R^{\mathsf{T}^{\mathsf{T}}} = R$$

For the inverse of the semantics of `null` we get the following equation.

$$[\![\texttt{null}]\!]^{\mathsf{T}} = [\![\texttt{True}]\!]^{\mathsf{T}} \circ [\![\texttt{Nil}]\!] \cup [\![\texttt{False}]\!]^{\mathsf{T}} \circ \mathsf{L} \circ [\![\texttt{Cons}]\!]$$

3 Function Patterns

The programming language Curry supports a feature called Function Patterns that is very closely connected with program inversion. As the name suggests a function pattern is the extension of pattern matching from constructors to functions. The standard example for function patterns is the operation `last` that takes a list and yields the last element of this list. We employ the operation `append` that takes two lists and appends the second at the end of the first.

```
last :: List a -> a
last (append xs (Cons x Nil)) = x
```

We can define a function that inverts a given function by using function patterns. The following function takes a higher order function `f` and a value `b` and yields the corresponding argument.

```
invert :: (a -> b) -> b -> a
invert f b = inv b
  where inv (f a) = a
```

Note that we have to introduce the function `inv` only because we cannot use variable `f` twice in the pattern matching of `invert`.

On the other hand we can express an arbitrary function pattern by function inversion. We assume *invert* to be an inversion primitive. We can define the operation `last` on basis of *invert*. The operation `snd` takes a pair and yields its second component.

```
last :: List a -> a
last l = snd (invert(snoc) l)
 where
  snoc (xs,x) = append xs (Cons x Nil)
```

In contrast to the inversion in a strict setting the inversion that is defined by means of function patterns does not fulfil the algebraic laws of inversion. For example consider the operation `head` which yields the first element of a list and the operation `bools` which takes a Boolean value and yields an infinite list containing this value.

```
head :: List Bool -> Bool
head (Cons x xs) = x

bools :: Bool -> List Bool
bools x = Cons x (bools x)
```

The expression `invert bools` always yields an error no matter what we apply it to. This is not a restriction of function patterns but seems to be very natural for any inversion of the operation `bools` in a lazy context. On the basis of `head` and `bools` we can define an operation `bId` which is the identity on Boolean values.

```
bId :: Bool -> Bool
bId x = head (bools x)
```

The definition of `bId` can be made pointfree by using the operation `(*)` which is similar to the function composition `(.)` but with interchanged arguments.

```
bId :: Bool -> Bool
bId = bools * head
```

As `bId` is the inversion on Boolean values the inversion of this operation should be the identity on Boolean values, too. In fact the expression `inverse bId` is the identity on Boolean values. But because `invert bools` yields undefined for any value we get the following inequation.

$$[\![\texttt{invert (bools * head)}]\!] \neq [\![\texttt{(invert head) * (invert bools)}]\!]$$

In our relation algebraic model we define the semantics of `(*)` as the relation algebraic composition $\circ$. Therefore, if we define $[\![\texttt{invert f}]\!] = inv([\![\texttt{f}]\!])$ for all operations `f` and an appropriate meta operation inv we get the following inequation.

$$inv([\![\texttt{bools}]\!] \circ [\![\texttt{head}]\!]) \neq inv([\![\texttt{head}]\!]) \circ inv([\![\texttt{bools}]\!])$$

Therefore function patterns seem to break the relation algebraic law $(R \circ S)^{\mathsf{T}} = S^{\mathsf{T}} \circ R^{\mathsf{T}}$. Note that this is not necessarily the case for all possible relation algebraic models. Note furthermore that this law is essential for the automatic inversion presented in [3]. That is, we cannot directly adapt automatic inversion of strict programs to lazy programs.

4 Partial Evaluation

We assume that we can use partial evaluation to at least approximate the semantics of function patterns. That is, we believe that the programs that break the inversion law are programs that can be simplified by partial evaluation. For these simplified programs we can easily compute their inverses. For example consider the operations `head`, `bools` and `bId` from above.

```
head :: List Bool -> Bool
head (Cons x xs) = x

bools :: Bool -> List Bool
bools x = Cons x (bools x)

bId :: Bool -> Bool
bId x = head (bools x)
```

The lazy semantics of these operations are the least fix-points of the following equations. In the lazy setting we use the relation U to model laziness.

$$head = \mathsf{U} \cup [\![\texttt{Cons}]\!]^{\mathsf{T}} \circ \pi$$

$$bools = \mathsf{U} \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!]$$
$$bId = \mathsf{U} \cup bools \circ head$$

The following equations present a partial evaluation of the operation `bId`. We prove that `bId` is semantically equivalent to an identity operation.

$$\begin{aligned}
bId &= \mathsf{U} \cup bools \circ head \\
&= \mathsf{U} \cup (\mathsf{U} \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!]) \circ (\mathsf{U} \cup [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi) \\
&\{(R \cup S) \circ T = R \circ T \cup S \circ T\} \\
&= \mathsf{U} \cup \mathsf{U} \circ \mathsf{U} \cup \mathsf{U} \circ [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!] \circ \mathsf{U} \\
&\cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!] \circ [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi \\
&\{R \neq \mathsf{O} \Rightarrow R \circ \mathsf{U} = \mathsf{U}\} \\
&= \mathsf{U} \cup \mathsf{U} \circ [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!] \circ [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi \\
&\{C \ constructor \ \Rightarrow \mathsf{U} \circ [\![\, C \,]\!]^{\mathsf{T}} = \mathsf{O}\} \\
&= \mathsf{U} \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ [\![\, \texttt{Cons} \,]\!] \circ [\![\, \texttt{Cons} \,]\!]^{\mathsf{T}} \circ \pi \\
&\{C \ constructor \ \Rightarrow [\![\, C \,]\!] \circ [\![\, C \,]\!]^{\mathsf{T}} = \mathsf{I}\} \\
&= \mathsf{U} \cup [\mathsf{I}, \mathsf{I}] \circ (\mathsf{I} \parallel bools) \circ \pi \\
&\{S \neq \mathsf{O} \Rightarrow (R \parallel S) \circ \pi = \pi \circ R\} \\
&= \mathsf{U} \cup [\mathsf{I}, \mathsf{I}] \circ \pi \\
&\{R, S \neq \mathsf{O} \Rightarrow ([R, S]) \circ \pi = R\} \\
&= \mathsf{U} \cup \mathsf{I}
\end{aligned}$$

We can easily compute the inverse of the result of the partial evaluation. We hope that we can automate the process of partial evaluation and use it to define a program transformation that computes the inverse of a lazy operation which serves as an efficient implementation of function patterns.

References

1. Antoy, S., Hanus, M.: Declarative programming with function patterns. In: Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05), Springer LNCS 3901 (2005) 6–22
2. Hanus, M., Antoy, S., Braßel, B., Engelke, M., Höppner, K., Koj, J., Niederau, P., Sadre, R., Steiner, F.: PAKCS: The Portland Aachen Kiel Curry System. Available at `http://www.informatik.uni-kiel.de/~pakcs/` (2007)
3. Glück, R., Kawabe, M.: A method for automatic program inversion based on LR(0) parsing. Fundamenta Informaticae **66**(4) (2005) 367–395
4. Kawabe, M., Glück, R.: The program inverter LRinv and its structure. In Hermenegildo, M.V., Cabeza, D., eds.: PADL. Volume 3350 of Lecture Notes in Computer Science., Springer (2005) 219–234
5. Braßel, B., Christiansen, J.: Denotation by transformation - towards obtaining a denotational semantics by transformation to point-free style. In: LOPSTR. (2007) To be published.

First-Order Theorem Prover Evaluation w.r.t. Relation- and Kleene Algebra

Han-Hing Dang and Peter Höfner

Institut für Informatik, Universität Augsburg, Germany
Han-Hing@gmx.de, hoefner@informatik.uni-augsburg.de

Abstract. Recently it has been shown that off-the-shelf first-order automated theorem provers can successfully verify statements of substantial complexity in relation and Kleene algebras. Until now most of our proof automation had been done using McCune's theorem prover Prover9, while others like SPASS and Vampire were not used extensively. In this paper we use more than 500 theorems to compare and evaluate 13 first-order theorem provers.

1 Introduction

To reach more automation within the areas of relation and Kleene algebras it has been shown that off-the-shelf automated theorem proving (ATP) systems can successfully verify statements of substantial complexity. Examples cover computational logics, relational reasoning, termination analysis, hybrid system analysis and the refinement calculus (see e.g. [6–8]). So far, mainly McCune's Prover9 tool was used for first-order automated reasoning. Motivated by the success and the variety of first-order ATP systems there arises the question whether Prover9 is the best choice for reasoning in such algebras.

Next to this first-order approach, there are others using interactive, higher-order theorem provers (e.g. [9]) or special purpose first-order proof systems [12]. Such approaches require either development effort or user interaction; a disadvantage relative to off-the-shelf first-order ATP systems.

There are general ATP system competitions (e.g. CASC [16]), but obviously these events cannot focus on special algebras. A comparison in the area of demonic refinement algebra with around 40 theorems and 11 theorem provers was done in [8]. It shows that Prover9 and Vampire seem to be the best for that purpose. But, up to now, nobody has done a comparison based on a huge amount of Kleene-like theorems. In total we fed different ATP systems with more than 500 theorems, which allows a reasonable evaluation of the theorem provers involved.

Our main result is that the ATP systems show dramatically different behaviour. On more difficult proof tasks, the best systems are clearly Prover9 and Waldmeister. The latter one can only be used for unit equational logic (i.e., purely equational reasoning). Moreover, we show that the success of theorem provers depends on the encoding of the problems.

In the remainder the expression "*algebra*" will be used as a short hand for "relation and Kleene algebra".

2 The Setting

We have evaluated 13 ATP systems[1] for finding proofs of *algebraic* theorems: Darwin 1.4.1, E 0.999, Equinox 1.3, Geo 2007f, iProver 0.2, leanCoP 2.0, Metis 2.0, Otter 3.3, Prover9 0607, SPASS 3.0, SNARK, Vampire 9.0 and Waldmeister 806[2]. We consistently use a black-box approach to theorem proving, i.e., we do not care about the search strategies or hints the ATP systems use. Initial tasks attempted with all the systems allowed us to select the most powerful systems for our evaluation. In particular, we compared Darwin, E, Otter, Prover9, SPASS, Vampire and Waldmeister in detail. All the other ATP systems failed already in proving some basic properties; it seems that they are not adequate for our task.

For the experiments we used computers with a hyper-threaded 3.0 GHz Intel Pentium 4 CPU and 1 GB memory, running a Linux 2.6 system. We set a CPU time limit of 600 s, which is known to be more than sufficient for the ATP systems to prove almost all the theorems they would be able to prove even with a significantly higher limit [15]. To have a uniform encoding of the involved structures we used the TPTP-format and Sutcliffe's SystemOnTPTP system.

An overview over the results is given in Section 3. Due to lack of space we cannot give the results nor the definitions of the involved structures in full detail. All these as well as the encodings are presented at a website [5].

It is well known that (human) reasoning in variants of Kleene algebras is often inequational. When producing the proof goals, we therefore split equations $s = t$ into inequations ($s \leq t$ and $t \leq s$), but also keep the equational variants, which probably are particularly hard for the ATP systems. Furthermore we also split each equivalence $s \Leftrightarrow t$ into two implications. Finally, implications containing an equation are split in a similar way where possible.

In sum, this yields 500 expressions as proof goals. We have also tested an equational encoding of relational algebra; therefore we used in fact 650 proof goals. We tried to prove them using only the axioms as hypotheses. We do this to have a uniform comparison base for all ATP systems, although it is well known that, in order to obtain ATP proofs of more difficult laws, further lemmas often need to be added to the axioms.

3 The Results

We split the class of all tested goals into three categories: variants of Kleene algebras (e.g. omega algebra or demonic refinement algebra), extensions of Kleene algebras (e.g. modal Kleene algebras) and relation algebras.

Variants of Kleene algebras. This category contains theorems from idempotent semirings, Kleene algebras, omega algebras [2] and demonic refinement algebras [17]. We put these algebras into one single category, since the axiom sets are similar and still small (compared to the next category). Overall we tried

[1] References for the used ATP systems can be found at [5].

[2] Waldmeister is equational based and accepts only universally quantified equations over a many-sorted signature; therefore it cannot directly used for Kleene algebra.

to prove 200 theorems. The proof goals vary from quite simple properties like isotony of addition to very complex formulas like Back's atomicity refinement law [1]. It cannot be expected that any of the ATP systems can prove such a theorem all by itself, since even the shortest proof by hand of this theorem in demonic refinement algebra is almost two pages long and uses lots of auxiliary lemmas. This result was confirmed by our experiments.

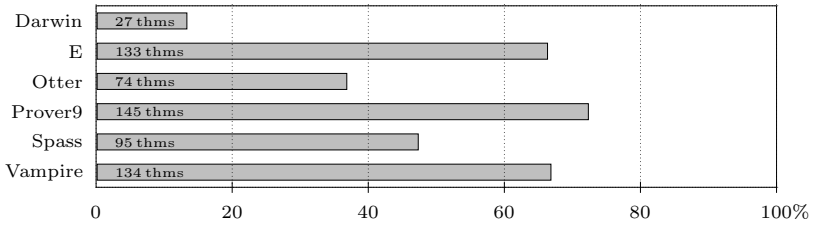


Fig. 1. Results for 200 theorems of variants of Kleene algebra

The results of our experiments are summarised in Fig. 1. The bars illustrate the percentage of theorems proved by the corresponding ATP system. The exact numbers are inside the bars. Although we focused on the most promising ATP systems, these show dramatically different behaviour. The best ones are clearly Prover9, E and Vampire. Overall we think that the success of ATP systems is quite impressive. Prover9 is able to show about 75 % of the goals starting from the axioms only. But, the results confirm that adding lemmas or the use of hypothesis learning techniques are indispensable for ATP systems. In particular, the use of such techniques yields proofs of all more involved theorems (cf. [6–8]).

Analysing the results we recognised that not only human reasoning in *algebra* is often inequational. Also the ATP systems perform much better when splitting equations. Often the systems cannot succeed with equations whereas they are able to prove both inequations. Moreover the different algorithms and strategies of ATP systems behave similar. That means, if Prover9 cannot find a proof, the chance that any other system will succeed is really small.

Extensions of Kleene algebras. This category covers extensions of idempotent semirings, Kleene algebras and omega algebras. In particular, structures with tests [10], with domain [3], and modal structures [14] are included.

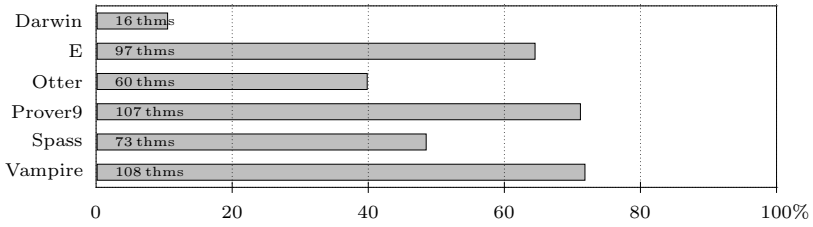


Fig. 2. Results for 150 theorems for extensions of Kleene algebras

The encodings of the extensions are based on the original axiomatisations. For example, modal Kleene algebra is based on the domain operator, which itself is based on a test algebra (a Boolean subalgebra). Such an axiomatisation yields

a dramatical increase in the number of axioms. Therefore, at first sight, these theorems seem to be more complex for ATP systems. But in fact, the results are basically the same as for the basic variants of Kleene algebra.

Relational algebra. Relational algebra can be encoded using universally quantified equations over a single signature. We used this fact to compare an inequational setting with an equational one.

To encode relation algebra we used an axiomatisation of Maddux [13]. In total, we gave 150 theorems to the ATP systems. Whenever an inequality $s \leq t$ arises, we also produced the equivalent form $s + t = t$ as proof goal and skip the order axiom $a \leq b \Leftrightarrow a + b = b$. To eliminate implications we skolemised the hypotheses. This yields an inequational and an equational encoding. The latter was also given to Waldmeister, which is known to be the most powerful system for equational deduction. The results for both encodings are listed in Fig. 3.

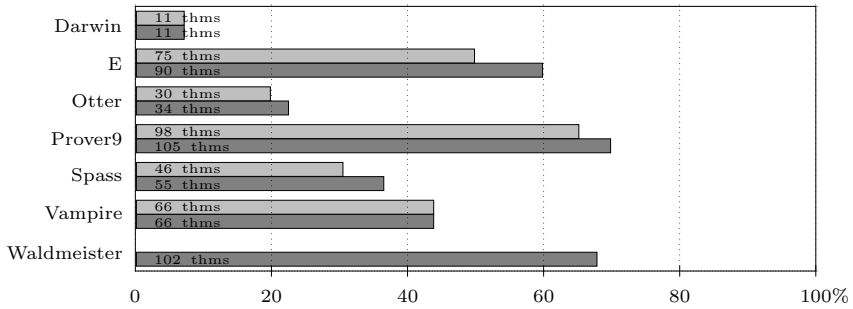


Fig. 3. Results for 150 theorems of relation algebra

The first bar presents the standard encoding, the second the equational one. As a result of our relational experiments we recognised that an equational encoding is significantly better than the standard one. Again the goals cover simple as well as complex theorems. Since we could not transform all theorems into an equational setting, Waldmeister performes even better than illustrated. In fact, it was able to proved 102 out of 135 theorems ($\approx 75\%$). Moreover, the results show that relational algebra is more complex for ATP systems than Kleene algebra. This might be due to the compact axiomatisation of Maddux, or the number of operations directly affects the power of the used systems.

4 Conclusion and Outlook

We tried to find out which theorem provers is the best for our tasks. In total we compared 13 ATP systems and fed them with more than 500 theorems. The best systems are Prover9 and, in the case of an equational encoding, Waldmeister.

But, no ATP system was able to prove all given theorems from the pure axiom set. This is not surprising since we add complex proof goals, like Back's atomicity refinement law. In previous work it has been shown that all the given theorems can be proved with Prover9 [6–8]. The strategies used to reach the goal are various combinations of an increased time limit, adding auxiliary lemmas as additional hypotheses and removal of axioms. For example, adding properties like

transitivity of the ordering, isotony of all operations, or the Dedekind formula would lead to better results. On the other hand, adding all known lemmas would lead to a state space explosion. Therefore axiom selection systems like SRASS become more and more important.

The inclusion of such tools into our experiments is part of our future work. Moreover we want to check whether SPASS performs better if one uses the fact that it can handle relational expressions directly. Furthermore, it will be interesting to analyse the produced results in more detail. For example, we suspect that there exist better *algebraic* encodings. In particular, we want to test a characterisation for the domain operator, which needs no tests [4], and a description for modal Kleene algebra based on modules [11]. Both characterisations yield less axioms and therefore we hope that ATP would yield better results.

Acknowledgements. We thank R. Glück, B. Möller and G. Struth for valuable discussions and comments and G. Sutcliffe for providing the software for TPTP. Last, we are grateful to him and to J. Flierl for their great technical support.

References

1. R.-J. Back. A method for refining atomicity in parallel algorithms. In E. Odijk, M. Rem and J.-C. Syr (eds.), *PARLE '91*, LNCS 366, pp. 199–216. Springer, 1989.
2. E. Cohen. Separation and reduction. In R. Backhouse and J. Oliveira (eds.), *MPC 2000*, LNCS 1837, pp. 45–59. Springer, 2000.
3. J. Desharnais, B. Möller and G. Struth. Kleene algebra with domain. ACM Trans. Comp. Logic, 7(4):798–833, 2006.
4. J. Desharnais and G. Struth. Domain semirings revisited. Technical Report CS-08-01, University of Sheffield, 2008.
5. P. Höfner and G. Struth. <`http://www.dcs.shef.ac.uk/~georg/ka`>.
6. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig (ed.), *CADE 2007*, LNAI 4603, pp. 279–294. Springer, 2007.
7. P. Höfner and G. Struth. On automating the calculus of relations. Technical Report CS-08-05, University of Sheffield, 2008.
8. P. Höfner, G. Struth and G. Sutcliffe. Automated verification of refinement laws. (submitted).
9. W. Kahl. Calculational relation-algebraic proofs in Isabelle/Isar. In R. Berghammer, B. Möller and G. Struth (eds.), *RelMiCS 7/AKA 2*, LNCS 3051, pp. 179–190. Springer, 2004.
10. D. Kozen. Kleene algebra with tests. ACM Trans. Program. Lang. Syst., 19(3):427–443, 1997.
11. H. Leiß. Kleene modules and linear languages. J. Log. Algebr. Program., 66(2):185–194, 2006.
12. W. MacCaull and E. Orlowska. Correspondence results for relational proof systems with application to the lambek calculus. Studia Logica, 71(3):389–414, 2002.
13. R. Maddux. Relation-algebraic semantics. Theo. Comp. Sc., 160(1&2):1–85, 1996.
14. B. Möller and G. Struth. Algebras of modal operators and partial correctness. Theo. Comp. Sc., 351(2):221–239, 2006.
15. G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. Artificial Intelligence, 131(1–2):39–54, 2001.
16. G. Sutcliffe and C. Suttner. The state of CASC. AI Comm., 19(1):35–48, 2006.
17. J. von Wright. From Kleene algebra to refinement algebra. In E. A. Boiten and B. Möller (eds.), *MPC 2002*, LNCS 2386, pp. 233–262. Springer, 2002.

The Theory of Allegories and Automated Proof Generation

Joel Glanfield*

Department of Computer Science,
Brock University,
St. Catharines, Ontario, Canada, L2S 3A1
joel.glanfield@brocku.ca

Abstract. The equational theory of allegories (ALL) is a decidable fragment of the theory of relations. Research to date has focused on providing a decision algorithm. We describe the development of an algorithm that will provide, in addition to a decision-certificate, a derivation. The algorithm is an extension of the decision procedure that extracts the individual rules (used throughout the proof) from normalization techniques used in the decision algorithm. Since the decision algorithm involves the association of terms with directed graphs and the normalization of these graphs, and since there are a relatively small number of general cases describing all possible graph-reductions, we focus on solving some of the problems that arise while attempting to describe such reductions.

1 Background

If we consider the context of the theory of relations, an allegory is a subtheory which drops the operations *complement* and *union*. We take the categorical approach as outlined in [1], which defines an allegory as follows:

Definition 1. *An ALLEGORY is a category with the following equations:*

1. $1; R = R = R; 1$
2. $R; (S; T) = (R; S); T$
3. $R \sqcap R = R$
4. $R \sqcap S = S \sqcap R$
5. $R \sqcap (S \sqcap T) = (R \sqcap S) \sqcap T$
6. $(R^{\smile})^{\smile} = R$
7. $(R; S)^{\smile} = S^{\smile}; R^{\smile}$
8. $(R \sqcap S)^{\smile} = R^{\smile} \sqcap S^{\smile}$
9. $R; (S \sqcap T) \sqsubseteq R; S \sqcap R; T$
10. $R; S \sqcap T \sqsubseteq (R \sqcap T; S^{\smile}); S$

The above definition is also related to categorical approaches introduced in [5] and mentioned in [2, 6, 7].

In his Ph.D thesis, Claudio Gutiérrez provided both a decision algorithm and a complete framework for calculating with this theory [4]. Decidability can be shown using normalization techniques on graphs that represent terms in the theory. In brief, two terms are equal if and only if there is an isomorphism between

* The author gratefully acknowledges support from the Natural Sciences and Engineering Research Council of Canada.

the normal forms of both graphs. Since the normalization process involves the factorization of graphs in a sequential process, and since a complete arithmetic framework has already been provided, it must be possible to extend this process to provide a derivation *in addition to* a decision certificate. If we could provide such an algorithm, then the implication is that there is potential for software to produce a derivation of any theorem in the equational theory. Hence we have also attempted to implement our findings in an existing proof system (i.e. RelAPS) [3].

2 Decidability of ALL

The decision algorithm presented in [4] is based on a graph-theoretical framework where terms in ALL are represented by graphs. One may consider that a relational variable is represented by a directed edge (labelled by the variable) connecting two vertices (representing the source and target of the relation). Every term in the theory of allegories has a corresponding graph; the details for constructing the graphs are found in [4].

If we are to determine whether two different terms are equal, the algorithm states that a series of reductions are to be performed on the graphs until a normal form for each graph is found (which are then checked for an isomorphism).

Figure 1 demonstrates the normalization process. We start with two terms and their respective graphs. These graphs are reduced (a finite number of times) until a normal form is produced. It is then determined whether the normal forms of the two graphs are isomorphic.

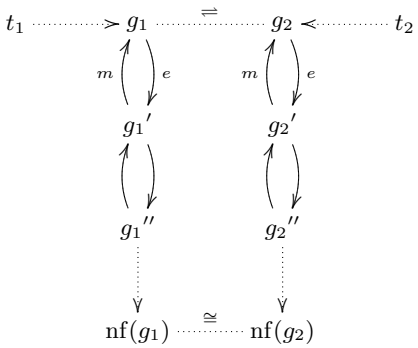


Fig. 1. Normalization process of graphs representing terms in ALL

3 Automatic Derivation

3.1 Categorical Context of Graphs for Terms in ALL

In [4], Gutiérrez presents 'a complete computational procedure for doing arithmetic in the theory of allegories' along with proof. Since the procedure is complete, it is implied that it is always possible to reason about reductions between graphs (representing terms in ALL) using equations from the theory of allegories. Since there are a relatively small number of cases (modulo symmetries) representing all possible reductions, we are in the process of implementing them in software. Since each case corresponds to an equation in the theory, this suggests that for any possible reduction, a corresponding equation can be shown.

What do we mean by 'reduction'? First, we must consider the set of all possible graphs representing terms in ALL. The graph operations are parallel composition ($g_1 \| g_2$), sequential composition ($g_1 | g_2$), and converse (g^{-1}), which relate to the theoretical operations intersection, composition and converse respectively. Some circumstances arise where branching in a graph occurs, hence a corresponding operation ($\mathrm{br}(g)$) was added to the set (called PLI_X). As a consequence, the *dom* operation is added to ALL as an operational extension that allows us to deal with such situations in order to allow a term to be constructed (for specific details we refer the reader to page 31 of [4]). The set PLI_X is closed under the operations mentioned. We then consider the categorical context of PLI_X and specifically the morphisms between the graphs of this category. The morphisms we are interested in are those between two graphs (g_1, g_2) where g_1 has had at least two vertices identified. The reduction steps are done by identifying two vertices v_1,v_2 in a graph, where vertex v_1 has at least the same set of edges as v_2. The first vertex is then removed from the graph. These reductions are then repeated indefinitely until there is no longer a possibility of identifying two vertices. After this process is complete, if we find an epimorphism from g_1 to g_2 and a monomorphism from g_2 to g_1, then we say g_1 has been reduced to g_2. However, it is possible that the identification of two vertices does not provide the desired morphisms (i.e. this occurs when g_2 is not a subgraph of g_1). Hence, we continue to factor the original graph until we find a subgraph (if one can be found); and thus we have a composition of arrows where the overall reduction gives an epimorphism and monomorphism.

The bottom line is that we associate theorems in ALL with different graph-reductions which occur while finding an mono-epi factorization. In the case where a reduction is a composition of arrows, we associate a theorem to the overall reduction of g_1 to g_2 where we ignore the vertices identified during the intermediate steps of the composition.

3.2 Relating Reductions to Theorems

Since we wish to associate each reduction with a theorem in ALL, we start by finding the minimal subgraph from the two vertices identified. The subgraph is then associated with the smallest term to which a theorem is applied. It has been

shown in [4] that such minimal subgraphs have a general structure and relatively few specific cases arise (modulo symmetries). Hence, we have theorems in ALL which describe every possible reduction. A problem arises when the subgraph does not directly correspond to a subterm of the given term. As an example, consider the following formula:

$$\mathrm{dom}(x); y \cap x = x \cap y$$

Working with the left-hand side, if we find the corresponding graph and perform the reduction, we obtain the following:

The rule corresponding to this reduction (recall that rules are determined by considering the minimal subgraph) is $\mathrm{dom}(x); x = x$ (equation (83) in [4]) and is drawn as follows:

It is obvious that the rule given cannot be applied directly to an appropriate subterm (i.e. $\mathrm{dom}(x); x$ cannot be applied to $(\mathrm{dom}(x)); y \cap x$ without some manipulation). However, we have a method that allows us to relate terms corresponding to graphs in the same equivalence class. In [4], a *standardization* process is given that allows us to find standard forms of equivalent terms. If we show a proof of the derivation of the standard form of a term, we can then relate two terms that have the same graph. For example, according to the definition of standardization given in [4], we can prove the following:

$$\mathrm{dom}(x); y \cap x = ... = (\mathrm{dom}(x)); (y \cap x)$$

Then, we provide a proof of the standardization of the form of the term we are looking for, namely $\mathrm{dom}(x); x \cap y$:

$$\mathrm{dom}(x); x \cap y = ... = (\mathrm{dom}(x)); (x \cap y)$$

We then combine the first proof with the second (while reversing the second proof) to obtain the following:

$$\mathrm{dom}(x); y \cap x = ... = (\mathrm{dom}(x)); (x \cap y) = ... = \mathrm{dom}(x); x \cap y$$

At this point we have the term in a form such that the rule can be applied directly, i.e. $\mathrm{dom}(x); x \cap y = x \cap y$ since $\mathrm{dom}(x); x = x$ can be applied.

The following is the entire proof of $\mathrm{dom}(x); y \cap x = x \cap y$ using the methodology described above:

$$\begin{aligned}\mathrm{dom}(x); y \cap x &=_{(43')} (\mathrm{dom}(x)); (y \cap x)\\ &=_{(43')} \mathrm{dom}(x); x \cap y\\ &=_{(83)} x \cap y,\end{aligned}$$

where both $43'$ and 83 refer to theorems proven in [4] ($43'$ is the symmetric version of 43).

4 Conclusion

Using a previously determined framework for performing arithmetic in ALL, one can decide which theorem in ALL is related to a given graph-reduction. Problems arise when the identification of two vertices does not lead to an embedding. When one is eventually found (by continuing to identify vertices), we look for the rule that can be applied to the overall reduction. Also, we have shown - from the use of standardization techniques - an example that demonstrates how terms may be manipulated in order to allow direct application of rules.

References

1. P. J. Freyd and A. Scedrov. *Categories, Allegories.* North-Holland, 1990.
2. H. Furusawa and W. Kahl. A study on symmetric quotients.
3. J. Glanfield and M. Winter. Relaps: A proof system for relational categories. In R. Schmidt and G. Struth, editors, *9th International Conference on Relational Methods and Kleene Algebra in Computer Science - PhD Programme*, pages 50–54, 2006.
4. C. Gutiérrez. *The Arithmetic and Geometry of Allegories - normal forms and complexity of a fragment of the theory of relations.* PhD thesis, Wesleyan University, 1999.
5. J. Olivier and D. Serrato. Catégories de dedekind. morphismes dans les catégories de schroder. *C.R. Acad. Sci. Paris*, 290:939–941, 1908.
6. G. Schmidt and T. Ströhlein. *Relations and graphs: discrete mathematics for computer scientists.* Springer-Verlag, London, UK, 1993.
7. M. Winter. A new algebraic approach to l-fuzzy relations convenient to study crispness. *Inf. Sci.*, 139(3-4):233–252, 2001.

Import Networks, Fuzzy Relations and Semirings

Roland Glück

Institut für Informatik, Universität Augsburg,
D-86135 Augsburg, Germany
glueck@informatik.uni-augsburg.de

Abstract. In this work we give a brief sketch how to handle circulations in import networks with fuzzy relations. We use techniques developed by Kawahara in [3] and modified in [1].

1 Introduction

Circulations in networks are similar to network flows, with two differences: there is no flow from a source to a sink, and at every node of the network a certain amount of flow is pumped into the network or has to leave it.

The aim is to give an algebraic approach to this problem, with the perspective of automated reasoning in this area.

2 Fuzzy Relations

2.1 Definitions and Basic Operations

Definition 2.1. A *fuzzy relation* α between sets X and Y, written $\alpha : X \leftrightarrow Y$, is a mapping from $X \times Y$ into the interval $[0, 1]$. A fuzzy Relation between a set X and itself is called a *fuzzy endorelation*. A fuzzy relation with a range contained in $\{0, 1\}$ is called Boolean.

The *empty relation* $\mathbf{0}_{XY}$, the *universal relation* ∇_{XY} and the *identity relation* id_X are given by $\mathbf{0}_{XY}(x, y) = 0$, $\nabla_{XY}(x, y) = 1$ for all $(x, y) \in X \times Y$ and $id_X(x, y) = \delta_{xy}$ for $(x, y) \in X \times X$. A fuzzy relation with a range contained in $\{0, 1\}$ is called *boolean*.

For $a, b \in [0, 1]$ we define $a \vee b = \max\{a, b\}$, $a \wedge b = \min\{a, b\}$, $a \ominus b = \max\{0, a - b\}$ and $a \oplus b = \min\{1, a + b\}$.

Consistent with these definitions we define the operators $\bigvee_{x \in X} x$ and $\bigwedge_{x \in X} x$ for arbitrary subsets $X \subseteq [0, 1]$ by $\bigvee_{x \in X} x := sup\{x \in X\}$ and $\bigwedge_{x \in X} x := inf\{x \in X\}$.

For fuzzy relations $\alpha, \beta : X \leftrightarrow Y$ the join $\alpha \sqcup \beta$, the meet $\alpha \sqcap \beta$, the truncating difference $\alpha \ominus \beta$ and the truncating sum $\alpha \oplus \beta : X \leftrightarrow Y$ are the pointwise extensions of the operators defined above: $(\alpha \sqcup \beta)(x, y,) = \alpha(x, y) \vee \beta(x, y)$, $(\alpha \sqcap \beta)(x, y,) = \alpha(x, y) \wedge \beta(x, y)$, $(\alpha \ominus \beta)(x, y,) = \alpha(x, y) \ominus \beta(x, y)$ and $(\alpha \oplus \beta)(x, y,) = \alpha(x, y) \oplus \beta(x, y)$.

Two fuzzy relations $\alpha, \beta : X \leftrightarrow Y$ are said to be *disjoint* if $\alpha \sqcap \beta = \mathbf{0}$. We abbreviate the union of two disjoint fuzzy relations α and β by $\alpha \dot{\sqcup} \beta$.

For two fuzzy relations $\alpha, \beta : X \leftrightarrow Y$ we write $\alpha \sqsubseteq \beta$, if $\alpha(x, y) \leq \beta(x, y)$ holds for all $(x, y) \in X \times Y$.

It is easy to see that meet, join and truncating sum are commutative and associative. Moreover, join distributes over meet and vice versa.

Another important operation on fuzzy relations is *scalar multiplication*. For real numbers $k \in [0, 1]$ and a fuzzy relation $\alpha : X \leftrightarrow Y$ the scalar product $k\alpha$, also written $k \cdot \alpha$, is defined by $(k\alpha)(x, y) = k \cdot \alpha(x, y)$ for all $(x, y) \in X \times Y$. It distributes over $\sqcup, \sqcap, \ominus$ and $\oplus$.

For a fuzzy relation $\alpha : X \leftrightarrow Y$ the *converse* $\alpha^\sharp : Y \leftrightarrow X$ is defined by $\alpha^\sharp(y, x) = \alpha(x, y)$. It commutes with scalar multiplications and distributes over $\sqcup, \sqcap, \ominus$ and $\oplus$.

We define the *composition* $\alpha\beta : X \leftrightarrow Z$ of two fuzzy relations $\alpha : X \leftrightarrow Y$ and $\beta : Y \leftrightarrow Z$ by $\alpha\beta(x, z) = \bigvee_{y \in Y} (\alpha(x, y) \wedge \beta(y, z))$. A fuzzy relation $\alpha : X \leftrightarrow Y$ is called *univalent* if $\alpha^\sharp \alpha \sqsubseteq id_Y$.

The composition of fuzzy relations distributes over join, i.e., $\alpha(\beta \sqcup \gamma) = \alpha\beta \sqcup \alpha\gamma$ and $(\alpha \sqcup \beta)\gamma = \alpha\gamma \sqcup \beta\gamma$. In general the composition does not distribute over meet, but for for univalent α the equality $\alpha(\beta \sqcap \gamma) = \alpha\beta \sqcap \alpha\gamma$ and for injective γ the equality $(\alpha \sqcap \beta)\gamma = \alpha\gamma \sqcap \beta\gamma$ hold.

Composition commutes with scalar multiplication, i.e., $k(\alpha\beta) = (k\alpha)\beta = \alpha(k\beta)$. Finally, composition is contravariant w.r.t. converse, i.e., $(\alpha\beta)^\sharp = \beta^\sharp \alpha^\sharp$.

The *n-th power* α^n of a fuzzy endorelation $\alpha : X \leftrightarrow X$ is defined inductively by $\alpha^0 = id_X$ and $\alpha^{n+1} = \alpha\alpha^n$ for $n \in \mathbb{N}_0$. The *reflexive and transitive closure* α^* of a fuzzy endorelation is defined by $\alpha^* = \bigvee_{n \in \mathbb{N}_0}$.

An important concept in the one of test relations:

Definition 2.2. *A Boolean subrelation of id_X is called a test relation on X.*

A subset X' of X corresponds to a test relation $\tau(X)$ on X, given by $\tau(X)(x, y) = 1$ if $x = y$ and $x \in X'$ and $\tau(X') = 0$ otherwise. Contrary every test relation on X describes a subset X' of X in an obvious manner. So we will use test relations to characterise subsets. For a single element $x \in X$ we abbreviate the corresponding test relation simply by x. Such a test relation is also called a *point relation*. These point relations can be algebraically characterised as minimal nonzero tests, i.e, tests p so that for all tests $q \neq 0$ the inequality $q \sqsubseteq p$ implies $q = p$.

For every test relation τ a unique test relation τ^c, the *complement* of τ, exists, characterised by $\tau\tau^c = \mathbf{0}_{XX} = \tau^c\tau$ and $\tau \sqcup \tau^c = id_X$. For a test relation $\tau(X')$ its complement is $\tau(\overline{X'})$.

To change the range (and hence the domain) of a fuzzy endorelation we introduce two operations, the embedding and the projection:

Definition 2.3. For a fuzzy endorelation $\alpha : X \leftrightarrow X$ on X and a superset $\lceil X \rceil$ of X, the *embedding* $\lceil \alpha \rceil$ of α into $\hat{X}$ is $\lceil \alpha \rceil : \hat{X} \leftrightarrow \hat{X}$, given by $\lceil \alpha \rceil(x, y) = \alpha(x, y)$ for all $(x, y) \in X \times X$ and $\lceil \alpha \rceil(x, y) = 0$ otherwise.

The embedding of fuzzy relations distributes over join, meet, truncating sum, truncating difference and composition, i.e., $\lceil \alpha \circ \beta \rceil = \lceil \alpha \rceil \circ \lceil \beta \rceil$ for $\circ \in \{\sqcup, \sqcap, \oplus, \ominus, \cdot\}$. In particular we have $\lceil \alpha \rceil = \lceil id \rceil \lceil \alpha \rceil = \lceil \alpha \rceil \lceil id \rceil$. Embedding also commutes with converse, i.e., $\lceil \alpha^\sharp \rceil = (\lceil \alpha \rceil)^\sharp$. Moreover, it is order-preserving: $\alpha \sqsubseteq \beta$ implies $\lceil \alpha \rceil \sqsubseteq \lceil \beta \rceil$.

The dual operation to embedding is projection:

Definition 2.4. For a fuzzy endorelation $\alpha : \hat{X} \leftrightarrow \hat{X}$ its *projection* $\lfloor \alpha \rfloor : X \leftrightarrow X$ to a subset $X \subseteq \hat{X}$ is given by $\lfloor \alpha \rfloor(x, y) = \alpha(x, y)$ for all $x, y \in X$.

The projection has algebraic properties dual to the ones of the embedding. It is also order preserving, it commutes with the converse and distributes over join, meet, truncating sum and truncating difference, but in general not over composition.

For a test τ both the embedding $\lceil \tau \rceil$ and the projection $\lfloor \tau \rfloor$ are tests, too.

2.2 Cardinality of Fuzzy Relations

Definition 2.5. The *cardinality* $|\alpha|$ of a fuzzy relation $\alpha : X \leftrightarrow Y$ is defined by $|\alpha| = \sum_{(x,y) \in X \times Y} \alpha(x, y)$.

Obvious properties of the cardinality are:

- $|\alpha| \geq 0$
- $|\alpha| = 0 \Leftrightarrow \alpha = 0_{XY}$
- $|\alpha| = |\alpha^\sharp|$
- Cardinality is an isotone function, i.e., $\alpha \sqsubseteq \beta$ implies $|\alpha| \leq |\beta|$.

Because we will limit ourselves to fuzzy relations over finite sets the cardinality will always be a nonnegative real number.

A fuzzy relation $\alpha : X \leftrightarrow Y$ is called *normalised* if $|\alpha| \leq 1$. This implies $|\beta| \leq 1$ for all fuzzy relations β with $\beta \sqsubseteq \alpha$.

A connection between join, meet and cardinality is the equality

$$|\alpha \sqcup \beta| = |\alpha| + |\beta| - |\alpha \sqcap \beta| \tag{1}$$

for arbitrary fuzzy relations $\alpha, \beta : X \leftrightarrow Y$. In particular, it states $|\alpha \sqcup \beta| = |\alpha| + |\beta|$ for disjoint fuzzy relations α and β.

For fuzzy relations α, β, γ with $\beta \sqsubseteq \alpha$ and $\gamma \sqsubseteq \alpha \ominus \beta$ the equality $|\gamma| + |\beta| = |\gamma \oplus \beta|$ holds. If $\beta \sqsubseteq \alpha$ then $|\alpha \ominus \beta| = |\alpha| \ominus |\beta|$.

If $\alpha \sqsubseteq \frac{1}{2} id$ and $\beta \sqsubseteq \frac{1}{2} id$ hold then $|\alpha \oplus \beta| = |\alpha| + |\beta|$.

3 Flows and Circulations

Definition 3.1. For a finite set X of nodes, an s-t-network N is a triple $N = (\alpha : X \leftrightarrow X, s, t)$ where s and t are two distinct elements of X and α satisfies $\alpha \sqcap \alpha^\sharp = \mathbf{0}_{XX}$. If the condition $\alpha \sqcap \alpha^\sharp$ is dropped N is called s *pseudo-s-t-network*. An *import network* is a triple $I = (\alpha : X \leftrightarrow X, \iota : X \leftrightarrow X, \omega : X \leftrightarrow X)$ with $\alpha \sqcap \alpha^\sharp = \mathbf{0}_{XX}$, $\alpha, \iota, \omega \sqsubseteq \frac{1}{2} id_X$ and $\iota \sqcap \omega = \mathbf{0}_{XX}$. An *import pseudonetwork* as defined analogously as above.

In both cases α is called the *capacity constraint*, s is the *source*, t is the *sink*, ι is the *import* and *omega* is the *output*.

Definition 3.2. A *flow* on an s-t-(pseudo)network $N = (\alpha : X \leftrightarrow X, s, t)$ is a fuzzy relation $\varphi : X \leftrightarrow X$ with $\varphi \sqsubseteq \alpha$ and $|\tau\varphi| = |\varphi\tau|$ for all test relations $\tau \sqsubseteq (s \sqcup t)^c$. A *circulation* φ on an import (pseudo)network $I = (\alpha : X \leftrightarrow X, \iota : X \leftrightarrow X, \omega : X \leftrightarrow X)$ is a fuzzy endorelation on X with $\varphi \sqsubseteq \alpha$ and $|(\varphi \oplus \iota)\tau| = |\tau(\varphi \oplus \omega)|$.

So a flow and a circulation have to respect the capacity constraints, the other two conditions are called *flow conservation*. The *value* $val(\varphi)$ of a flow φ in an s-t-network is given by $val(\varphi) = |s\varphi|$. It satisfies the equalities $val(\varphi) = |s\varphi| = |\varphi t|$.

A *cut* τ in an s-t-network is a test relation τ with $s \sqsubseteq \tau \sqsubseteq t^c$. The *capacity* $c(\tau)$ of a cut τ is defined via $c(\tau) = \tau\alpha\tau^c$.

A flow φ on an s-t-network N is called *maximal* if $val(\varphi) \geq val(\psi)$ holds for all flows ψ on N. A maximal flow always exists, while a circulation need not. The famous Max Flow-Min Cut-Theorem states that the value of a maximal flow equals the minimal capacity of a cut in an s-t-network. Such a cut τ is said to be saturated by the maximal flow φ with the consequence that $\tau\alpha\tau^c = \tau\varphi\tau^c$ holds.

4 Determining Circulations

After these preliminaries we apply our tools to import networks. Our goal is to obtain a criterion whether an import network admits a circulation or not. The strategy will be to construct from an import network an s-t-network and to reduce the problem of the existence of a circulation in the import network to the existence of a maximal flow of a certain value in the derived s-t-network.

Given an import network $I = (\alpha : X \leftrightarrow X, \iota : X \leftrightarrow X, \omega : X \leftrightarrow X)$ we construct an s-t-network $N = (\hat{\alpha} : \hat{X} \leftrightarrow \hat{X}, s, t)$ as follows: $\hat{X}$ is given by $\hat{X} = X \dot\cup \{s\} \dot\cup \{t\}$. $\hat{\alpha}$ is defined by $\hat{\alpha} = \alpha_1 \sqcup \alpha_2 \sqcup \alpha_3$ with $\alpha_1 = \bigsqcup_{x \in X} |\iota x| s \lceil \nabla \rceil \lceil x \rceil$, $\alpha_2 = \lceil \alpha \rceil$ and $\alpha_3 = \bigsqcup_{x \in X} |\omega x| \lceil x \rceil \lceil \nabla \rceil t$.

Due to lack of space we omit the proof that N is indeed an s-t-network and concentrate on the essential theorem of this paper:

Theorem 4.1. *There is a circulation on I iff the value of a maximal flow φ on N equals both $|\iota|$ and $|\omega|$.*

Proof. As a short preliminary we show that $c(s) = |\iota|$ and $c(t^c) = |\omega|$ hold. For this we calculate $s\hat{\alpha}s^c = s(\alpha_1 \sqcup \alpha_2 \sqcup \alpha_3)s^c = s\alpha_1 s^c \sqcup s\alpha_2 s^c \sqcup s\alpha_3 s^c$. By rather simple considerations we can see that only the term $s\alpha_1 s^c$ does not vanish, so we have $c(s) = |s\hat{\alpha}s^c| = |s\alpha_1 s^c| = |s \bigsqcup_{x \in X} |\iota x| s \lceil \nabla \rceil \lceil x \rceil|$. Because of $ss = s$, disjointness of the big join and distributivity we can replace this by $\Sigma_{x \in X} |\iota x|$, which is by similar reasons the same as $|\iota \bigsqcup_{x \in X} x|$, which yields $|\iota|$. Analogously we obtain $c(t^c) = |\omega|$. Furthermore we have by construction $\alpha_1 = s\alpha = s\alpha s^c$, $s\alpha_2 = s\alpha_3 = \mathbf{0}_{\hat{X}\hat{X}}$ and analogous properties for t.

Let now φ be a maximal flow on N with value $|\iota| = |\omega|$. Then we claim that $\lfloor\varphi\rfloor$ is a circulation on I. Obviously, due to the properties of embedding and projection, $\lfloor\varphi\rfloor$ respects the capacity constraint on I. Let now τ be an arbitrary test on X. Because of the flow conservation of φ we have $|\varphi\lceil\tau\rceil| = |\lceil\tau\rceil\varphi|$. From here on we can reason as follows:

$$
\begin{aligned}
& |\varphi\lceil\tau\rceil| = |\lceil\tau\rceil\varphi| \Rightarrow \\
= \quad & \{\![\ id_{\hat{X}} = s\dot{\sqcup}X\dot{\sqcup}t\ ]\!\} \\
& |(s\dot{\sqcup}X\dot{\sqcup}t)\varphi\lceil\tau\rceil| = |\lceil\tau\rceil\varphi(s\dot{\sqcup}X\dot{\sqcup}t)| \Rightarrow \\
= \quad & \{\![\ \text{disjointness, } \varphi \sqsubseteq \hat{\alpha}\text{, construction of } \hat{\alpha}\ ]\!\} \\
& |s\varphi\lceil\tau\rceil| + |X\varphi\lceil\tau\rceil| = |\lceil\tau\rceil\varphi t| + |\lceil\tau\rceil\varphi X| \Rightarrow \\
= \quad & \{\![\ \varphi \text{ saturates } s \text{ and } t^c\text{, remarks above}\ ]\!\} \\
& |s\hat{\alpha}\lceil\tau\rceil| + |X\varphi\lceil\tau\rceil| = |\lceil\tau\rceil\hat{\alpha}t| + |\lceil\tau\rceil\varphi X| \Rightarrow \\
= \quad & \{\![\ \text{remarks above, construction of } \hat{\alpha}\ ]\!\} \\
& |\iota\tau| + |\lfloor\varphi\rfloor\tau| = |\tau\omega| + |\tau\lfloor\varphi\rfloor| \Rightarrow \\
= \quad & \{\![\ \lfloor\varphi\rfloor \sqsubseteq \alpha,\ \alpha, \omega\iota \sqsubseteq 0.5id_X\text{, properties of } \oplus\ ]\!\} \\
& |(\lceil\varphi\rceil \oplus \iota)\tau| = |\tau(\lceil\varphi\rceil \oplus \omega)|
\end{aligned}
$$

Let now γ be a circulation on I. Then we construct a fuzzy endorelation φ on $\hat{X}$ by $\varphi = \lceil\gamma\rceil \sqcup \bigsqcup_{x\in X} |\iota x|s\lceil\nabla\rceil\lceil x\rceil \sqcup \bigsqcup_{x\in X} |\omega x|\lceil x\rceil\lceil\nabla\rceil t$. By construction, φ satisfies the capacity constraint of N; flow conservation can be shown similar as above. φ saturates the cuts s and t^c. Because of $c(s) = |\iota|$ and $c(t^c) = \omega$ φ is indeed a maximal flow on N with value $|\iota| = |\omega|$.

5 Conclusion

We gave a short sketch of how to apply Kawahara's methods to other related network problems. The proofs remained naturally a little bit sketchy, but it became visible how these methods can be extended for the use in other areas. As the mathematical structure of fuzzy relations is a Kleene algebra and automated reasoning in Kleene algebra is on the rise (cf. [2]) we soon expect first machine-generated proofs of theorems concerning networks.

References

1. R. Glück. Network flows, semirings and fuzzy relations. Technical Report 2008-01, Institut für Informatik, Universität Augsburg, 2008. `http://www.opus-bayern.de/uni-augsburg/volltexte/2008/726/`.
2. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig, editor, *Automated Deduction — CADE–21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 279–294. Springer, 2007.
3. Y. Kawahara. On the cardinality of relations. In R. A. Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2006.

Implication and Functional Dependency in Formal Contexts

Toshikazu Ishida, Kazumasa Honda and Yasuo Kawahara

Department of Informatics, Kyushu University, Fukuoka, 819-0395, Japan
t-ishida@i.kyushu-u.ac.jp

Abstract. Formal concept analysis is a mathematical field applied to data mining. Usually, a formal concept is defined as a pair of sets, called extents and intents, for a given formal context in binary relation. In this paper we review the idea that functional dependency is complete and sound for Armstrong's inference rules. Further, we prove that implication of formal concept is complete and sound for Armstrong's inference rules. Still, we give an example which shows the difference between implication and functional dependency.

1 Introduction

In data mining, we aim to discover hidden information, such as patterns and correlations, from massive data. In fact, the method is widely used in economic and scientific activities. Formal concept analysis is a mathematical field proposed by R. Wille in 1970's. Based on lattice theory, it enables us to search logic and knowledge structures, such as patterns and correlations, by means of concept lattices obtained from database. Therefore, formal concept analysis is applicable to data mining.

The database consists of binary relations between objects and attributes. A formal concept is defined as a set of pairs of objects and attributes which satisfy given conditions. The sets are called extents and intents.

A functional dependency is a correlation of attributes. The phenomenon occurs when an attribute in the database uniquely determines another attribute. On the other hand, B. Ganter and R. Wille defined another dependency (called implication [1]) which is different from functional dependency.

In this paper, we review the idea that functional dependency is complete and sound for Armstrong's inference rules. Further, we prove that implication of formal concepts is complete and sound for Armstrong's inference rules. Still, we give an example which shows the difference between implication and functional dependency.

2 Intensional Contexts

A formal context is a binary relation $\alpha : X \multimap Y$, which is equivalent to its intent function $\alpha^{@} : X \to \wp(Y)$. Thus $\alpha : X \multimap Y$ is equivalent to an X-indexed set $\mathcal{T} = \{T_x \mid x \in X\}$ of subsets of Y, that is, $T_x = x\alpha^{@}$.

	y_1	y_2	y_3	$\cdots$	
x_0	1	1	0	$\cdots$	T_0
x_1	1	1	1	$\cdots$	T_1
x_2	0	1	0	$\cdots$	T_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

$\leftrightarrow \quad \mathcal{T} = \{T_0, T_1, T_2, \dots\} \subseteq \wp(Y).$

An *intensional context* $\mathcal{T}$ on an attribute set Y is a subset of $\wp(Y)$, in other words, a family of subsets of Y.
The basic idea on Formal Concept Analysis (FCA) by B.Ganter and R.Wille [1], uses closure operations. These are defined for a formal context. For intensional contexts, the operation of a set of attributes B is modified as follows: $B^{\downarrow\uparrow} = \bigcap\{T \in \mathcal{T} \mid B \subseteq T\}$. In this sense the formal concept lattice $\mathcal{T}^*$ for an intensional context $\mathcal{T}$ is a subset $\mathcal{T}^* = \{\cap\mathcal{A} \mid \mathcal{A} \subseteq \mathcal{T}\}$ of $\wp(Y)$. It is trivial that $\mathcal{T}^*$ is a complete lattice. Also $\mathcal{T}^*$ is a closure system and $Y \in \mathcal{T}^*$.

3 Armstrong's Inference Rules

Armstrong's inference rules give a basic framework to treat the logical structure on dependencies on an attribute set. Let A and B be subsets of an attribute set Y. A formal expression $A \rhd B$, namely, an ordered pair of subsets A and B of Y, is called a *dependency* on Y.

Let $\mathcal{L}$ be a set of dependencies. We define that a dependency $A \rhd B$ is *provable* from $\mathcal{L}$ (written $\mathcal{L} \vdash A \rhd B$). $\mathcal{L} \vdash A \rhd B$ is defined following rules.

Definition 1. *Armstrong's Inference rules :*

$$\text{[A0]}\ \frac{}{A \rhd A} \quad \text{[A1]}\ \frac{A \rhd B}{A \cup C \rhd B} \quad \text{[A2]}\ \frac{A \rhd B \quad B \cup C \rhd D}{A \cup C \rhd D}$$

$$\text{[A0']}\ \frac{A \supseteq B}{A \rhd B} \quad \text{[A1']}\ \frac{A \rhd B \quad C \supseteq D}{A \cup C \rhd B \cup D} \quad \text{[A2']}\ \frac{A \rhd B \quad B \rhd C}{A \rhd C}$$

□

The system of inference rules [A1'], [A2'] and [A3'] is a mild variant of [A1], [A2] and [A3]. For a set L of dependencies we define a subset $A_{\mathcal{L}}$ of Y by $A_{\mathcal{L}} = \{y \in Y \mid \mathcal{L} \vdash A \rhd \{y\}\}$.

Lemma 1. *If B is a finite subset of Y, then $\mathcal{L} \vdash A \rhd B \ \leftrightarrow\ B \subseteq A_{\mathcal{L}}$.* □

4 Functional Dependency

In this section, we review a functional dependency [2]. In definition of a functional dependency, we use an equivalence relation [3–6].

Definition 2. *For each subset A of Y we define an equivalence relation $\theta[A]$: $\wp(Y) \rightharpoonup \wp(Y)$ by $(S,T) \in \theta[A] \leftrightarrow S \cap A = T \cap A$.* □

Proposition 1. *Let A and B be subsets of Y. Then*

(a) $\theta[Y] = \mathrm{id}_{\wp(Y)}$ *and* $\theta[\emptyset] = \nabla_{\wp(Y)\wp(Y)}$,
(b) $\theta[A \cup B] = \theta[A] \sqcap \theta[B]$,
(c) $\theta[A \cap B] = \theta[A]\theta[B]$. □

Definition 3. *Let $\mathcal{T}$ be an intensional context on Y.*

$$\mathcal{T} \models_F A \rhd B \leftrightarrow \forall S, T \in \mathcal{T}.\ (S \cap A = T \cap A \rightarrow S \cap B = T \cap B)$$
$$\leftrightarrow \forall S, T \in \mathcal{T}.\ (S,T) \in \theta[A] \rightarrow (S,T) \in \theta[B].$$

□

If $\mathcal{T} \models_F A \rhd B$ then $A \rhd B$ is called a functional dependency on $\mathcal{T}$.

Proposition 2. *Let $\mathcal{T} = \{S, T\}$ be an intensional context on Y. Then*

$$\mathcal{T} \models_F A \rhd B \ \leftrightarrow\ ((S,T) \in \theta[A] \rightarrow (S,T) \in \theta[B]).$$

□

Proposition 3. *Let $\mathcal{T}$ be an intensional context and $A \rhd B$ a functional dependency on Y.*
(A1') *If $A \supseteq B$ then $\mathcal{T} \models_F A \rhd B$.*
(A2') *If $\mathcal{T} \models_F A \rhd B$ and $C \supseteq D$ then $\mathcal{T} \models_F A \cup C \rhd B \cup D$.*
(A3') *If $\mathcal{T} \models_F A \rhd B$ and $\mathcal{T} \models_F B \rhd C$ then $\mathcal{T} \models_F A \rhd C$.* □

(A1'), (A2') and (A3') are called reflexive law, augmentation law and transitive law.

Proposition 4. *Let A be a proper subset of Y. There exists a set $\mathcal{T}_0$ such that*

(a) *$C \subseteq A$ if and only if $\mathcal{T}_0 \models_F \emptyset \rhd C$,*
(b) *$C \not\subseteq A$ if and only if $\mathcal{T}_0 \models_F C \rhd Y$.*

□

5 Implication

In this section, we review an implication [1].

Definition 4. *Let $\mathcal{T}$ be an intensional context on Y, and $\mathcal{L}$ a set of dependencies. We define $\mathcal{T} \models_I A \rhd B$ and $\mathcal{T} \models_I \mathcal{L}$ as follows:*

(a) $\mathcal{T} \models_I A \rhd B \ \leftrightarrow\ \forall T \in \mathcal{T}.\ (A \subseteq T \rightarrow B \subseteq T)$,
(b) $\mathcal{T} \models_I \mathcal{L} \ \leftrightarrow\ \forall A \rhd B \in \mathcal{L}.\ \mathcal{T} \models_I A \rhd B$.

□

If $\mathcal{T} \models_I A \rhd B$ then $A \rhd B$ is called implication. A dependency $A \rhd B$ is valid (as implication) for an intensional context $\mathcal{T}$ on Y if $\mathcal{T} \models_I A \rhd B$. And a set of implication $\mathcal{L}$ is valid for $\mathcal{T}$ on Y if $\mathcal{T} \models_I \mathcal{L}$.

Proposition 5. (a) $\mathcal{T} \models_I A \rhd A$, (A0)
(b) *If $\mathcal{T} \models_I A \rhd B$ then $\mathcal{T} \models_I A \cup C \rhd B$,* (A1)

(c) *If* $\mathcal{T} \models_I A \rhd B$ *and* $\mathcal{T} \models_I B \cup C \rhd D$ *then* $\mathcal{T} \models_I A \cup C \rhd D$. (A2)
□

Proposition 6. *Let* A *be a proper subset of* Y. *There exists a set* $\mathcal{T}_0$ *such that*

(a) $C \subseteq A$ *if and only if* $\mathcal{T}_0 \models_I \emptyset \rhd C$,
(b) $C \not\subseteq A$ *if and only if* $\mathcal{T}_0 \models_I C \rhd Y$. □

Proposition 7. *Any intensional context* $\mathcal{T}$ *and dependency* $A \rhd B$ *satisfy the following.*

(a) $\mathcal{T} \models_I A \rhd B$ *if and only if* $\mathcal{T}^* \models_I A \rhd B$.
(b) $\mathcal{T} \models_I A \rhd B$ *if and only if* $B \subseteq A^{\downarrow\uparrow}$. □

6 Soundness and Completeness

Now we will state the soundness and the completeness theorems of functional and logical dependencies for intensional contexts.

Theorem 1. *Let* $A \rhd B$ *be a dependency and* $\mathcal{L}$ *a set of dependencies on a finite set* Y. *Then the following equivalence holds:*

$$\mathcal{L} \vdash A \rhd B \leftrightarrow \forall \mathcal{T} \subseteq \wp(Y).\ (\mathcal{T} \models_\bullet \mathcal{L} \rightarrow \mathcal{T} \models_\bullet A \rhd B),$$

where $\bullet = F$ *or* I. □

Definition 5. *A set of dependencies* L *is closed, if*
$\forall A \rhd B. L \vdash A \rhd B \rightarrow A \rhd B \in L$. □

Theorem 2 (Maier). *A set* L *of dependencies is closed if and only if* L *satisfies following* B1∼3.

B1. $A \rhd A \in L$.
B2. *if* $A \rhd B \in L$ *then* $A \cup C \rhd B \in L$.
B3. *if* $A \rhd B, B \cup C \rhd D \in L$ *then* $A \cup C \rhd D \in L$. □

Therefore, a functional dependency and an implication are sound and complete for Armstrong's Inference rules.

7 Difference between Implication and Functional Dependency

In this section, we show the difference between an implication and a functional dependency, by using examples.

Let an intensional context $\mathcal{T}$ be $\{\{a, b\}, \{b, c\}, \{c\}\}$.

	a	b	c
x	×	×	
y		×	×
z			×

We consider the dependency $\{a\} \rhd \{b\}$. The set $\{a\}^{\downarrow\uparrow}$ is $\{a, b\}$. Therefore $\mathcal{T} \models_I \{a\} \rhd \{b\}$. But, for $\{b, c\}, \{c\} \in \mathcal{T}$, $\{b, c\} \cap \{a\} = \{c\} \cap \{a\}$, and $\{b, c\} \cap \{b\} \neq \{c\} \cap \{b\}$. Therefore $\{a\} \rhd \{b\}$ is not a functional dependency on $\mathcal{T}$. On the other hand, we consider the dependency $\{a\} \rhd \{c\}$. It is a functional dependency on $\mathcal{T}$, but it is not an implication on $\mathcal{T}$. Hence, an implication and a functional dependency are different. We will show other examples in the presentation.

We consider the condition on which the functional dependency and the implication of an intensional context are equivalent. And we find the following condition.

Proposition 8. *Let $\mathcal{T}$ be an intensional context. Define a set $\mathcal{T}'$ of subsets of Y by $\mathcal{T}' = \{(S^- \cup T) \cap (T^- \cup S) \mid S, T \in \mathcal{T}\}$. Then $\mathcal{T}' \models_I A \rhd B$ if and only if $\mathcal{T} \models_F A \rhd B$.*

8 Summary and Outlook

In this paper, we review the idea that functional dependency is sound and complete for Armstrong's inference rules. Further, we prove that implication of formal concept is complete and sound for Armstrong's inference rules. Still, we give an example which shows the difference between implication and functional dependency. In the future, we will consider the conditions on which implication and functional dependency are equivalent.

References

1. Ganter, B., Wille, R.: Formal Concept Analysis. Springer-Verlag (1999)
2. Codd, E.: A relational model of data for large shared data banks. Communications of the ACM **13** (1970) 377–387
3. Tarski, A.: On the calculus of relations. Journal of Symbolic Logic **6** (1941) 73–89
4. Olivier, J., Serrato, D.: Catégories de dedekind. morphismes dans les catégories de schröder. C. R. Acad. Sci. Paris **260** (1980) 939–941
5. Kawahara, Y.: Theory of relations. Lecture note (Japanese)
6. Okuma, H., Kawahara, Y.: Relational aspects of relational database dependencies. Bulletin of Informatics and Cybernetics **32** (2000) 91–104

Towards an Algebraic Composition of Semantic Web Services

Florian Lautenbacher and Peter Höfner

Institute of Computer Science, University of Augsburg, Germany
[lautenbacher|hoefner]@informatik.uni-augsburg.de

Abstract. To realise a software that needs to interact with other computers, service-oriented architecture and especially Web Services are gaining more and more attention. Existing components need to be composed in order to reach a predefined goal. For Web Services this can be realised using data defined in WSDL or using semantic Web Services such as SAWSDL. We apply relation algebra to describe Web Services as well as the composition of existing semantic Web Services.

1 Introduction

Most companies nowadays have business processes where frequent interaction with other companies happens on a regular basis. Hence, the involved systems must be able to interact somehow in order to realise predefined goals and finally to produce goods or services. This is only possible if every system can read the exchanged messages which means that the output of one involved system must be compatible with the input of another. The service-oriented architecture (SOA) and -paradigm can be applied to couple the systems in services which then interchange messages and can be discovered using a shared directory.

Web Services are one way to realise a SOA. The Web Services need to be described in a language such as the Web Service Description Language (WSDL) and message exchange happens using the SOAP standard. But these standards have shortcomings: since the exchanged messages are only described using human-comprehensible terms, they can be understood by different developers in various ways. Hence, standards of the semantic web can be applied to provide a unique meaning for those terms in all systems using ontologies.

The W3C has defined an extension to WSDL which is called SAWSDL, Semantic Annotations for WSDL and XML Schema, to support the use of concepts in an ontology for the definition of Web Services. Additionally, several attempts of semantic Web Services have been developed during the last years: OWL-S, SWSF or WSMO (e.g. [3]) to name just a few. The ontologies defined in these standards can then be combined using SAWSDL (see e.g. [7]).

An existing Web Service description can be used to achieve an automatic composition. To achieve a goal, several Web Services need to be composed and must interact with each other. In this paper we describe an algebra for the description of (semantic) Web Services and show how this algebra can assist the user in the composition of Web Services.

2 (Semantic) Web Service Standards

Each Web Service needs to have a specified interface in order to interact with others. WSDL [2] is a language designed to describe such an interface. A WSDL document consists of four parts which describe different levels of abstraction: `types, interfaces, bindings` and `services`. `Types` contain definitions for the datatypes used in messages. `Interfaces` describe which operations are supported by a service. `Bindings` and `services` specify the protocols and endpoints used to access the service. Input, output and fault messages specify operations of `interfaces`. They rely on data types described in `types`. Basic data types are defined by an XML Schema. They can be nested to more complex types.

SAWSDL [5], recommended by the W3C since 2007, is a set of extensions for WSDL to provide a standard description format. It allows a semantic annotation of a Web Service description using three different kinds of attributes: with `modelReference` one can create an association between a part in the WSDL document and a concept in an ontology. Type definitions, element declarations, attribute declarations as well as interfaces can be annotated. Using the attribute `liftingSchemaMapping` one can describe the mapping between the WSDL file and semantic data whereas `loweringSchemaMapping` specifies the mapping between a semantic concept and the constructs in the WSDL file.

3 A Formalisation of Web Services

In [4], we defined Web Service composition using relation algebra (e.g [9]). In this section we will recapitulate the basic definitions.

In the interfaces of WSDL several Web Methods are defined. These receive input messages and reply with output messages which both can be of a simple type such as string, integer, etc. or of a complex type. We will assume that the types of the input and the output data are known before and therefore there is a *knowledge set* $\mathcal{K}$, a set which includes the inputs and outputs as subsets. The concrete binding information or fault messages within WSDL are currently neglected for the sake of simplicity.

Definition 3.1. A *Web Method* is a tuple (I, O), where $I \subseteq O \subseteq \mathcal{K}$.

The condition $I \subseteq O$ guarantees that we do not lose any information, i.e., any information which is known before the execution of a Web Method is also known afterwards. $(\emptyset, O)$ represents a Web Method where no input is needed, i.e., it can be executed at any time.

In the following we will use a grammar-style notation to ease the reading and to focus on the input and the "real" output. In particular, we write $\{I \rightarrow O-I\}$ instead of (I, O). When possible, we will even skip the set brackets.

A Web Method is the simplest form of a Web Service, but it is atomic. Following WSDL, a Web Service may contain more than one Web Method, hence we define the concept of a *simple Web Service*.

Definition 3.2. A *simple Web Service* is a collection of Web Methods.

Operations like choice or sequential composition can also be applied to Web Services. Choice is the set-theoretic union and composition is the multiplication. As described in [4] this definition yields a strange behaviour. Therefore, we define an (extended) Web Service and Web Services composition as follows:

Definition 3.3. The *(extended) Web Service* of a simple Web Service W is the relation $\{(I \cup E, O \cup E) : (I, O) \in W, E \subseteq \mathcal{K} \backslash O\}$ and denoted by $\ll W \gg$. *Web Service composition* of V and W is then defined by $V \circ W =_{df} \ll V \gg ; \ll W \gg$, where ; denotes standard sequential composition of relations.

In this definition E is the context and the extension of the simple Web Service W, which just takes any information that is not needed as input for execution and adds this information unchanged to the output.

It is straightforward to show that extended Web Services are closed under choice and Web Service composition. Therefore they form an idempotent semiring[1]. Finite iteration of Web Services can be determined by the reflexive and transitive closure, denoted by *. It is easy to show that $\ll V^* \gg = \ll V \gg^*$. Henceforce, Web Services form also a Kleene algebra.

4 Web Service Restriction

In order to express the needed input data to perform a certain action or to describe goals we introduce the concept of tests. This allows an automatic composition of Web Services as described in [4].

One defines a *test* in an idempotent semiring [6] to be an element $p \leq 1$ that has a complement q relative to 1, i.e., $p + q = 1$ and $p \cdot q = 0 = q \cdot p$. In relation algebra every subidentity can be seen as a test.

Tests can be used to model assertions for Web Services. But they are also the basis for defining modal operators [1] which are used for modelling termination and to determine whether some goals can be reached by composing Web Services or not. Additionally, they can be used to compute an initial state that allows a composition of Web Services to be executed.

Informally, in the context of Web Services the forward diamond $|a\rangle p$ characterises the set of possible information with at least one successor in p when executing the Web Service a, i.e., the preimage of the set p under a. $|a]p$ characterises the situation where there is no execution of a, that starts in p and terminates in $\neg q$, the complement of q. Whenever an execution of a terminates in $\neg q$, the execution has to start in $\neg p$ and therefore $|a]p$ models the possible infomation from which execution of a is guaranteed to terminate in an element of p or the execution is not possible. The combination of both modal operators ($|a\rangle p$ and $|a]p$) guarantees that at least one result of the Web Service a exists and all resulting information is in p. This can be expressed by $|a\rangle p \cdot |a]p$.

[1] A semiring is called idempotent if $a + a = a$ for all elements.

5 Algebraic Composition of Semantic Web Services

Since all inputs and outputs of a Web Service can be annotated with concepts from an ontology, we have a closer look at these annotations now. Concepts in an ontology (e.g. defined in OWL [8]) can be connected with other concepts through user-defined *ObjectProperties* or *DataProperties*. With `SubClassOf` one can define inheritance and using `EquivalentClasses` the equivalence of two classes is specified. To formalise these concepts we discuss two possibilities:

First alternative

After each Web Service we include an additional one that extends the output parameters of the former with the concepts that can be "reasoned" through the ontology. Analogous to Definition 3.3 we define an ontology-based Web Service composition as follows:

Definition 5.1. *Ontological Web Service composition* of V and W is defined as $V \hat{\circ} W =_{df} V \circ Ont^* \circ W$, whereas Ont is a set of tuples (s,t) describing relations of concepts in the ontology with $s,t \in \mathcal{K}$ and hence a Web Service itself.

These relations have been stored e.g. in an OWL-file and need to be translated into tuples. After each call of Ont^* the available data has been extended with the `EquivalentClasses` or `SubClassOf` of the output data.

The Kleene star supports also transitive relations in the ontology. Now the second Web Service can be invoked and will find its input data, if it has been computed by V or by Ont^*.

Let us have a look at a small example: Assume Web Services B and Z. The former extracts the `birthdate` of a person from a database, i.e., $B =$ (`firstName lastName` $\rightarrow$ `birthdate`). Z needs some `date` to calculate the corresponding zodiac sign, i.e., $Z =$ (`date` $\rightarrow$ `zodiacSign`). In Ont the relations (`birthdate`, `date`) and (`firstName`, `name`) have been stored. $B \hat{\circ} Z$ yields that B and Z can be composed, since `birthdate` and `date` are related to each other in the ontology. The composed Web Service is then (`firstName lastName` $\rightarrow$ `date zodiacSign`).

Second alternative

For each Web Method we define additional ones by replacing the input data with equivalent classes and sub-classes.

Definition 5.2. Let S,T be sets with $S,T \subseteq \mathcal{K}$. A *refinement relation* is defined as $S \sqsubseteq T \Leftrightarrow_{df} \forall s \in S\ \exists t \in T : (s,t) \in Ont^*$. The *refined Web Service* of W is given by $W_{\sqsubseteq} =_{df} \{(\hat{I} \rightarrow O) \mid (I \rightarrow O) \in W,\ \hat{I} \sqsubseteq I,\ \hat{I} \subseteq \mathcal{K}\}$.

Applying the same example as before, we can now change Z and determine the Web Service $Z_{\sqsubseteq}$. In particular, $Z_{\sqsubseteq}$ includes (`birthdate` $\rightarrow$ `zodiacSign`). B can be composed with $Z_{\sqsubseteq}$, since the input of B and output of $Z_{\sqsubseteq}$ is identical.

Discussion

The first alternative uses a straight-forward relationship between the ontology

and our characterisation of Web Services. That is why the construction is quite natural. Nevertheless, it creates a lot of tuples that are not needed for the composition. Moreover, the output produced through the ontology might lead to confusion since the same information can now occur in different data of the output. For example `birthdate` will occur in `birthdate` and `date`.

The second alternative does not create unnecessary data; but the Web Services must be changed. At first sight this seems more complicated. Nevertheless, this modification has to be done only once. Moreover, if a Web Service accepts a `date` as input, it is obvious that it also accepts `birthdate` as input. Therefore, this approach is also natural.

We prefer the second approach since it reflects the intuition what data a Web Service may accept. Moreover, this approach can be extended with preconditions and effects lateron as needed in the area of semantic Web Services.

6 Conclusion and Outlook

In this paper we presented a method to describe Web Services based on a relation algebra. Additionally, we outlined two ideas how to describe semantic Web Services and ontologies. As this is current research, there are several things that still need to be done: An ontology does not only support `EquivalentClasses` or `SubClassOf`, but also other relationships between classes that we want to take care of. The tuples of the ontology might be extended to tripels to store these information. Also `lowering-` or `liftingSchemaMapping` are currently ignored. We cover only some parts of WSDL in our approach, but most of the others are not necessary for the composition anyway, but only for the enactment lateron.

References

1. J. Desharnais, B. Möller, and G. Struth. Termination in modal Kleene algebra. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *IFIP TCS2004*, pages 647–660. Kluwer, 2004.
2. W. Dostal, M. Jeckle, I. Melzer, and B. Zengler. *Service-orientierte Architekturen mit Web Services (in German)*. Spektrum Akademischer Verlag, 2005.
3. D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, and D. Roman. *Enabling Semantic Web Services - The Web Service Modeling Ontology*. Springer, 2007.
4. P. Höfner and F. Lautenbacher. Algebraic Structure of Web Services. In S. Escobar and M. Marchiori, editors, *WWV 07*, ENTCS (to appear), 2008.
5. J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, Nov./Dec.:60–67, 2007.
6. D. Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, 1997.
7. D. Martin, M. Paolucci, and M. Wagner. Toward Semantic Annotations of Web Services. In *OWL-S: Experiences and Directions, June, 6*, 2007.
8. B. Motik, P. F. Patel-Schneider, and I. Horrocks. OWL 1.1 Web Ontology Language - Structural Specification. Member subm., W3C, 2006.
9. G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Springer, 1993.

Multirelational Model of Lazy Kleene Algebra

Norihiro Tsumagari[1], Koki Nishizawa[2], and Hitoshi Furusawa[3]

[1] Graduate School of Science and Engineering, Kagoshima University
u01tsuma@eniac.sci.kagoshima-u.ac.jp
[2] Graduate School of Information Sciences, Tohoku University
nishizawa@kb.ecei.tohoku.ac.jp
[3] Faculty of Science, Kagoshima University
furusawa@sci.kagoshima-u.ac.jp

Abstract. This paper shows that the set of up-closed multirelations over a set forms a lazy Kleene algebra. Up-closed Multirelations are known as providing models of game logic. In game logic iterations are very important structures. An iteration is interpreted as the reflexive transitive closure of an up-closed multirelation. But it does not seem that a study of the (reflexive) transitive closure is enough. We investigate the reflexive transitive closure of an up-closed multirelation and show that the closure operator plays a rôle of the star of a lazy Kleene algebra consisting of the set of up-closed multirelations.

1 Introduction

A notion of lazy Kleene algebra is introduced by Möller [7] as a variation of Kleene algebra (KA) introduced by [4]; lazy Kleene algebra is KA's relaxation for the treatment of systems such as the calculus of finite and infinite streams.

Up-closed multirelations are studied as a semantic domain of programs. They serve predicate transformer semantics with both angelic and demonic nondeterminism in the same framework [3, 10, 11] . In [1] it has been shown that the set of finitary and total up-closed multirelations on a set forms a probabilistic Kleene algebra [5, 6]. Also it is known that up-closed multirelations provide models of game logic introduced by Parikh [9]. An overview of game logic has been given by Pauly and Parikh [8] . Goranko [2] and Venema [12] have studied the operations of game logic from an algebraic point of view. They have given complete axiomatizations of iteration-free game logic. Since an iteration is interpreted as the reflexive transitive closure it is necessary to establish a complete axiomatisation of game logic with iteration. We study the notion and we show that the set of up-closed multirelations forms a lazy Kleene algebra.

2 Lazy Kleene Algebra

We recall the definition of lazy Kleene algebras introduced in [7].

Definition 1. A *IL-semiring* is a tuple $(K, +, \cdot, 0, 1)$ with the following properties:

- $(K, +, 0)$ is a commutative monoid.
- $(K, \cdot, 1)$ is a monoid.
- The $\cdot$ operation satisfies the left-zero law and distributes over $+$ in its left argument :

$$0a = 0 \quad \text{and} \quad (a + b)c = ac + bc.$$

- In addition, $(K, +, \cdot, 0, 1)$ satisfies the following conditions for $a, b, c \in K$:

$$a + a = a \quad \text{and} \quad b \leq c \Rightarrow ab \leq ac \tag{1}$$

where $\cdot$ is omitted and the order $\leq$ is defined by $a \leq b$ iff $a + b = b$. □

Forgetting (1) from IL-semirings, we obtain *left semirings* [7].

Definition 2. A *lazy Kleene algebra* is a tuple $(K, +, \cdot, ^*, 0, 1)$ such that $(K, +, \cdot, 0, 1)$ is an IL-semiring and the *star* * satisfies the following conditions for $a, b \in K$:

$$1 + aa^* \leq a^* \tag{2}$$

$$ab \leq b \implies a^*b \leq b \tag{3}$$

where $\cdot$ is omitted and the order $\leq$ is defined by $a \leq b$ iff $a + b = b$. □

Clearly, Kozen's Kleene algebras are lazy Kleene algebras. In addition to (2), (3) probablistic Kleene algebras [5, 6] require the following two conditions:

$$a0 = 0 \tag{4}$$

$$a(b + 1) \leq a \Rightarrow ab^* \leq a \ . \tag{5}$$

3 Up-Closed Multirelations

In this section we recall definitions and basic properties of multirelations and their operations. More precise information on these can be obtained from [3, 10, 11].

A *multirelation* R over a set A is a subset of the Cartesian product $A \times \wp(A)$ of A and the power set $\wp(A)$ of A. A multirelation is called *up-closed* if $(x, X) \in R$ and $X \subseteq Y$ imply $(x, Y) \in R$ for each $x \in A$, $X, Y \subseteq A$. The null multirelation $\emptyset$ and the universal multirelation $A \times \wp(A)$ are up-closed, and will be denoted by 0 and ∇, respectively. The set of up-closed multirelations over A will be denoted by $\mathsf{UMRel}(A)$.

For a family $\{R_i \mid i \in I\}$ of up-closed multirelations, the union $\bigcup_{i \in I} R_i$ and the intersection $\bigcap_{i \in I} R_i$ are up-closed. So $\mathsf{UMRel}(A)$ is closed under arbitrary union $\bigcup$ and intersection $\bigcap$. Also the following holds.

Lemma 1. *A tuple* $(\mathsf{UMRel}(A), \cup, \cap, 0, \nabla)$ *is a complete distributive lattice.* □

$R + S$ denotes $R \cup S$ for a pair of up-closed multirelations R and S.

For a pair of multirelations $R, S \subseteq A \times \wp(A)$ the composition $R; S$ is defined by

$$(x, X) \in R; S \ \text{ iff } \ \exists Y \subseteq A.((x, Y) \in R \text{ and } \forall y \in Y.(y, X) \in S) \ .$$

The set $\mathsf{UMRel}(A)$ is closed under the composition ;. Also the following holds.

Lemma 2. *A tuple* $(\mathsf{UMRel}(A), ;)$ *is a semigroup.* □

The *identity* $1 \in \mathsf{UMRel}(A)$ is defined by

$$(x, X) \in 1 \text{ iff } x \in X \ .$$

Proposition 1. $(\mathsf{UMRel}(A), +, ;, 0, 1)$ *is an IL-semiring.* □

4 Reflexive Transitive Closure

We give a construction of the reflexive transitive closure of an up-closed multirelation. For $R \in \mathsf{UMRel}(A)$, a mapping $\varphi_R \colon \mathsf{UMRel}(A) \to \mathsf{UMRel}(A)$ is defined by

$$\varphi_R(\xi) = R; \xi + 1 \ .$$

Then, the mapping φ_R preserves the inclusion $\subseteq$.

Next, we consider $\bigcap\{\xi \in \mathsf{UMRel}(A) \mid \varphi_R(\xi) \subseteq \xi\}$. For convenience, $\mathcal{M}_R$ denotes the set $\{\xi \in \mathsf{UMRel}(A) \mid R; \xi + 1 \subseteq \xi\}$. It is obvious that $1 \in \bigcap \mathcal{M}_R$ and $R \in \bigcap \mathcal{M}_R$, since $1 \subseteq \xi$ and $R \subseteq \xi$ for each $\xi \in \mathcal{M}_R$.
The following two conditions are sufficient to show that $\bigcap \mathcal{M}_R$ is transitive.

$$R; (\bigcap \mathcal{M}_R) \ \subseteq \ \bigcap \mathcal{M}_R \tag{6}$$

$$R; P \subseteq P \Longrightarrow (\bigcap \mathcal{M}_R); P \subseteq P \ . \tag{7}$$

Lemma 3.

$$R; (\bigcap \mathcal{M}_R) \subseteq \bigcap \mathcal{M}_R$$

□

We need some proofs for (7). So we consider the following structure.

Definition 3. *For each* $P, R \in \mathsf{UMRel}(A)$,

$$\mu[P, R] := \bigcap\{S \mid P; S + R \subseteq S\}$$

□

From this definition, we see that

$$(x, X) \in \mu[P, R] \ \textit{iff} \ \forall S.[\ P; S + R \subseteq S \Rightarrow (x, X) \in S\ ].$$

Lemma 4. *For each* $P, Q, R \in \mathsf{UMRel}(A)$,

$$\mu[P, R]; Q = \mu[P, R; Q]$$

□

By the definition of $\bigcap \mathcal{M}_R$ and $\mu[P, R]$, we obtain that $\bigcap \mathcal{M}_R = \bigcap\{Q \mid R; Q + 1 \subseteq Q\} = \mu[R, 1]$. So we have the following property.

Lemma 5.

$$R; P \subseteq P \Longrightarrow (\bigcap \mathcal{M}_R); P \subseteq P$$

□

Therefore, $\bigcap \mathcal{M}_R$ satisfies (6) and (7). It is immediate that $\bigcap \mathcal{M}_R$ is transitive, i.e.

$$(\bigcap \mathcal{M}_R); (\bigcap \mathcal{M}_R) \subseteq \bigcap \mathcal{M}_R \ .$$

We have already shown that $\bigcap \mathcal{M}_R$ includes R and is reflexive and transitive. The following property is sufficient to show that $\bigcap \mathcal{M}_R$ is the least reflexive transitive up-closed multirelation including an up-closed multirelation R.

Lemma 6. *Let $R \in \mathsf{UMRel}(A)$ and $\chi \in \mathsf{UMRel}(A)$ be reflexive, transitive, and including R. Then $\bigcap \mathcal{M}_R \subseteq \chi$.* □

We have already proved the following.

Theorem 1. *$\bigcap \mathcal{M}_R$ is the reflexive transitive closure of an up-closed multirelation R.* □

5 The Star

For an up-closed multirelation R we define R^* as

$$R^* = \bigcap \{\xi \in \mathsf{UMRel}(A) \mid R; \xi + 1 \subseteq \xi\} \ .$$

The reflexivity of R^* and Lemma 3 show the star satisfies (2). (3) has also already been shown by Lemma 5.

Therefore we have the following.

Theorem 2. *A tuple $(\mathsf{UMRel}(A), +, ;, ^*, 0, 1)$ satisfies all conditions of lazy Kleene algebras.* □

This tuple need not satisfy (5) because of the following example.

Example 1. We consider $A := \mathbb{N}$, and $P, R \in \mathsf{UMRel}(A)$ such that

$$P = \{(n, W) \in A \times \wp(A) \mid W \text{ is infinite set.}\}$$
$$R = \{(n, W) \in A \times \wp(A) \mid \exists m \in W. n \leq m + 1\}$$

In this case $P; (R + 1) = P; R$ since $1 \subseteq R$, and $P; R \subseteq P$. So $P; (R + 1) \subseteq P$. Also it is shown by induction on n that $(n, \{0\}) \in R^n$ for each n. For a natural number n, $(n, \{0\}) \in P; R^*$ since $(n, \mathbb{N}) \in P$, $(n, \{0\}) \in R^n$ and $R^n \subseteq R^*$, but $(n, \{0\}) \notin P$. □

6 Conclusion

This paper has studied up-closed multirelations. Then we have shown that the set of up-closed multirelations over a set is a lazy Kleene algebra, where

- the zero element is given by null multirelation,
- the unit element is given by the identity multirelation,
- the addition is given by binary union,
- the multiplication is given by the composition of multirelations, and
- the star is given by the reflexive transitive closure.

This lazy Kleene algebra is not a probabilistic Kleene algebra by Example 1.

References

1. H. Furusawa, N. Tsumagari, and K. Nishizawa. A Non-Probabilistic Relational Model of Probabilistic Kleene Algebras. To appear in R. Berghammer, B. Möller and G. Struth, eds., a volume of LNCS.
2. V. Goranko. The Basic Algebra of Game Equivalences. Studia Logica 75(2): 221-238 (2003).
3. C. Martin, S. Curtis, and I. Rewitzky. Modelling Nondeterminism. In D. Kozen, ed., *Mathematics of Program Construction*, volume 3125 of LNCS, 228-251. Springer, 2004.
4. D. Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation*, 110: 366-390 (1994).
5. A. McIver and T. Weber. Towards Automated Proof Support for Probabilistic Distributed Systems. In G. Sutcliffe and A. Voronkov, eds., *Proceedings of Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of LNAI, 534-548. Springer, 2005.
6. A. McIver, E. Cohen, and C. Morgan. Using Probabilistic Kleene Algebra for Protocol Verification. In R. A. Schmidt, ed., *Relations and Kleene Algebra in Computer Science*, volume 4136 of LNCS, 296-310. Springer, 2006.
7. B. Möller. Lazy Kleene Algebra. In D. Kozen, ed., *Mathematics of Program Construction*, volume 3125 of LNCS, 252-273. Springer, 2004.
8. M. Pauly and R. Parikh. Game Logic - An Overview. Studia Logica 75(2): 165-182 (2003).
9. R. Parikh. The Logic of Games. Annals of Discrete Mathematics 24: 111-140 (1985).
10. I. Rewitzky. Binary Multirelations. In H. de Swart, E. Orlowska, G. Schmidt, M. Roubens, eds., *Theory and Applications of Relational Structures as Knowledge Instruments*, volume 2929 of LNCS, 256-271. Springer, 2003.
11. I. Rewitzky and C. Brink. Monotone Predicate Transformers as Up-Closed Multirelations. In R. A. Schmidt, ed., *Relations and Kleene Algebra in Computer Science*, volume 4136 of LNCS, 311-327. Springer, 2006.
12. Y. Venema, Representation of Game Algebras. Studia Logica 75(2): 239-256 (2003).

Author Index