On Automating the Calculus of Relations

Peter Höfner[1] and Georg Struth[2]

[1] Institut für Informatik, Universität Augsburg, Germany
hoefner@informatik.uni-augsburg.de
[2] Department of Computer Science, University of Sheffield, United Kingdom
g.struth@dcs.shef.ac.uk

Abstract. Relation algebras provide abstract equational axioms for the calculus of binary relations. They name an established area of mathematics and have found numerous applications in computing. We prove more than hundred theorems of relation algebras with off-the-shelf automated theorem provers. They form a basic calculus from which more advanced applications can be explored. We also present two automation experiments from the formal methods literature. Our results further demonstrate the feasibility of automated deduction with complex algebraic structures. They also open a new perspective for automated deduction in relational formal methods.

1 Introduction

Relations are among the most ubiquitous concepts in mathematics and computing. Relational calculi have their origin in the late nineteenth century and their initial development was strongly influenced, but then overshadowed, by the advancement of mathematical logic. Around 1940, Alfred Tarski revived the subject by formalising the calculus of binary relations alternatively within the three-variable fragment of first-order logic and as an abstract relation algebra within first-order equational logic [28]. Today, relation algebras form an established field of mathematics with numerous textbooks and research publications.

The relevance of relational calculi in computing has been realised since the early beginnings. Relational approaches had considerable impact on program semantics, refinement and verification through the work of Dijkstra, Hoare, Scott, de Bakker, Back and others. Formal methods like Alloy [14], B [2] and Z [25], or Bird and de Moor's algebraic approach to functional program development [6] are strongly relational. Further applications of relations in computing include data bases, graphs, preference modelling, modal reasoning, linguistics, hardware verification and the design of algorithms. Here, our main motivation is program development and verification.

To support relational formal methods, various tools have been developed. Interactive proof-checkers for relation algebras have been implemented [30, 16] and relational techniques have been integrated into various proof checkers for B or Z. Special purpose first-order proof systems for relation algebras, including tableaux and Rasiowa-Sikorski calculi, have been proposed [18, 17]. Translations

of relational expressions into (undecidable) fragments of predicate logics have been implemented [24] and integrated into the SPASS theorem prover [31]. Finite relational properties can efficiently be analysed with tools similar to model checkers [4]. Modal logics can automatically be translated into relational calculi to be further analysed by the relational tools available [10]. But is it really necessary to base automated reasoning with relation algebras either on interactive theorem provers, on special-purpose calculi or on finitary methods? Would off-the-shelf automated theorem provers (APT systems) necessarily fail to derive anything interesting when provided with Tarski's equational axioms?

This paper provides the proof of concept that a direct axiomatic integration of relation algebras into modern off-the-shelf ATP systems is indeed feasible. Our main contributions are as follows: First, we identify axiomatisations of relation algebras that are particularly suitable for proof search. Second, we prove more than hundred theorems of relation algebra with the ATP system Prover9 and the counterexample generator Mace4 [21]. These experiments mechanise the calculus from a standard mathematical textbook [20] and yield a verified basis on which further applications can be built. Experiments show that Prover9 and Waldmeister are currently best suited for this task [8]. Third, we present two extended examples that further demonstrate the applicability of the approach. The first one automates an example from Abrial's B-Book [2] that analyses kinship relations in a fictitious society. The second one automatically analyses simulation laws in data refinement. In addition, we consider relation algebras as Boolean algebras with operators, thus take the initial steps towards reasoning automatically about modalities in this approach. Detailed information, including all input and output files (in TPTP format) can be found at a website [1]. Selected theorems will become part of the TPTP library in summer 2008 [26].

Our overall experience is positive: Many textbook-level theorems and calculational proofs that eminent mathematicians found worth publishing some decades ago can nowadays be automatically verified within a few minutes from the axioms of relation algebras by off-the-shelf ATP systems. More complex statements require either inequational reasoning, for which we also provide an axiomatisation, or "learning" of the hypotheses, for which we use heuristics. In conclusion, the axiomatic integration of computational algebras into off-the-shelf ATP systems offers a simple yet powerful alternative to interactive approaches, special-purpose procedures and finitary methods. A new kind of application of automated deduction in formal methods seem therefore possible.

2 Binary Relations and Relation Algebras

Binary relations and their basic operations feature in most introductory courses on discrete mathematics. More information can be found, e.g., in the textbooks by Maddux [19], and Schmidt and Ströhlein [23].

A binary relation R on a set A is just a subset of $A \times A$ — a set of ordered pairs. Since relations are sets, unions $R \cup S$, intersections $R \cap S$ and complements $\overline{R}$ of relations can be taken such that the set of all binary relations forms a

Boolean algebra. The *relative product* $R;S$ of two relations R and S is the set of all pairs (a,b) such that $(a,c) \in R$ and $(c,b) \in S$ for some $c \in A$. The *converse* $\breve{R}$ of a relation R is the set of all pairs (a,b) with $(b,a) \in R$. The *identity relation* 1_A on A is the set of all pairs (a,a) with $a \in A$. The structure $(2^{A^2}, \cup, ;, \overline{\ }, \breve{\ }, 1_A)$ is called proper relation algebra of all binary relations over A. To define relation algebras more abstractly, binary relations are replaced by arbitrary elements of some carrier set A and a set of equational axioms is given.

A *relation algebra* is a structure $(A, +, ;, \overline{\ }, \breve{\ }, 1)$ satisfying the axioms

$$(x+y)+z = x+(y+z) \ , \qquad x+y = y+x \ , \qquad x = \overline{\overline{x}+\overline{y}}+\overline{\overline{x}+y} \ ,$$
$$(x;y);z = x;(y;z) \ , \qquad (x+y);z = x;z+y;z \ , \qquad x;1 = x \ ,$$
$$\breve{\breve{x}} = x \ , \qquad (x+y)\breve{\ } = \breve{x}+\breve{y} \ , \qquad \breve{x};\overline{x;y}+\overline{y} = \overline{y} \ .$$

We assume that relative products bind more strongly than joins and meets and that complementation and converse bind more strongly than relative products. The first line contains Huntington's axioms for Boolean algebras [13, 12]. The join operation is denoted by $+$, and the meet of Boolean algebra can be defined as $x \cdot y = \overline{\overline{x}+\overline{y}}$. Since relation algebras are Boolean algebras, they form posets with respect to $x \leq y \Leftrightarrow x+y = y$ that have greatest elements $\top = x+\overline{x}$ and least elements $0 = x \cdot \overline{x}$. The axioms in the second line define the relative product. The axioms in the third line define the operation of conversion. A TPTP-encoding is given in Appendix A.

On the one hand, this axiomatisation is very compact and initial experiments suggest that it is particularly suitable for automation. Further hypotheses can easily be added by need while keeping the initial set small. On the other hand, humans may find it difficult to prove even simple facts from these axioms alone.

A relation algebra is representable iff it is isomorphic to a proper relation algebra, but not all relation algebras have that property. This means that they are too weak to prove some truths about binary relations. But this weakness is balanced by a strength: While the expressiveness of the calculus of binary relations is precisely that of the three-variable fragment of first-order logic, a translation into logic can introduce quite complex expressions with nested quantifiers and destroy the inherent algebraic structure of a statement. This can obfuscate the decomposition of complex theorems into lemmas and the control of hypotheses needed in proofs. Relation algebras can yield simpler, more modular and more concise specifications and proofs. This situation is similar to pointwise versus pointfree functional programming. Relation algebras are also interesting for ATP systems because already the equational theory is undecidable.

3 Boolean Algebras: A Warm-Up

The automated analysis of Boolean algebras is one of the great success stories of automated deduction [22]; proofs with these and similar lattices are well-documented through the TPTP library and various publications. Here and in the next sections we follow Maddux's book [19] in proving a series of theorems that

constitute the core calculus of relation algebras. The series starts with proving Boolean properties from Huntington's axioms that are later useful in relational proofs. All experiments used a Pentium 4 processor with 3 GHz and Hyper-Threading and 2 GB memory. All input files in TPTP-format and all Prover9 outputs can be found at a website [1]. Here, we only briefly summarise our results. As a default, we proved our goals from the full axiom set. All deviations are explicitly mentioned.

Proposition 3.1. *Huntington's axioms imply the standard equational axioms for Boolean algebras.*

By "standard" we mean the usual equational lattice axioms, the distributivity law(s) and the axioms for Boolean complementation that can be found in any book on lattices. In particular, Prover9 showed in less than one second that $x + \overline{x} = y + \overline{y}$ and $x \cdot \overline{x} = y \cdot \overline{y}$. This expresses the existence of a unique greatest element $\top$ and a unique least element 0 such that $x + \overline{x} = \top$ and $x \cdot \overline{x} = 0$.

Lattices, Boolean algebras, and therefore relation algebras can also be defined as posets, and order-based proofs by hand are usually easier. A proof of $x = y$ can be achieved by proving $x \leq y$ and $y \leq x$ separately. We therefore provide an order-based encoding as an alternative to the equational one by adding the definition $x \leq y \Leftrightarrow x + y = y$. Join, meet, relative product and conversion are isotone with respect to $\leq$; complementation is antitone. For example,

$$x \leq y \Rightarrow z; x \leq z; y \quad \text{and} \quad x \leq y \Rightarrow \overline{y} \leq \overline{x} \ .$$

Reflexivity and transitivity of $\leq$ and the monotonicity properties can be very useful additional hypotheses in more complex examples. Our general experience is that order-based automated proofs are better when there is a simple order-based proof by hand. But there are exceptions to the rule and in advanced applications, both variants should be explored.

Proposition 3.2. *Huntington's axioms and the order-based axioms for Boolean algebras are equivalent.*

Prover9 needed less than one second for the left-to-right direction and less than two minutes for its converse.

In sum, we proved more than 40 theorems about Boolean algebras (the TPTP-library currently contains around 100). Prover9 succeeded with every single task we tried. This basic library of automatically verified statements can be used as hypotheses for more advanced applications with relation algebras.

4 Boolean Algebras with Operators

Boolean algebras provide the foundations for more complex structures such as relation algebras, cylindric algebras [11] or modal algebras [7]. All these structures can be understood as Boolean algebras with operators, which makes them interesting candidates for ATP. But to our knowledge, an axiom-based approach

with off-the-shelf ATP systems has not yet been attempted. This section further follows Maddux's book, but also Jónsson and Tarski's seminal article on Boolean algebras with operators [15] by automatically proving some of their calculational statements. Since the proofs in these sources are essentially order-based, we strongly rely on that axiomatisation for Prover9.

According to a classical definition by Jónsson and Tarski, two functions f and g on a Boolean algebra are *conjugate* if they satisfy

$$f(x) \cdot y = 0 \Leftrightarrow x \cdot g(y) = 0 \ .$$

With the order-based encoding of Boolean algebras and the conjugation property, Prover9 could verify a series of laws that is documented in Table 1. Again, the most important laws can be organised into a lemma.

#	Theorem	t[s]	#	Theorem	t[s]
(1)	$f(x+y) \leq z \Leftrightarrow f(x) + f(y) \leq z$	182.51	(2)	$f(x+y) = f(x) + f(y)$	0.16
(3)	$f(0) = 0$	0.10	(4)	$x \leq y \Rightarrow f(x) \leq f(y)$	14.98
(5)	$f(\overline{g(x)}) \leq \overline{x}$	138.93	(6)	$f(x \cdot y) \leq f(x) \cdot f(y)$	147.64
(7)	$f(x) + f(y) \leq f(x+y)$	141.11	(8)	$f(x) \leq y \Rightarrow x \leq \overline{g(\overline{y})}$	34.81
(9)	$f(x) \leq y \Leftarrow x \leq \overline{g(\overline{y})}$	8.10	(10)	$f(x) \cdot y \leq f(x \cdot g(y)) \cdot y$	241.92
(11)	$\forall x, y.(f(x) \cdot y = 0 \Leftrightarrow x \cdot h(y) = 0) \Rightarrow \forall z.(g(z) = h(z))$				86.75
(12)	$f(x \cdot \overline{g(y)}) \leq f(x) \cdot \overline{y}$	144.81			

Table 1. Laws for Conjugates

Lemma 4.1. *Conjugate functions on a Boolean algebra*

(i) are strict and additive;
(ii) induce a Galois connection;
(iii) satisfy modular laws;
(iv) are in one-to-one correspondence.

Proof. (i) holds by Equation (1), (2) and (3); (ii) holds by Equation (8) and (9); (iii) holds by Equation (10); (iv) holds by Equation (11). □

Often, a law of the form $f(x) \cdot y \leq f(x \cdot g(y))$ is called *modular law*. Other properties displayed in Table 1 are isotonicity of conjugates (Equation (4)), a cancellation law (Equation (5)) and a subdistributivity law (Equation (6)).

Most of the theorems could be proved entirely from Huntington's axioms and the conjugation axiom, but some more complex ones such as (10) required the addition of additional hypotheses, like the standard distributivity laws.

We used a manual form of "hypothesis learning" which can easily be implemented as an automated procedure. To this end, we started with very small axiom sets from which "explosive" axioms such as commutativity had been discarded. We then added further hypotheses until a counterexample generator failed and there was hope that the hypotheses are strong enough to entail the

goal.The selection of these hypotheses was usually based on a mixture of semantic knowledge and blind guessing. We then tried the ATP system and repeated the procedure with different hypotheses if the proof search failed within reasonable time. Usually, we set a time limit of $300\,s$. It has been experimentally confirmed that the probability that an ATP system will prove a theorem beyond that threshold is very low [27]. Interestingly, we frequently encountered situations where hypotheses that were crucial for success were not needed in the proof itself, but acted as catalysers for redundancy elimination. Detailed information about all deviations from the standard axiomatisation is provided at our website [1].

The results of this section make some proofs in relation algebras (e.g., the modular laws) simpler and more convenient. But they are also of wider interest. Boolean algebras with operators are algebraic variants of (multi)modal logics. Our experiments therefore suggest that the automation of modal logics through the combination of off-the-shelf ATP systems with algebras might be a feasible alternative to existing special-purpose calculi and decision procedures that extends to undecidable first-order modal logics.

5 Relation Algebras

Relation algebras can be perceived as Boolean algebras with operators corresponding to functions like $\lambda x.a; x$. As already mentioned, we follow Maddux's first-order axiomatisation. There are second-order variants of relation algebras in which the underlying Boolean algebra is assumed to be complete and atomic [23]. Relation algebras are quite rich and complex structures; they are expressive enough for modelling set theory [29]. Again we can group some of our experiments into lemmas.

Lemma 5.1. *Relation algebras are idempotent semirings with respect to join and composition.*

Proof. An idempotent semiring is a structure $(S, +, ;, 0, 1)$ such that $(S, +, 0)$ is a commutative idempotent monoid, $(S, ;, 1)$ is a monoid, multiplication distributes over addition from left and right, and 0 is a left and right annihilator, i.e., $0; a = 0 = a; 0$. The facts needed for proving this are shown in Table 2. □

#	Theorem	t[s]
(13)	$x;(y+z) = x;y + x;z$	3.9
(14)	$1;x = x$	0.06
(15)	$0;x = 0$	
(16)	$x;0 = 0$	

Table 2. Relational Semiring Laws

Lemma 5.2. *In relation algebras,*

#	Theorem	t[s]
(17)	$x \leq y \Leftrightarrow \breve{x} \leq \breve{y}$	0.13
(18)	$x \leq y \Rightarrow x;z \leq y;z$	100.31
(19)	$x \leq y \Rightarrow z;x \leq z;y$	

Table 3. Isotonicity Laws

#	Theorem	t[s]
(20)	$x;y \cdot z = 0 \Rightarrow y \cdot \breve{x};z = 0$	287.82
(21)	$x;y \cdot z = 0 \Leftarrow y \cdot \breve{x};z = 0$	264.49

Table 4. Schröder Laws

(i) relative products and conversion are isotone;
(ii) the maps $\lambda y.x;y$ and $\lambda y.\breve{x};y$ are conjugates.

Proof. See Table 3 for (i) and Table 4 for (ii). □

The equivalence expressed by Equations (20) and (21) is called *Schröder law*. This law is one of the working horses of relation algebras; often in the equivalent and the dual form $x;y \leq \overline{z} \Leftrightarrow \breve{x};z \leq \overline{y} \Leftrightarrow z;\breve{y} \leq \overline{x}$. The Schröder laws are of course also very helpful additional hypotheses for ATP systems.

The fact that the Schröder laws express conjugation shows a significant limitation of the first-order approach. In a higher-order setting it would now be possible to transfer all generic properties of conjugate functions or adjoints of a Galois connection to the relational level. It would also be possible to exploit the semiring duality that links the two equivalences of the Schröder laws. In the pure first-order setting, all this work remains explicit.

Having identified the above conjugation it is evident that modular laws hold in relation algebras, too. But while an automation in Boolean algebra with operators was possible, we did not succeed in relation algebra without an axiom restriction. The reason is that the operation of function application, which is present in Boolean algebras with operators but not in relation algebras, reduces the applicability of associativity of composition and prunes the search space. To learn the appropriate restriction, we reused the axioms listed in the Prover9 output for Boolean algebras with operators. This was the key to success.

Lemma 5.3. *The modular laws and the Dedekind law hold in relation algebras.*

Proof. See Table 5. □

#	Theorem	t[s]
(22)	$x;(y \cdot z) \leq x;y \cdot x;z$	25.34
(23)	$z;x \cdot y \leq z;(x \cdot \breve{z};y) \cdot y$	5444.61
(24)	$z;x \cdot y \leq (z \cdot y;\breve{x});(x \cdot \breve{z};y)$	3.28

Table 5. Modular and Dedekind Laws

The Dedekind law (24) is another fundamental law of relation algebras. It is also the most complex law automated in this section. We had to restrict the relation algebra axioms and to add the modular laws as further hypotheses.

A rich calculus can be developed from these basic laws. Most of the laws we tried could again be proved without any restriction from the relation algebra axioms and with reasonable running times. Examples are displayed in Table 6.

#	Theorem	t[s]
(25)	$\breve{0} = 0$	0.35
(26)	$\breve{\top} = \top$	
(27)	$\breve{\overline{x}} = \overline{\breve{x}}$	
(28)	$(x \cdot y)\breve{} = \breve{x} \cdot \breve{y}$	
(29)	$0 = \breve{x} \cdot y \Leftrightarrow 0 = x \cdot \breve{y}$	0.45
(30)	$\breve{1} = 1$	0.03
(31)	$x \leq x; \top$	0.26
(32)	$x \leq \top; x$	
(33)	$\top; \top = \top$	
(34)	$x; y \cdot \overline{x; z} = x; (y \cdot \overline{z}) \cdot \overline{x; z}$	184.71
(35)	$\overline{y; x}; \breve{x} \leq \overline{y}$	3.81

Table 6. Further Relational Laws

Our experiments show that the calculus of relation algebras, as presented in textbooks, can be automated without major obstacles. This does not mean that this calculus is trivial. Novices might find it difficult to prove the laws in this section by hand from the axioms given.

6 Functions, Vectors and other Concepts

Relation algebras allow the abstract definition of various concepts, including functions, vectors, points, residuals, symmetric quotients or subidentities, and the proof of their essential properties [19, 23]. These and more advanced concepts are important, for instance, for program development with Alloy, B or Z, or for the construction and verification of functional programs. Abrial's B-Book [2], in particular, contains long lists of algebraic properties involving these concepts that have been abstracted from concrete binary relations.

A *vector* (or *subset*) is an element x of a relation algebra that satisfies $x; \top = x$. An intuition can perhaps best be provided through finite relations. These can be represented as Boolean matrices with ones denoting that elements corresponding to rows and columns are ordered pairs. In this setting, vectors correspond to row-constant matrices, which are the only matrices that are preserved under multiplication with the matrix that contains only ones. Prover9 could easily verify a series of basic properties of vectors. They are displayed in Table 7. To speed up proofs we sometimes added some natural hypotheses like monotonicities. Again, all details can be found at our website [1].

For proving (41) and (42) we also used the Dedekind law; the other statements did not require further hypotheses.

A *test* (or subidentity) is an element below 1. Prover9 could prove a series of laws for tests that are displayed in Table 8. They immediately imply that the subalgebra of subidentities of a relation algebra is a Boolean algebra.

#	Theorem	t[s]
(36)	$x;\top = x \Rightarrow \overline{x};\top = \overline{x}$	0.14
(37)	$x;\top = x \;\&\; y;\top = y \Rightarrow (x \cdot y);\top = x \cdot y$	0.02
(38)	$x;\top = x \Rightarrow (x \cdot 1);y = x \cdot y$	0.13
(39)	$x;\top = x \Rightarrow (y \cdot \breve{x});(x \cdot z) \leq y;(x \cdot z)$	0.22
(40)	$x;\top = x \Rightarrow (y \cdot \breve{x});(x \cdot z) \leq (y \cdot \breve{x});z$	0.27
(41)	$x;\top = x \Rightarrow (y \cdot \breve{x});(x \cdot z) \geq y;(x \cdot z)$	1.46
(42)	$x;\top = x \Rightarrow (y \cdot \breve{x});(x \cdot z) \geq (y \cdot \breve{x});z$	1.61

Table 7. Vector Laws for Relation Algebras

#	Theorem	t[s]
(43)	$x \leq 1 \Rightarrow \breve{x} = x$	15.26
(44)	$x \leq 1 \Rightarrow x;\top \cdot y = x;y$	63.38
(45)	$x \leq 1 \Rightarrow \overline{x;\top} \cdot 1 = \overline{x} \cdot 1$	15.22
(46)	$x \leq 1 \;\&\; y \leq 1 \Rightarrow x;y = x \cdot y$	8.84
(47)	$x \leq 1 \;\&\; y \leq 1 \Rightarrow x;z \cdot y;z = (x \cdot y);z$	129.94
(48)	$x \leq 1 \Rightarrow x;y \cdot \overline{z} = x;y \cdot \overline{x;z}$	63.71

Table 8. Test Laws for Relation Algebras

A (partial) *function* is an element x of a relation algebra satisfying $\breve{x};x \leq 1$. This condition concisely expresses that no domain element of a function can be mapped to more than one range element. Facts about functions are displayed in Table 9. Equation (49) says that functions are closed under composition.

#	Theorem	t[s]
(49)	$\breve{x};x \leq 1 \;\&\; \breve{y};y \leq 1 \Rightarrow \breve{(x;y)};x;y \leq 1$	11.18
(50)	$\breve{x};x \leq 1 \Rightarrow x;(y \cdot z) = x;y \cdot x;z$	740.08
(51)	$\breve{x};x \leq 1 \Rightarrow x;y \cdot x;\overline{y} = 0$	0.15
(52)	$x \leq 1 \Rightarrow \breve{x};x \leq 1$	0.01

Table 9. Laws for Functions

In Equation (51) we used the distributivity law (50) to reduce waiting time. Additional experiments with functions are again presented at our website.

An element of a relation algebra is *total* if $1 \leq x;\breve{x}$. It is *injective* if its converse is a function and it is *surjective* if its converse is total. Simple properties like the ones in Table 9 can again easily be automated. Basic results for other entities and derived operations are also feasible.

Finally, it can be verified that the Tarski rule $x \neq 0 \Leftrightarrow \top;x;\top = \top$ is not implied by Maddux's axiomatisation of relation algebra. Mace4 produces a counterexample with 4 elements within a few seconds.

The experiments of this section show that properties of many standard mathematical concepts can easily be proved automatically from our relational basis.

Including the proofs in Boolean algebras, our website documents more than 100 theorems about relation algebras.

7 Abrial's Relatives

This section contains a first example that further demonstrates the power of relation algebra combined with state-of-the-art ATP. Abrial's B-Book [2] contains a nice non-programming application of relational reasoning by analysing kinship relations. To make the example more interesting (and less realistic), he imposes some severe restrictions on marriages and families. Abrial's example is appealing because its specification is compact, all relational operations occur, and proofs are short, but non-trivial for humans. The following list shows Abrial's initial assumptions on kinship relations, but we formalise them in a slightly different, more pointfree way.

1. PERSON is a set (cf. Section 6): $\mathsf{PERSON};\top = \mathsf{PERSON}$.
2. No person can be a man and a woman at the same time: $\mathsf{Men} \cdot \mathsf{Women} = 0$.
3. Every person is either a man or a woman: $\mathsf{Men} + \mathsf{Women} = \mathsf{PERSON}$.
4. Only women have husbands, who must be men:
   $\overline{\mathsf{Women}};\mathsf{Husband} = 0 \ \wedge \ \mathsf{Husband};\overline{\mathsf{Men}} = 0$.
5. Women have at most one husband: $\mathsf{injective}(\mathsf{Husband})$.
6. Men have at most one wife: $\mathsf{Wife} = \mathsf{Husband}^\smile \ \wedge \ \mathsf{injective}(\mathsf{Wife})$.
7. Mothers are married women: $\mathsf{Mother} \leq \mathsf{Women} \ \wedge \ \mathsf{Mother};\overline{\mathsf{Husband}} = 0$.

Abrial then defines further concepts from the ones just introduced.

$$
\begin{aligned}
\mathsf{Spouse} &= \mathsf{Husband} + \mathsf{Wife} \ , \\
\mathsf{Father} &= \mathsf{Mother};\mathsf{Husband} \ , \\
\mathsf{Children} &= (\mathsf{Mother} + \mathsf{Father})^\smile \ , \\
\mathsf{Daughter} &= \mathsf{Children};\mathsf{Women} \ , \\
\mathsf{Sibling} &= (\mathsf{Children}^\smile;\mathsf{Children}) \cdot \overline{1} \ , \\
\mathsf{Brother} &= \mathsf{Sibling};\overline{\mathsf{Women}} \ , \\
\mathsf{SiblingInLaw} &= \mathsf{Sibling};\mathsf{Spouse} + \mathsf{Spouse};\mathsf{Sibling} + \mathsf{Spouse};\mathsf{Sibling};\mathsf{Spouse} \ , \\
\mathsf{NephewOrNiece} &= (\mathsf{Sibling} + \mathsf{SiblingInLaw});\mathsf{Children} \ , \\
\mathsf{UncleOrAunt} &= \mathsf{NephewOrNiece}^\smile \ , \\
\mathsf{Cousin} &= \mathsf{UncleOrAunt};\mathsf{Children} \ .
\end{aligned}
$$

The specification of Sibling, in particular, may deserve some explanation. The relation $\mathsf{Children}^\smile;\mathsf{Children}$ links each child not only with its siblings, but also with itself. Intersecting with $\overline{1}$ eliminates the reflexive part of this relation and thus yields the real siblings.

Based on this specification, Abrial presents ten proof tasks which are shown in Table 10, except for a pointwise law the proof of which would require additional axioms. Proofs of the last four facts from Table 10 are displayed in the B-Book and they alone cover more than two pages.

#	Theorem	t[s]
(53)	$\mathsf{Mother} = \mathsf{Father}; \mathsf{Wife}$	
(54)	$\mathsf{Spouse} = \mathsf{Spouse}^\smile$	
(55)	$\mathsf{Sibling} = \mathsf{Sibling}^\smile$	
(56)	$\mathsf{SiblingInLaw} = \mathsf{SiblingInLaw}^\smile$	1.54
(57)	$\mathsf{Cousin} = \mathsf{Cousin}^\smile$	
(58)	$\mathsf{Father}; \mathsf{Father}^\smile = \mathsf{Mother}; \mathsf{Mother}^\smile$	
(59)	$\mathsf{Father}; \mathsf{Mother}^\smile = 0$	
(60)	$\mathsf{Mother}; \mathsf{Father}^\smile = 0$	
(61)	$\mathsf{Father}; \mathsf{Children} = \mathsf{Mother}; \mathsf{Children}$	1.48

Table 10. Kinship Relations

8 Simulation Laws for Data Refinement

Program refinement investigates the stepwise transformation of abstract specifications to executable code. Data refinement is a variant that considers the transformation of *abstract* data types (ADTs) such as sets into *concrete* ADTs such as lists, stacks or queues. Abstract ADTs are observed through the effects of their operations on states, and operations are usually modelled as binary relations. Two further operations model the initialisation and finalisation of ADTs with respect to a global state space. By definition, an abstract ADT is refined by a concrete ADT if the relation induced by all execution sequences of abstract operations between an abstract initialisation and an abstract finalisation is contained in the relation induced by the corresponding concrete sequences. To replace this by a local criterion, abstraction relations are introduced that relate inputs and outputs of operations at the abstract and the concrete level. de Roever and Engelhardt's book contains further information [9]. Program refinement often requires inequational reasoning. Therefore we used isotonicity of multiplication and transitivity as additional hypothesis in our experiments. Named by de Roever and Engelhardt according to the shape of the corresponding diagrams, U-simulations, L-simulations and their converses can be considered. Formally, let x, y and z be elements of some relation algebra. Then

- x *U-simulates* y with respect to z ($x \subseteq^z_U y$) if $\breve{z}; x; z \leq y$,
- x *L-simulates* y with respect to z ($x \subseteq^z_L y$) if $\breve{z}; x \leq y; \breve{z}$,
- x *$\breve{U}$-simulates* y with respect to z ($x \subseteq^z_{\breve{U}} y$) if $x \leq z; y; \breve{z}$,
- x *$\breve{L}$-simulates* y with respect to z ($x \subseteq^z_{\breve{L}} y$) if $x; z \leq z; y$.

In all these definitions, z is the *abstraction relation* and $\subseteq$ the *simulation relation*. We now consider compositionality properties of simulations.

Theorem 8.1 ([9]). *Let z be a simulation relation.*

(i) $x_1 \subseteq^z_U y_1$ *and* $x_2 \subseteq^z_U y_2$ *imply* $x_1; x_2 \subseteq^z_U y_1; y_2$ *if z is total.*
(ii) $x_1 \subseteq^z_{\breve{U}} y_1$ *and* $x_2 \subseteq^z_{\breve{U}} y_2$ *imply* $x_1; x_2 \subseteq^z_{\breve{U}} y_1; y_2$ *if z is a function.*
(iii) $x_1 \subseteq^z_L y_1$ *and* $x_2 \subseteq^z_L y_2$ *imply* $x_1; x_2 \subseteq^z_L y_1; y_2$*, and similarly for $\breve{L}$.*

Proof. Prover9 needed less than 3 *s* for each individual claim. □

Theorem 8.2 ([9]). *$x \subseteq_L^{z_1} y$ and $x \subseteq_L^{z_2} y$ imply $x \subseteq_L^{z_1;z_2} y$, and similarly for the other simulations.*

Proof. Prover9 needed less than 1 *s* for the each implication. □

Also the implications between different simulations could be automated.

Theorem 8.3 ([9]).

(i) If the simulation relation is a function then $\breve{U}$-simulation implies L-, $\breve{L}$- and U-simulation, and U-simulation is implied by L- and $\breve{L}$-simulation.
(ii) If the simulation relation is total then U-simulation implies L-, $\breve{L}$ and $\breve{U}$-simulation, and $\breve{U}$-simulation is implied by L- and $\breve{L}$-simulation.

Proof. Prover9 presented proofs for both claims after less than 1 *s*. □

We now prove simulation laws for iterations of relations. In relation algebras over complete Boolean algebras, the finite iteration of an element x is defined through the reflexive transitive closure

$$x^* = x^0 + x^1 + x^2 + \ldots = \sup(x^i : i \geq 0) \ ,$$

where $x^0 = 1$ and $x^{i+1} = x; x^i$. This definition is rather useless for ATP systems. We therefore use the well known unfold and induction laws

$$1 + x; x^* \leq x^* \ , \qquad 1 + x^*; x \leq x^* \ ,$$
$$z + x; y \leq y \Rightarrow x^*; z \leq y \ , \qquad z + y; x \leq y \Rightarrow z; x^* \leq y \ .$$

The proof that these laws hold in relation algebras over complete Boolean algebras requires a simple induction. Based on this first-order encoding, a series of laws could easily be verified automatically. Some example experiments are listed in Table 11; more can be found at our website.

#	Theorem	t[s]
(62)	$x^*; x^* = x^*$	2.55
(63)	$(x^*)^* = x^*$	
(64)	$x; y \leq y \Rightarrow x^*; y \leq y$	0.02
(65)	$z; x \leq y; z \Rightarrow z; x^* \leq y^*; z$	2.20
(66)	$(x; y)^*; x = x; (y; x)^*$	2.05

Table 11. Relational Iteration Laws

To speed up proofs we used the join splitting law $x + y \leq z \Leftrightarrow x \leq z \wedge y \leq z$ as an additional hypothesis in the proofs of (65) and (66).

The properties from Table 11 yield useful hypotheses for automating de Roever and Engelhardt's soundness proofs of simulations for data refinement. Soundness of simulations means that the existence of a simulation between the particular operations of ADTs implies that there is a data refinement. In the sequel, we restrict our attention to L-simulations.

Theorem 8.4 ([9]). *L-simulations are sound for data refinement.*

Proof. Let i^a, x_i^a and f^a denote the initialisation, the operations and the finalisation at the abstract level; let i^c, x_i^c and f^c denote the corresponding relations at the concrete level. We can assume that $i^c \leq i^a; \breve{z}$, $x^c \subseteq_L^z x^a$ and $\breve{z}; f^c \leq f^a$ holds for arbitrary atomic operations x^a and x^c. We must show that $i^c; s^c; f^c \leq i^a; s^a; f^a$ holds for arbitrary sequences s^a and s^c of operations.

The proof uses structural induction over s^a. We first prove that $s^c \subseteq_L^z s^a$ holds for some s^c. The entire induction can, of course, not be treated by ATPs, but the particular base cases and induction steps can.

We consider the empty operation 0 and the skip operation 1 as base cases. Prover9 showed in 9.95 *s* that $0 \subseteq_L^z 0$ and $1 \subseteq_L^z 1$. The case of atomic operations holds by assumption. For the induction step we consider abstract operations of the form $s_1^a; s_2^a$, $s_1^a + s_2^a$ and $(s^a)^*$.
(i) Let $s_1^c \subseteq_L^z s_1^a$ and $s_2^c \subseteq_L^z s_2^a$. But $s_1^c; s_2^c \subseteq_L^z s_1^a; s_2^a$ has already been shown in Theorem 8.1.
(ii) Let $s_1^c \subseteq_L^z s_1^a$ and $s_2^c \subseteq_L^z s_2^a$. Using a distributivity law as additional hypothesis, Prover9 needed 1.78 *s* to show that $s_1^c + s_2^c \subseteq_L^z s_1^a + s_2^a$.
(iii) Let $s^c \subseteq_L^z s^a$. For the automated proof we used the unfold and induction laws of reflexive transitive closure and added Equation (65) as an additional hypothesis. Then Prover9 could show that $(s^c)^* \subseteq_L^z (s^a)^*$ in less than 1 *s*.

For the final step, assume that $i^c \leq i^a; \breve{z}$, $\breve{z}; f^c \leq f^a$ and $s^c \subseteq_L^z s^a$, which has just been shown. Prover9 then showed in 0.53 *s* that $i^c; s^c; f^c \leq i^a; s^a; f^a$. □

Automated proofs for the remaining simulations are also straightforward. Interestingly, de Rover and Engelhardt do not mention that U must be total and $\breve{U}$ must be a function in these proofs; Mace4 immediately found counterexamples to Part (iii) of these proofs without these assumptions. Here, we also used Equation (66) instead of (65) as additional hypothesis.

L- and $\breve{L}$-simulations are also complete for data refinement, but proofs in the literature are pointwise (cf. [9]) and additional effort would be required to extract a purely relation-algebraic proof. We leave this for future work.

9 Outlook

Relations can not only model abstract data types. They also provide standard semantics for imperative and functional programs. In the imperative case, it is very natural to model the input/output behaviour of a program as a binary relation between states encoded as vectors. The standard weakest liberal precondition semantics for partial correctness and the weakest precondition semantics for total correctness can be defined in the setting of relation algebra [5, 20]. The wlp-operator for a program x and a state p can be defined as $\mathsf{wlp}(x,p) = \overline{x;\overline{p}}$. The wp-operators with respect to a program x, a state p and a vector $\tau(x)$ denoting the guaranteed termination of x can be defined as $\mathsf{wp}(x,p) = \mathsf{wlp}(x,p) \cdot \tau(x)$. Standard laws of the $\mathsf{w(l)p}$-calculi such as $\mathsf{wlp}(x+y,p) = \mathsf{wlp}(x,p) \cdot \mathsf{wlp}(y,p)$,

$\mathsf{wp}(x+y,p) = \mathsf{wp}(x,p)\cdot\mathsf{wp}(y,p)$, $\mathsf{wlp}(x,p\cdot q) = \mathsf{wlp}(x,p)\cdot\mathsf{wlp}(x,q)$ or $\mathsf{wp}(x,p\cdot q) = \mathsf{wp}(x,p)\cdot\mathsf{wp}(x,q)$ could then be automatically derived without any difficulties.

Our examples suggest that a combination of relation algebras with ATP systems could contribute to make formal methods more automatic and user-friendly. All current verification tools for formal methods like B or Z are highly interactive. Although the translation of system specifications into relational semantics is rather simple and can yield very concise expressions, the manipulation of relational expressions in verification tasks is usually cumbersome for non-experts. Encapsulating the calculus as far as possible by using ATP behind the scenes could improve this situation. Practical verification tasks often require the integration of algebraic techniques into a wider context: Most induction proofs require higher-order reasoning, but the base case and the induction step can often be discharged algebraically. Other applications might require pointwise reasoning with concrete functions and relations and with assignments. But pointwise properties can often be abstracted into bridge-lemmas and proofs then confined to the abstract algebraic layer. In this sense we envisage the integration of relation algebras into ATP systems as a novel light-weight formal method that should be extended by higher-order techniques and combined with decision procedures.

10 Conclusion

We automatically verified more than hundred theorems of relation algebras with Prover9 and Mace4. Many of these proofs were considered worth publishing by eminent mathematicians some decades ago and most students would probably still find them difficult. Our experiments suggest that the automation of relation algebras with off-the-shelf theorem provers is feasible. This presents an interesting alternative to higher-order, special-purpose, translational and finitist approaches. The statements proved form a basic library that can safely be used and extended. Our results pave the way for interesting applications in relational software development methods and automated deduction with modal logics. A larger case study, in which the experiments of this paper have been replayed with other ATP systems [8], confirms our results.

We envisage three main directions for further work. First, to be more useful in formal methods, ways of combining the abstract pointfree level with the concrete level of data need to be developed. Second, an integration of ordered chaining techniques [3] into modern ATP systems would certainly make relational reasoning, which is predominantly inequational, more efficiently. Third, a combination with more powerful hypothesis learning techniques seems indispensable for tackling more complex applications and larger specifications. The obvious impact on formal verification technology makes these tasks certainly worth pursuing.

References

1. `http://www.dcs.shef.ac.uk/~georg/ka`.
2. J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996.

3. L. Bachmair and H. Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, 1998.
4. R. Berghammer and F. Neumann. An OBDD-based computer algebra system for relations. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtov, editors, *Computer Algebra in Scientific Computing (CASC 2005)*, volume 3718 of *LNCS*, pages 40–51. Springer, 2005.
5. R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.
6. R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1996.
7. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
8. H.-H. Dang and P. Höfner. First-order theorem prover evaluation w.r.t. relation- and Kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *RelMiCS10/AKA5 - PhD Programme*, Technical Report 2008-04, University of Augsburg, pages 48–52, 2008.
9. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 2001.
10. A. Formisano, E. G. Omodeo, and E. Orłowska. A Prolog tool for relational translations of modal logics: A front-end for relational proof systems. In B. Beckert, editor, *TABLEAUX 2005: Position Papers and Tutorial Descriptions*, Fachberichte Informatik, pages 1–11. Universität Koblenz-Landau, 2005.
11. L. Henkin, D. J. Monk, and A. Tarski. *Cylindric Algebras, Part I*. North-Holland, 1971.
12. E. V. Huntington. Boolean algebra. A correction. *Trans. AMS*, 35:557–558, 1933.
13. E. V. Huntington. New sets of independent postulates for the algebra of logic. *Trans. AMS*, 35:274–304, 1933.
14. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
15. B. Jónsson and A. Tarski. Boolean algebras with operators I. *American J. Mathematics*, 74:891–939, 1951.
16. W. Kahl. Calculational relation-algebraic proofs in Isabelle/Isar. In R. Berghammer, B. Möller, and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *LNCS*, pages 179–190. Springer, 2004.
17. W. MacCaull and E. Orlowska. Correspondence results for relational proof systems with application to the Lambek calculus. *Studia Logica*, 71(3):389–414, 2002.
18. R. Maddux. A sequent calculus for relation algebras. *Annals of Pure and Applied Logic*, 25:73–101, 1983.
19. R. Maddux. *Relation Algebras*. Elsevier, 2006.
20. R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1&2):1–85, 1996.
21. W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9`.
22. W. McCune. Solution of the Robbins problem. *J. Automated Reasoning*, 19(3):263–276, 1997.
23. G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Springer, 1993.
24. C. Sinz. System description: ARA — An automated theorem prover for relation algebras. In D. McAllester, editor, *CADE-17*, volume 1831 of *LNAI*, pages 177–182. Springer, 2000.
25. J. M. Spivey. *Understanding Z*. Cambrigde University Press, 1988.
26. G. Sutcliffe and C. Suttner. The TPTP problem library: CNF release v1.2.1. *J. Automated Reasoning*, 21(2):177–203, 1998.

27. G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131(1–2):39–54, 2001.
28. A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.
29. A. Tarski and S. R. Givant. *A Formalization of Set Theory Without Variables*. American Mathematical Society, 1987.
30. D. von Oheimb and T. F. Gritzner. Rall: Machine-supported proofs for relation algebra. In W. McCune, editor, *CADE 1994*, volume 1249 of *LNCS*, pages 380–394. Springer, 1997.
31. C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: SPASS version 3.0. In F. Pfenning, editor, *CADE 2007*, volume 4603 of *LNAI*, pages 514–520. Springer, 2007.

A Relation Algebra in TPTP-style

```
%---- Boolean algebra (Huntington)
fof(join_commutativity,axiom,(
    ! [X0,X1] : join(X0,X1) = join(X1,X0) )).

fof(associativity,axiom,(
    ! [X0,X1,X2] : join(X0,join(X1,X2)) = join(join(X0,X1),X2) )).

fof(Huntington,axiom,(
    ! [X0,X1] : X0 = join(complement(join(complement(X0),complement(X1))),
                          complement(join(complement(X0),X1))) )).

fof(meet_definiton,axiom,(
    ! [X0,X1] : meet(X0,X1) = complement(join(complement(X0),complement(X1))) )).

%---- Sequential Composition
fof(composition_associativity,axiom,(
    ! [X0,X1,X2] : composition(X0,composition(X1,X2)) =
                   composition(composition(X0,X1),X2) )).

fof(composition_identity,axiom,(
    ! [X0] : composition(X0,one) = X0 )).

fof(composition_distributivity,axiom,(
    ! [X0,X1,X2] : composition(join(X0,X1),X2) =
                   join(composition(X0,X2),composition(X1,X2)) )).

%---- Converse
fof(converse_idempotence,axiom,(
    ! [X0] : converse(converse(X0)) = X0 )).

fof(converse_additivity,axiom,(
    ! [X0,X1] : converse(join(X0,X1)) = join(converse(X0),converse(X1)) )).

fof(converse_multiplicativity,axiom,(
    ! [X0,X1] : converse(composition(X0,X1)) =
                composition(converse(X1),converse(X0)) )).

fof(converse_cancellativity,axiom,(
    ! [X0,X1] : join(composition(converse(X0),complement(composition(X0,X1))),
                complement(X1)) = complement(X1) )).
```