# RALL: Machine-supported Proofs for Relation Algebra

David von Oheimb[1] and Thomas F. Gritzner[2]

[1] Fakultät für Informatik, Technische Universität München,
D–80290 München, Germany.
E-mail: `oheimb@informatik.tu-muenchen.de`
URL: `http://www.informatik.tu-muenchen.de/~oheimb/`
[2] Universität der Bundeswehr München, D–85577 Neubiberg, Germany.
E-mail: `Thomas.Gritzner@UniBw-Muenchen.DE`

**Abstract.** We present a theorem proving system for abstract relation algebra called RALL (= Relation-Algebraic Language and Logic), based on the generic theorem prover Isabelle. On the one hand, the system is an advanced case study for Isabelle/HOL, and on the other hand, a quite mature proof assistant for research on the relational calculus. RALL is able to deal with the full language of heterogeneous relation algebra including higher-order operators and domain constructions, and checks the type-correctness of all formulas involved. It offers both an interactive proof facility, with special support for substitutions and estimations, and an experimental automatic prover. The automatic proof method exploits an isomorphism between relation-algebraic and predicate-logical formulas, relying on the classical universal-algebraic concepts of atom structures and complex algebras.

*Keywords:* Relation algebra, Isabelle, interactive and automatic theorem proving, atom structure

## 1 Introduction

Relational methods have become important to computer scientists since the early Seventies. Application fields of relational methods include the semantics and verification of computer programs and communication systems, graph theory, and database modeling, where they serve as a general means of formalization. In particular, the logical treatment of relational methods by relation algebra has received increased attention [24, 12, 28, 25, 1, 3, 7, 9], which moreover has become evident from meetings on the topic of "Relational Methods in Computer Science" (RelMiCS) [5, 10].

Usually, in research and in applications of relation algebra the validity of formulas is investigated and proved by hand. Regarding this procedure, it is most desirable to have a mechanical theorem prover helping to establish and check the proofs. This gives a maximum degree of confidence in the obtained results, which is necessary particularly for safety-critical applications in the fields mentioned above.

To obtain a theorem proving system for relation algebra, we employ the generic theorem prover *Isabelle* [22, 17, 18] as the basis of our development for two reasons. As Isabelle offers a universal meta logic and very convenient equipment for defining new object logics and proof tactics, theorem proving systems can be constructed for arbitrary logics in a relatively small amount of time. Isabelle also gives rise to some standardization in the field of theorem provers, so theorem provers based on Isabelle can be easily made available to any interested research institution.

The resulting theorem proving system, called RALL (= *R*elation-*A*lgebraic *L*anguage and *L*ogic), provides facilities for both interactive and automatic theorem proving. It supports the full language of relation algebra including higher-order operators such as arbitrary joins and meets and quantifications over relations in higher-order domain constructions such as powersets. The type-correctness of all formulas is guaranteed also for heterogeneous relation algebras. The interactive part of RALL is mainly driven by forward and backward chaining, substitutions in order to exploit relational equations, and estimations as applications of relational inclusions.

RALL also includes a facility for automatic theorem proving. Because relation algebra is a higher-order logic, an automatic proof method for relation algebra is not easy to establish. This paper employs a method that is based on the classical universal-algebraic concept of atom structures and complex algebra [13, 15, 16], generalized to heterogeneous relation algebras. The method performs a transformation of relation-algebraic formulas into propositions over the universe of atoms of a relation algebra, as the predicate logic at the level of atoms can be handled more adequately. A very important distinguishing mark of the presented method is that it carries further the abstractness of the finitely axiomatized relation algebra without reference to the logic of binary relations as predicates with two individual variables.

This paper is organized as follows: In Section 2, we introduce the notion of a (heterogeneous) relation algebra and review the logical basis of the present work. Section 3 explains how relation algebra can be implemented as an object logic of Isabelle. The interactive part of the obtained theorem proving system is given in Section 4. Section 5 describes the transformation to predicate logic and its usage for the construction of the automatic proof facility. In Section 6, we summarize the results of the paper, compare RALL with other related systems, and discuss additional topics with respect to possibly interesting future work.

## 2 Relation-Algebraic Preliminaries

This section briefly introduces the relation-algebraic background of our work.

### 2.1 Relation Algebras

The present work is based on the notion of a *heterogeneous abstract relation algebra* that stems from the algebraisation of the calculus of relations, the domain and codomain of which are (in general) not of isomorphic type.

On the one hand, relation algebras have a set-theoretic character, i.e. the set of all relations with same type form a Boolean lattice. Therefore, there are the Boolean algebra operators join, meet, and complement (allowing formulas like $\mathtt{Q}\sqcup(\mathtt{R}\sqcap-\mathtt{S})$ for relations $\mathtt{Q}$, $\mathtt{R}$, and $\mathtt{S}$ of identical type) with their well-known properties. Any Boolean lattice can also be seen from an order-theoretic aspect, with a partial ordering relation $\sqsubseteq$ and universal lower and upper bounds $\mathtt{O}$ and $\mathtt{L}$. In this work, we consider complete atomic Boolean lattices. A Boolean lattice is called *complete* if, for any (possibly infinite) set $\mathtt{A}$, the least upper bound $\mathtt{Sup(A)}$ and greatest lower bound $\mathtt{Inf(A)}$ (corresponding to arbitrary joins and meets) exist. A complete Boolean lattice is called *atomic* if any element is equal to the join of all atoms contained in it, where an *atom* is a non-zero element of a lattice that cannot be decomposed as join of other non-zero elements. For more details, see [4, 13].

On the other hand, relation algebras have a monoidal character and therefore include the operators for relational composition and transposition with their monoidal properties $\mathtt{Q}\cdot(\mathtt{R}\cdot\mathtt{S}) = (\mathtt{Q}\cdot\mathtt{R})\cdot\mathtt{S}$, $\mathtt{I}\cdot\mathtt{R} = \mathtt{R} = \mathtt{R}\cdot\mathtt{I}$, and $\mathtt{R\hat{}\hat{}} = \mathtt{R}$ (for relations $\mathtt{Q}$, $\mathtt{R}$, $\mathtt{S}$, and identity relation $\mathtt{I}$, of suitable types). The essential relation-algebraic laws are the *Schröder Equivalences* $(-\mathtt{S}\cdot\mathtt{R\hat{}} \sqsubseteq -\mathtt{Q}) = (\mathtt{Q}\cdot\mathtt{R} \sqsubseteq \mathtt{S}) = (\mathtt{Q\hat{}}\cdot-\mathtt{S} \sqsubseteq -\mathtt{R})$. A full definition of heterogeneous relation algebras, extending the classical notion of homogeneous relation algebras [26, 13, 11, 27], can be found for example in [25, A.2.1].

A fundamental issue concerning relation algebras is the question of *representability*, viz. whether their axiomatization is complete with respect to their standard set-theoretic model. Lyndon has answered this question negatively for homogeneous abstract relation algebras [15], and this statement applies also to the heterogeneous case. The non-representability of relation algebras has the consequence that, in order to obtain an automated theorem prover for abstract relation algebra, we cannot simply use predicate logic restricted to binary predicates and the translation of relation-algebraic formulas into it.

### 2.2 Relational Atom Structures

The automated prover for abstract relation algebra, described in Section 5, makes use of a correspondence between relation-algebraic formulas and relational atom structures. This subsection gives deeper insight into the underlying concepts; it may be skipped for the first reading.

The concepts of *complex algebras* and *atom structures* have been investigated for general *Boolean algebras with operators* in [13, Section 3] and [11, 2.7.32ff.]. We review these concepts for the case of (non-simple) homogeneous relation algebras, while the extension to the heterogeneous case is straightforward.

As the correspondence theorem (given below) states, any relation algebra can be alternatively represented as an atom structure with a ternary predicate corresponding to the composition operator and two functions corresponding to the identity relations and transposition operator. In the sequel, we write the prefix “pre-” to clarify the correspondence.

**Definition 1 (Relational Atom Structure).** A structure $\mathfrak{A} = (A, i, C, T)$ is called a *(non-simple) relational atom structure* iff the following conditions hold:

1. $A$ is a set;
2. $i$ is a function $A \to A$ called *trace* (yielding a *pre-identity*);
3. $C$ is a ternary relation on $A$, i.e. $C \subseteq A \times A \times A$,
   the so-called *incidence relation* (*pre-composition*);
4. $T$ is a function $A \to A$ called *pre-transposition*, where $T \circ T = id_A$;
5. the *(right) pre-identity rules*:
   $$\forall a \in A\colon\ \langle a, a, i(a)\rangle \in C \quad \text{and}$$
   $$\forall a, b, c \in A\colon\ \langle c, a, i(b)\rangle \in C \ \Rightarrow\ c = a;$$
6. the *pre-associativity rule*:
   $$\forall a, b, c, d, e \in A\colon\ \langle d, a, b\rangle \in C \wedge \langle e, d, c\rangle \in C \ \Rightarrow$$
   $$\exists f \in A\colon\ \langle e, a, f\rangle \in C \wedge \langle f, b, c\rangle \in C;$$
7. the *pre-Schröder rules*:
   $$\forall a, b, c \in A\colon\ \langle c, a, b\rangle \in C \ \Rightarrow\ \langle a, c, T(b)\rangle \in C; \quad \text{and}$$
   $$\forall a, b, c \in A\colon\ \langle c, a, b\rangle \in C \ \Rightarrow\ \langle b, T(a), c\rangle \in C.$$

The following correspondence theorem has turned out to be an important means for setting up an appropriate machine-readable representation of abstract relation algebras. In particular, [16, 8] treats the investigation and construction of several abstract models of relation algebra by computer assistance.

**Theorem 2 (Correspondence Theorem).**
*(i) For every homogeneous relation algebra* $\mathfrak{A} = (A,\sqcup,\sqcap,-,\cdot,\hat{})$*, the structure* $\mathfrak{At}(\mathfrak{A}) = (At(\mathfrak{A}), i, C, T)$ *is a (non-simple) relational atom structure, where*

- $At(\mathfrak{A})$ *denotes the set of the atoms of* $\mathfrak{A}$*;*
- *for all* $a \in At(\mathfrak{A})$: $i(a) := (a\hat{}\cdot a)\sqcap\mathtt{I}$*;* $T(a) := a\hat{}$*; and*
- $C := \{\langle c, a, b\rangle \mid a, b, c \in At(\mathfrak{A}) \ \wedge\ c \sqsubseteq a\cdot b\}$.

*(ii) Conversely, associated to every (non-simple) relational atom structure* $\mathfrak{A} = (A, i, C, T)$*, there is a (non-simple) homogeneous relation algebra* $\mathfrak{Cm}(\mathfrak{A}) = (\mathcal{P}(A),\sqcup,\sqcap,-,\cdot,\hat{})$*, called the* complex algebra of $\mathfrak{A}$*, where*

- *for all* $\mathcal{R}, \mathcal{S} \subseteq A$: $\mathcal{R}\cdot\mathcal{S} := \{c \in A \mid \exists a \in \mathcal{R}, b \in \mathcal{S}\colon \langle c, a, b\rangle \in C\}$*;*
- *for all* $\mathcal{R} \subseteq A$: $\mathcal{R}\hat{} := \{T(a) \mid a \in \mathcal{R}\}$*;*
- $\mathtt{O} := \emptyset$*;* $\mathtt{L} := A$*; and* $\mathtt{I} := \{i(a) \mid a \in A\}$.

The correspondence theorem is also essential for our construction of an automatic proof procedure presented in Section 5. Exploiting its ($i$) direction, we translate any relation-algebraic formula $R$ into a predicate-logical formula $P$ at the atomic level. A full list of the logical equivalences that are necessary for the translation is given in Section 5.1. We just state here that all those equivalences (as well as the atom-level rules of the definition above) can be deduced from the axioms of (heterogeneous) abstract relation algebra alone, which implies that any proof of the atomic-level formula $P$ also validates the corresponding relation-algebraic formula $R$.

## 3 Formalization in Isabelle/HOL

This section gives a short introduction to Isabelle, the formalization of relation algebra as a set of Isabelle theories, and the language extent supported so far.

### 3.1 Isabelle

As the basis for our proof system, we use the generic theorem prover *Isabelle* [22]. Isabelle is generic in respect to both the object logics and the employed proof techniques. It has an universal meta logic, which is an intuitionistic higher-order logic, where its syntax is based on the simply-typed lambda calculus augmented with type classes à la Haskell. This yields a quite general basis for a large variety of expressive object logics.

Proofs can be conducted interactively or as a batch proof, using Standard ML [21] as command language. The goal is stated first, and then it is transformed by forward and backward chaining with given rules into subgoals until these become trivial. The operations transforming the proof state, called *tactics*, may be combined by *tacticals* to implement very powerful application-specific proof procedures and heuristics. For most object logics a *Simplifier* is provided (to help with equational reasoning) as well as the *Classical Reasoner* [22, Part II, Chapter 11], a kind of tableaux prover that can prove many standard theorems automatically, which involves some search strategies.

The object logics are defined in *theories*, specifying their syntax, definitions, and axioms. In doing so, the elaborated parser of Isabelle gives great freedom for using infix and mixfix syntax, graphical symbols, and arbitrary *syntax translations*, allowing a very intuitive presentation of formulas.

### 3.2 HOL

Our object logic of choice for the development of RALL is *HOL* [22, Part III, Chapter 4], an implementation of Church's Simple Theory of Types [6] in Isabelle. Being a higher-order predicate logic including set theory, HOL is suitable for formalizing even the higher-order constructs of relation algebra like infinite joins and meets. Furthermore, Isabelle/HOL makes the type system of the meta logic available at the object level. Therefore, it is easy to check type-correctness (which is very important for heterogeneous relation algebras) automatically, without bothering the user with the need to prove this property by hand. HOL provides the usual introduction and elimination rules of the calculus of natural deduction as meta-level rules. As an example of the Isabelle representation of such rules, consider the elimination rule for the universal quantification (the names with a "?" at the front meaning schematic variables)

`allE = "⟦ ∀x. ?P x; ?P ?z ⟹ ?R ⟧ ⟹ ?R",`

traditionally written as

$$\frac{\forall x.\, P(x) \qquad \begin{matrix}[P(z)]\\ \vdots \\ R\end{matrix}}{R}\,(\forall E).$$

The Simplifier and the Classical Reasoner are also set up for HOL.

### 3.3 The RALL Theories

RALL is structured as a hierarchy of Isabelle theories, together with the corresponding proof and tactic definition files.

The base element of the hierarchy is the `Lattice` theory, built upon the set theory of HOL. It defines the class of complete atomic Boolean lattices with the operators mentioned in Section 2.1 by its algebraic properties. The operators (and predicates) are polymorphically typed as, for example, the binary operator `Meet` with its graphical infix syntax _⊓_

```
Meet  :: (σ::lattice) ⇒ σ ⇒ σ                    (_⊓_)
```

where $\sigma$ is a type variable restricted to the type class `lattice`, which is the set of all types having the lattice operators in common. The operator semantics is given by algebraic and order-theoretic axioms such as `"(x⊓y)⊓z = x⊓(y⊓z)"` and `"(∀x∈A. l ⊑ x) ⟶ l ⊑ Inf(A)"`.

The `Relations` theory is built on top of `Lattice`, introducing the binary type constructor `(α, β)rel` for relations with domain $\alpha$ and codomain $\beta$. For any types $\alpha$ and $\beta$, `(α, β)rel` is declared to be a member of the type class `lattice`, which makes all lattice operators immediately applicable to relations. The typing of the monoidal operators (composition, identity, and transposition) in the following Isabelle declaration is essential for the type-correctness of heterogeneous relation algebras.

```
Comp  :: (α,β)rel ⇒ (β,γ)rel ⇒ (α,γ)rel      (_·_)
I     ::                       (α,α)rel
Conv  :: (α,β)rel            ⇒ (β,α)rel      (_^)
```

The axiomatization of the semantics of these operators is straightforward from the monoidal rules and the Schröder Equivalences mentioned in Section 2.1. Also the *Tarski Rule* `(R ≠ O) = (L·R·L = L)` is postulated, where its formulation as an equivalence additionally implies that the algebra contains at least two elements.

All the following theories contain definitions of constructs that are quite common in applications of relation algebra. Being definitions (using the meta-level equality "≡"), all these further extensions of the RALL language are logically conservative and therefore safe.

In the theory `Special`, special properties of relations are defined as predicates, namely function, order, and vector (i.e. subset) properties such as:

```
total_def       "Total R       ≡ I ⊑ R·R^"
transitive_def  "Transitive R  ≡ R·R ⊑ R"
vector_def      "Vector v      ≡ L·v = v"
```

The theory `Functionals` defines mappings between relations, e.g. residuals (useful for reasoning about weakest preconditions) and symmetric quotients:

```
right_res_def   "R≻S      ≡ −(R^·−S)"
sym_quot_def    "SyQ(R,S) ≡ (R≻S)⊓(S≻R)^"
```

Finally, the theory `Domain` contains relational domain constructions such as products and powersets, which require quantification over relations, like

```
powerset_def "Powerset e ≡ SyQ(e,e) ⊑ I ∧ ∀R. Total(SyQ(R^,e))"
```

As these examples indicate, many useful concepts, even higher-level ones, are formalized in RALL. Further definitions can be easily added if desired.

## 4 Interactive Proofs

This section gives an overview of the facilities that RALL provides for performing interactive proofs and their applications. As it discusses several basic concepts of proofs with Isabelle and relation algebra in general, it also serves as preparation for the following section about automatic proofs.

### 4.1 Overview on the Proof Facilities

In RALL, proofs are normally conducted by backward chaining from the goal, with each step being the application of a resolution tactic. A step may be either

1. a predicate-logical step at the level of formulas (propositions), or
2. an algebraic manipulation of terms: a substitution or an estimation, i.e. an application of rules like `subst = "⟦ s = t; P s ⟧ ⟹ P t"` and `comp_estim2 = "⟦ S' ⊑ S; R ⊑ S'·T ⟧ ⟹ R ⊑ S·T"`.

While the standard Isabelle/HOL tactics excellently cover the former, they provide only quite primitive support for the latter, especially for estimations. Namely, direct selection and manipulation of *sub*terms are difficult in Isabelle (yet normally not so much desired), and explicit order-theoretic reasoning seems not to be too common by now. So we have developed special support.

For a more convenient form of substitution a tactic called `sstac` and some derivations are given. They replace a specific instance of one side of an equation with the corresponding other side.

Estimations with an inclusion formula are performed using the monotonicity of most operators (respectively anti-monotonicity of the complement) and transitivity of the inclusion relation, but this may take some tedious steps. Some tactics like `mtac` do these steps automatically, which involves a search strategy.

The substitution and estimation tactics use higher-order unification to conveniently find a position in the subgoal where to operate on. The outcome of such a tactic may be quite non-deterministic and will therefore often demand backtracking in order to reach the intended application. To avoid this, we provide alternative forms of these tactics that allow the specification of the intended subterm as a restriction of the possible outcomes[3].

The developed tactics are very general and may therefore be used within many other applications of Isabelle involving algebraic and order-theoretic reasoning as well.

[3] In the special case of *simplifying* substitutions, the Simplifier is of course the most suitable tool with deterministic outcome, at least for confluent sets of rules.

Here is a simple example of an interactive proof in RALL using a predicate-logical step, an estimation, and a substitution, just to give an idea of how a proof is actually performed. The lines beginning with “>” contain the user’s input; the remaining lines give the (abbreviated) output of the proof system.

```
> goal Relations.thy "R'=R ⟶ (I⊓S)·R' ⊑ R";
   R'=R ⟶ (I⊓S)·R' ⊑ R
> by(safe_tac HOL_cs);  (* predicate logic *)
   (I⊓S)·R ⊑ R
> by(mtac meet_lb1 1);  (* "x⊓y ⊑ x" *)
   I·R ⊑ R
> by(sstac I_def1 1);   (* "I·R = R" *)
   R ⊑ R
> by(rtac incl_refl 1); (* "x ⊑ x" *)
   No subgoals!
```

This interactive proof on the machine exactly mirrors the steps that would have been done on paper; just imagine backward implications inserted between the output lines.

### 4.2 Application Examples

For all theories mentioned in Section 3.3, a number of basic theorems are proved and made available to the user, for example the order-theoretic properties of lattices, their full distributivity, properties of the complement and of the higher-order operators, and all the standard properties of the relational composition and transposition, like monotonicity and continuity, as contained e.g. in [25]. Even for the more application-specific entities, i.e. special relations, functionals and domain constructions, many of their basic properties are shown.

As a practical case study applied to current research, some sophisticated theorems of Desharnais [7] concerning the *sharpness* of relational products are treated, for example

```
⟦ Q1^·Q2⊓R1·R2^⊑p^·r; Q1⊓Q2·R2·R1^⊑S^·T; Q1·T^·T⊑Q1; S^·S·Q2⊑Q2;
  T^·T·R1⊑R1 ⟧ ⟹ Q1·R1⊓Q2·R2⊑(Q1·p^⊓Q2·r^)·(p·R1⊓r·R2)
```

which are part of an investigation about the suitability of relational products for modeling asynchronous parallel composition. Once established, the proofs of Desharnais’ formula and of many other questions of relation-algebraic research are not difficult by themselves but lengthy and therefore error-prone if done by hand. Here a machine-checked proof gives maximal confidence.

As this section illustrates, the interactive proof system of RALL is a mature tool for the full language of heterogeneous relation algebra. Using its special features for relation-algebraic inferences, proofs can be conducted almost as conveniently as on paper, while the system guarantees the soundness and type-correctness of the obtained results.

## 5 Automatic Proofs

This section describes our automatic proof procedure for relation algebra. We give the idea, the necessary lemmas, an outline of the implementation, and some example proofs.

### 5.1 Atomization

One main observation on Isabelle proofs is that inferences at the (outer) propositional level of formulas can be done very well, even with automatic tactics, whereas more algebraic derivations behave less pleasantly, except for mere simplifications. Also in general, such derivations are quite difficult to perform automatically because of their undirected (and therefore highly non-deterministic and loop-prone) search behavior. But the great challenge of this work was to explore what could nevertheless be done – even with limited effort by now.

Our key idea is to apply an equivalence transformation of the algebraic structures into propositional ones, turning an inclusion into an implication, a meet into a conjunction, etc. In a strongly atomic lattice, this can be achieved by regarding each element as the join of all atoms contained within it. The idea can be generalized to relation-algebraic structures, yielding an isomorphism to a relational structure, which is based on the following formalization of the atom, atomic inclusion, incidence, and trace concepts:

```
atom_def       "Atom a ≡ a≠0 ∧ ∀x. x≠0 ∧ x⊑a ⟶ x=a"
atom_in_def    "a↦x    ≡ Atom a ∧ a⊑x"
incidence_def "c⇝a·b  ≡ c↦a·b ∧ Atom a ∧ Atom b"
trace_def      "i(a)   ≡ a^·a⊓I"
```

The incidences and traces are studied in detail in [8]. It may be tempting to reduce the incidences to equations of atoms like c=a·b, but this is only possible for representable relation algebras. As it is an important requirement for RALL that the obtained results should be valid even for non-representable relation algebras, we must keep the incidences as elementary propositions (of the form c⇝a·b). With the above definitions, the key atomicity theorem can be formulated as strong_atomic = "Sup({a. a↦x}) = x", which is derivable from the axiom weak_atomic = "x≠0 ⟶ ∃a. a↦x" in a distributive lattice.

The *atomization* is an isomorphism to predicate logic, i.e. for each relation-algebraic operator, there is a transformation to a proposition or predicate with a corresponding structure, as listed below. Being equivalences, the transformation rules fully preserve the provability of formulas. The validity of some of these equivalences is non-trivial to prove by the standard relation-algebraic axioms because of the higher-order constructs involved; yet all of this has been done within RALL.

```
atom_in_incl   "x ⊑ y      = ∀a. a↦x ⟶ a↦y"
atom_in_meet   "a↦x⊓y      = a↦x ∧ a↦y"
atom_in_join   "a↦x⊔y      = a↦x ∨ a↦y"
atom_in_cmpl   "a↦−x       = Atom a ∧ ¬a↦x"
```

```
atom_in_inf    "a↦Inf(A)  = Atom a ∧ ∀z∈A. a↦z"
atom_in_sup    "a↦Sup(A)  = ∃z∈A. a↦z"
atom_in_comp   "a↦R·S     = ∃r s. a↝r·s ∧ r↦R ∧ s↦S"
atom_in_conv   "a↦R^      = a^↦R"
atom_in_0      "a↦0       = False"
atom_in_I      "a↦I       = Atom a ∧ a=i(a)"
atom_in_L      "a↦L       = Atom a"
```

We use the equivalences to atomize relation-algebraic formulas in the first step of automatic proofs, as outlined in the following subsection. The result of the atomization is relational atom structure as defined in Section 2.2, i.e. predicate-logical formulas with predicates of the form Atom X, X↦R, X↝Y·Z, or X=Y, where X, Y, and Z are of the form a, a^, i(a), and i(a^) for some atom a.

### 5.2 The Automatic Proof Procedure

Our automatic proof procedure works by first atomizing the goal and then performing mainly predicate-logical steps with depth-first search[4] like the Classical Reasoner. Together with the lemma atom_in_is_atom = "a↦x ⟹ Atom a", this is already sufficient for proofs that do not take into account the laws for relational composition and transposition, i.e. purely lattice-theoretic proofs.

For general relation-algebraic proofs, the atomic analogues of the monoidal axioms and Schröder Equivalences are necessary, as found in [16, 8]. Though proved in RALL from their non-atomic counterparts, they can be considered as axioms on the atomic level. There should also be a pre-Tarski rule, which has not yet been implemented, and is rarely needed anyway. For the pre-associativity, pre-identity, and pre-Schröder rules, there are several variants differing only in the position and transposition state of the incidence arguments. A typical representative of each rule type is given below.

```
assoc_1_0    = "⟦ a↝f·z; f↝x·y ⟧ ⟹  ∃g. a↝x·g ∧ g↝y·z"
ident_I2     = "⟦ i(j) = i(k); Atom k; Atom j ⟧ ⟹ k↝k·i(j)"
ident_D2     = "⟦ q↝k·i(j); Atom j ⟧ ⟹ q=k ∧ i(j)=i(k) ∧ Atom k"
schroeder_2 = "q↝r·s ⟹ r↝q·s^"
```

Special tactics are necessary for the application of these rules as most of them have counterparts with the reverse effect, leading to the risk of loops within the proof search. Besides this, the treatment of transposition in conjunction with these rules is non-trivial: In order to match a transposed schematic variable ?x^ with a constant a (yielding the substitution a^ for ?x), unification modulo involution of transposition is needed, but not supported by Isabelle itself. Furthermore, the equalities introduced or exploited by the identity properties demand extra handling. So we had to extend the Classical Reasoner to integrate the Simplifier and our special tactics. Meanwhile, general mechanisms for doing such extensions have found their way into the standard version of the Classical Reasoner.

[4] In special cases, a more expensive but less dangerous best-first search or iterative deepening may be used instead.

This is the overall structure of the automatic proof procedure.

1. Preparation:
   (a) unfolding of definitions
   (b) simplification (with rules like "−−x = x", "0·R = 0")
   (c) atomization, as described in Section 5.1
2. Main Part: depth-first search with
   (a) simplifications with rules like "a^^ = a" and i(i(a)) = i(a)"
   (b) usual predicate-logical steps of Classical Reasoner
   (c) application of atom lemmas like "q⇝r·s ⟹ Atom s", "Atom j ⟹ i(j)↦I", and the pre-identity rules (given above)
   (d) special tactics for
      i. generation of all association variants of incidence pairs within the premises, applying the pre-associativity rules
      ii. if there is a premise a↦I : rewriting of all atoms a to i(a), making the pre-identity rules applicable
      iii. solution of subgoals using the pre-Schröder rules

Taking the essence of the above and abstracting over auxiliary features like the treatment of transposition and of equality of atoms, the automatic proof procedure handles atomized goals as follows. Goals of the form[5] Atom X are solved by assumption from the premises or by applying rules like atom_in_is_atom. Goals of the form X↦R can only be solved by assumption. A goal of the form X⇝Y·Z may be solved by using the first kind of pre-identity rules or by assumption or application of some pre-Schröder rule (perhaps after generating additional incidence premises with the pre-associativity rules). The rest is done by a search for pure predicate-logical proofs.

### 5.3 Automatic Proofs

The automatic proof procedure (called fpg here) can be used quite easily. For example, the Dedekind rule can be proved by

```
> val dedekind = fpg "Q·R⊓S ⊑ (Q⊓S·R^)·(R⊓Q^·S)";
```

Within a few seconds (on a Sun SPARCstation 10), successful result is given. This proof is carried out by the special tactics in a straightforward way (without backtracking). Compare this with the standard proof of the theorem, which takes some tricky unfolding steps. Of course, more application-specific proofs can also be performed automatically, e.g.

```
> val sym_quot_red_lemma = fpg "SyQ(Q,R)·SyQ(R,S) ⊑ SyQ(Q,S)";
```

This proof takes more than one minute, but by further tuning the implementation of the proof procedure this time may be significantly reduced.

[5] These goals are rather redundant and could even be dispensed with by a more elaborated kind of atomization, e.g. by consigning the handling of the atom property to the type system.

### 5.4 Soundness and Completeness of the Proof Procedure

As for every proof system, soundness and completeness are important criteria for the judgment of the automatic proof procedure.

Our proof procedure can be considered sound for the following reasons. The meta logic of Isabelle is a small well-understood system, and the axiomatizations of both HOL and RALL are straightforward from standard definitions. For the rest of the system, the LCF-system approach of correctness by construction (i.e. guarded application of trusted components) gives the necessary confidence.

As it is often the case, the question of completeness must be answered negatively. In the first place, this is due to the general non-decidability of the underlying predicate calculus. For example, universal quantifications in the assumptions of a subgoal are instantiated only once at most.

The isomorphism for atomization affects all kinds of operators and there are no further theorems or tactics needed to perform the lattice-theoretic fraction of proofs. So in this area the procedure should be as powerful as the standard Classical Reasoner, which is quite satisfactory.

For actual relation-algebraic proofs, the procedure is usable, but far less exhaustive. The Tarski rule is missing completely, and the choice of additional rules and special tactics is more or less heuristic. A further problem is that attempts to prove large theorems often cause non-termination or produce a memory overflow after some time. This could be fixed by preventing possibilities for loops that may still exist, or by mere optimizations of the proof procedure. At least for small search spaces a solution is found in most cases.

## 6 Conclusion

We have presented a theorem proving system for relation algebra based on the generic theorem prover Isabelle. RALL makes the full language of heterogeneous relation algebra available including type-correct deduction. There is an almost one-to-one correspondence between proofs conducted by hand and interactive proofs with RALL, due to the help of some semi-automatic tactics. Thus no new style of performing proofs has to be adopted, while gaining the reliability of sound and type-correct machine-controlled inference steps. The system is very flexible and open for application-specific extensions.

RALL even offers an automatic proof facility. As developed by the first author in a fixed-time project [19], the automatic proof procedure has been left in an experimental stage. At the time of this writing we are unable to give a final statement about its general power, but we can state that at least for small theorems it gives respectable results. It seems quite promising to go further in the direction described in this paper.

Concerning automatic theorem proving in relation algebra, attempts to use the set of axioms of relation algebra directly at the relational level still fail. A probably more successful direction would be the component-oriented view, taking the relations as binary predicates and applying the tactics suited for the level of first-order predicate logic. However, the component-oriented view is not the one aimed at when working with relation algebra, since Tarski originally designed relation algebra in order to establish a set theory without point variables. Furthermore, current researchers adopt the view of applying relation algebra in the component-free manner for the purpose of an elegant and highly precise formal system.

Our work answers the demand for automatic proof facilities as follows. For every relation algebra, automatic theorem proving can be established by the atomic level view as there is always an embedding in a suitable complete atomic relation algebra that is subject to the described atomization technique. In the atomic level view, relations are merely designed as unary predicates on atoms of a relation algebra and use the relation-algebraic axioms as formulated for the atomic level. By the given transformation, we have obtained an isomorphism to predicate logic such that the abstract component-free style is treated with success analogously to the (inappropriate) component-oriented view. Nevertheless, whether algebraic logics that exclude atomization can be dealt with appropriately in order to obtain an automatic theorem proving system, is still a question of ongoing research.

For heterogeneous relation algebra there is another system called *RALF* that provides interactive theorem proving of relation-algebraic formulas [2]. RALF stresses a highly developed graphical interface and administration of theorems and proofs in progress. As the RALF system is a stand-alone product and has no connection to a generic theorem prover, it is very costly for developers (and even impossible for users) to extend the underlying logic and proof facilities. RALF does not support automatic theorem proving either.

There is also another relational logic formalized in Isabelle, namely the relational circuit description language *Ruby* [23]. Some of its circuit combinators are reminiscent of relation-algebraic operators. This implementation is based on the ZF set theory, and therefore the type-correctness of terms is not ensured automatically but left to the user as an additional proof obligation. Ruby has adopted the component-oriented view with relations as binary predicates, which makes it non-applicable for abstract relation algebras in general.

Recent research has attempted to combine fuzzy set theory with abstract relation algebra [14, 20]. Once axiomatized, the obtained fuzzy relation algebra can be used to represent informedness in data base semantics and is recommended to have a corresponding theorem proving system present. It seems to be a promising future research to extend the flexible RALL system in order to deal with fuzzy relation algebra.

Being an advanced application of Isabelle, RALL has brought up insights into the Isabelle system like the feasibility of substitution and monotonicity inferences, the limits of its unification procedure and its type system (not mentioned further in this paper), and the combination of search procedures with simplification and special-purpose tactics. Altogether, Isabelle has shown to be a very powerful and flexible theorem proving tool suitable for our needs.

RALL is available in the same manner as the Isabelle system itself, viz. it can be obtained on the World Wide Web from the URL address

`http://www.cl.cam.ac.uk/Research/HVG/Isabelle/projects.html`

which contains several contributions of logics and theorem proving systems implemented in Isabelle.

## Acknowledgments

We would like to thank Tobias Nipkow, Paul James, Wolfgang Naraschewski, Beate Hahn, Markus Wenzel, and some anonymous referees for their comments on draft versions of this paper. Furthermore, we have benefited from discussions with Lamia Labed-Jilani, one of Ali Mili's students, who is undertaking a project on relation-algebraic proofs using the OTTER system. Eventually, the second author also wishes to thank Armando M. Haeberer and the Fundação de Apoio a Pesouisa do Estado do Rio de Janeiro (FAPERJ) for the pleasant PARATÍ 95 meeting including the opportunity to give a talk on the subject of this paper.

## References

1. Backhouse, R.C., Hoogendijk, P., Voermans, E., van der Woude, J.C.S.P: A relational theory of datatypes. Eindhoven University of Technology, Dept. of Mathematics and Computer Science (1992)
2. Berghammer, R., Hattensperger, C., Schmidt, G.: RALF – A relation-algebraic formula manipulation system and proof checker. In: Nivat, M., Rattray, C., Rus, T., Scollo, G. (Eds.): Proc. 3rd Conference on Algebraic Methodology and Software Technology – AMAST '93. *Series:* Workshops in Computing, Springer-Verlag (1994) 407–408
3. Berghammer, R., Schmidt, G.: Relational specifications. In: Rauszer C. (ed.): Algebraic Methods in Logic and in Computer Science. *Series:* Banach Center Publications 28, Polish Academy of Sciences (1993) 167–190
4. Birkhoff, G.: Lattice Theory. AMS Colloquium Publications 25 ($^3$1967)
5. Brink, C., Schmidt, G.: Relational methods in computer science. Schloß Dagstuhl, Seminar Nr. 9403, Technischer Bericht Nr. 80 (1994)
6. Church, A.: A Formulation of the Simple Theory of Types. In: Journal of Symbolic Logic (1940) 56–
7. Desharnais, J., Baltagi, S., Chaib-draa, B.: Simple weak sufficient conditions for sharpness. Université Laval, Quebec, Research Report DIUL-RR-9404 (1994)
8. Gritzner, T.F.: Die Axiomatik abstrakter Relationenalgebren: Darstellung der Grundlagen und Anwendung auf das Unschärfeproblem relationaler Produkte. Technische Universität München, Diploma Thesis, *also available as report:* TUM-INFO-04-91 (1991)
9. Gritzner, T.F.: wp-Kalkül und relationale Spezifikation kommunizierender Systeme. Technische Universität München, Doctoral Dissertation (1995)
   Herbert Utz Verlag Wissenschaft, ISBN 3-931327-64-7 (1996)
10. Haeberer, A.M. (Ed.): Relational methods in computer science. Proceedings of PARATI '95. In preparation.

11. Henkin, L., Monk, J.D., Tarski, A., Cylindric Algebras, Parts I & II, *Series:* Studies in Logic and the Foundations of Mathematics 64 (1971) & 115 (1985), North-Holland Publ.Co.
12. Hoare, C.A.R., He Jifeng: The weakest prespecification, parts I&II. In: Fundamenta Informaticae IX (1986) 51–84 & 217–252
13. Jónsson, B., Tarski, A.: Boolean algebras with operators, parts I&II. In: American Journal of Mathematics 73 (1951) 891–939 & 74 (1952) 127–167
14. Kawahara, Y., Furusawa, H.: An algebraic formalization of fuzzy relations. Draft paper (April 19, 1995)
15. Lyndon, R.C., *The representation of relation(al) algebras, parts I&II*, *in:* Annals of Mathematics (II) 51 (1950) 707–729 & 63 (1956) 294–307, Princeton University Press
16. Maddux, R.: Finite integral relation algebras. In: Comer, S.D. (ed.), Universal Algebra and Lattice Theory, Lecture Notes in Mathematics 1149 (1985) 175–197, Springer-Verlag
17. Nipkow, T.: Term Rewriting and Beyond — Theorem Proving in Isabelle. In: Formal Aspects of Computing 1 (1989) 320–338
18. Nipkow, T.: Order-Sorted Polymorphism in Isabelle. In: Huet, G., Plotkin, G. (eds.): Logical Environments. Cambridge University Press (1993) 164–188
19. von Oheimb, D.: Zur Konstruktion eines auf Isabelle gestützten Beweissystems für die Relationenalgebra. Technische Universität München, Praktische Semesterarbeit (1995)
20. Ounalli, H., Jaoua, A.: On fuzzy difunctional relations. Draft paper presented at 2nd RelMiCS – PARATI '95.
21. Paulson, L.C.: ML for the Working Programmer. Cambridge University Press (1991)
22. Paulson, L.C.: Isabelle - A Generic Theorem Prover. Lecture Notes in Computer Science 828 (1994)
23. Rasmussen, O.: Formalizing Ruby in Isabelle ZF. In: Paulson, L.C. (Ed.): Proceedings of the First Isabelle Users Workshop. University of Cambridge (1995) 246–265
24. de Roever, W.P.: Recursive Program Schemes: Semantics and Proof Theory. Vrije Universiteit te Amsterdam, Doctoral Dissertation (1974)
25. Schmidt, G., Ströhlein, Th.: Relations and Graphs. *Series:* EATCS Monographs in Computer Science, Springer-Verlag (1993)
26. Tarski, A.: On the calculus of relations. In: Journal of Symbolic Logic 6 (1941) 73–89
27. Tarski, A., Givant, S.: A Formalization of Set Theory Without Variables. AMS Colloquium Publications 41 (1987)
28. Zierer, H.: Programmierung mit Funktionsobjekten: Konstruktive Erzeugung semantischer Bereiche und Anwendung auf die partielle Auswertung. Technische Universität München, Doctoral Dissertation, *also available as:* TUM-I8803 (1988)

This article was processed using the LaTeX macro package with LLNCS style