System Description: ARA – An Automatic Theorem Prover for Relation Algebras*

Carsten Sinz

Symbolic Computation Group, WSI for Computer Science,
Universität Tübingen, D-72076 Tübingen, Germany
sinz@informatik.uni-tuebingen.de

Abstract. ARA is an automatic theorem prover for various kinds of relation algebras. It is based on Gordeev's Reduction Predicate Calculi for n-variable logic (RPC_n) which allow first-order finite variable proofs. Employing results from Tarski/Givant and Maddux we can prove validity in the theories of simple semi-associative relation algebras, relation algebras and representable relation algebras using the calculi RPC_3, RPC_4 and RPC_ω. ARA, our implementation in Haskell, offers different reduction strategies for RPC_n, and a set of simplifications preserving n-variable provability.

1 Introduction

Relations are an indispensable ingredient in many areas of computer science, such as graph theory, relational databases, logic programming, and semantics of computer programs, to name just a few. So relation algebras – which are extensions of Boolean algebras – form the basis of many theoretical investigations. As they can also be approached from a logic point of view, an application of ATP methods promises to be beneficial.

We follow the lines of Tarski [TG87], Maddux [Mad83] and Gordeev [Gor95] by converting equations from various relation algebraic theories to finite variable first-order logic sentences. We are then able to apply Gordeev's n-variable calculi RPC_n to the transformed formulae.

Our implementation ARA is a prover for the RPC_n calculi with a front-end to convert relation algebraic propositions to 3-variable first-order sentences. It implements a fully automatic proof procedure, various reduction strategies, and some simplification rules, to prove theorems in the theories SSA[1], RA, and RRA.

2 Theoretical Foundations

Gordeev's Reduction Predicate Calculi. Gordeev developed a cut free formalization of predicate logic without equality using only finitely many distinct

* This work was partially supported by DFG under grant Ku 966/4-1.

[1] This "simple" variant of semi-associative relation algebra (SA) includes the identity axiom $A \odot \mathring{\mathbf{1}} = A$ only for literals A.

variables. In [Gor95] the reduction predicate calculi RPC_n were introduced. These term rewriting systems reduce valid formulae of n-variable logic to true. n-variable logic comprises only formulae containing no more than n distinct variables, the same restriction applies to RPC_n. Moreover, all RPC_n formulae are supposed to be in negation normal form.

Formulae provable in RPC_n are exactly those provable in the standard modus ponens calculus, or, alternatively, the sequent calculus with cut rule, using at most n distinct variables ([Gor95], Theorem 3.1 and Corollary). The RPC_n rewriting systems consist of the following rules:[2]

$$
\begin{array}{ll}
(R1) & A \vee \top \longrightarrow \top \\
(R2) & L \vee \neg L \longrightarrow \top \\
(R3) & A \wedge \top \longrightarrow A \\
(R4) & A \vee (B \wedge C) \longrightarrow (A \vee B) \wedge (A \vee C) \\
(R5) & \exists x A \longrightarrow \exists x A \vee A[x/t] \\
(R6) & \forall x A \vee B \longrightarrow \forall x A \vee B \vee \forall y(\forall y B \vee A[x{-}y][x/y]) \\
(R6') & \forall x A \longrightarrow \forall x A \vee A[{-}x] \vee \forall y(A[x{-}y][x/y])
\end{array}
$$

Here, L denotes an arbitrary literal, A, B and C are formulae, x and y are individual variables, and t is any term. A term may be a variable or a constant, function symbols do not occur in RPC_n. $F[x/t]$ denotes substitution of x by t, where $F[x/t]$ does not introduce new variables. $F[{-}x]$ and $F[x{-}y]$ are variable elimination operators (see [Gor95] for a rigorous definition), where $F[{-}x]$ deletes all free occurrences of x from F by replacing the respective (positive and negative) literals contained in F by falsum ($\bot$). The binary elimination operator $F[x{-}y]$ is defined by $F[x{-}y] = F[{-}y]$ if $x \neq y$, and $F[x{-}y] = F$ otherwise. Note, that the variable elimination operators are – just like substitution – meta-operators on formulae, and not part of the formal language of the logic itself.

A formula F is provable in RPC_n ($\mathrm{RPC}_n \vdash F$) iff it can be reduced to true, i.e., iff $F \xrightarrow{*}_{\mathrm{RPC}} \top$. We call a formula sequence $(F_0, \ldots, F_n)$ a reduction chain if $F_i \longrightarrow_{\mathrm{RPC}} F_{i+1}$. A reduction strategy is a computable function that extends a reduction chain by one additional formula. As all essential rules of RPC_n are of the form $F \longrightarrow F \vee G$, and thus no "wrong" reductions are possible, strategies can be used instead of a search procedure. This also means that no backtracking is needed and the RPC_n calculi are confluent on the set of valid formulae of n-variable logic.

ARA implements various reduction strategies rather than unrestricted breadth-first search or iterative deepening.

Translation from Relation Algebra to Predicate Logic. In order to prove formulae in the theory of relation algebra, we follow the idea of [TG87] and transform sentences of relation algebra to 3-variable first-order sentences. The transformation τ_{xyz} is straightforward, where x and y denote the predicate's arguments and z is a free variable. For brevity, we give only part of the definition

[2] $\vee$ and $\wedge$ are supposed to be AC-operators.

of τ_{xyz}, for predicate symbols, relative and absolute product ($\odot$ and $\cdot$):

$$\begin{aligned}\tau_{xyz}(R) &= xRy\\ \tau_{xyz}(\Phi \odot \Psi) &= \exists z(\tau_{xzy}(\Phi) \wedge \tau_{zyx}(\Psi))\\ \tau_{xyz}(\Phi \cdot \Psi) &= \tau_{xyz}(\Phi) \wedge \tau_{xyz}(\Psi)\end{aligned}$$

Here, Φ and Ψ stand for arbitrary relation algebraic expressions. A sentence $\Phi = \Psi$ from relation algebra is then translated by τ into an equivalence expression of first-order logic, a relational inclusion $\Phi \leq \Psi$ into an implication:

$$\begin{aligned}\tau(\Phi = \Psi) &= \forall x \forall y(\tau_{xyz}(\Phi) \leftrightarrow \tau_{xyz}(\Psi))\\ \tau(\Phi \leq \Psi) &= \forall x \forall y(\tau_{xyz}(\Phi) \rightarrow \tau_{xyz}(\Psi))\end{aligned}$$

We can simulate proofs in various theories of relation algebra by using the following tight link between n-variable logic and relation algebras proved by Maddux (see, e.g., [Mad83]):

1. A sentence is valid in every *semi-associative relation algebra (SA)* iff its translation can be proved in 3-variable logic.
2. A sentence is valid in every *relation algebra (RA)* iff its translation can be proved in 4-variable logic.
3. A sentence is valid in every *representable relation algebra (RRA)* iff its translation can be proved in ω-variable logic.

We have to restrict SA in case of 3-variable logic to its simple variant SSA, as RPC_n – as well as the more familiar Hilbert-Bernays first-order formalism – contains only the simple Leibniz law. Compared to the generalized Leibniz law used in Tarski's and Maddux's formalisms, this simple schema is finitely (i.e., as an axiom) representable in RPC_n. The corresponding refinement is not necessary in case of RA and RRA, as the generalized Leibniz law for 3-variable formulae is deducible from its simple form in RPC_4, and thus in RPC_ω (see, e.g., [Gor99]).

3 Implementation

The ARA prover is a Haskell implementation of the RPC_n calculi with a front end to transform relation algebraic formulae to first-order logic. It offers different reduction strategies and a set of additional simplification rules. Most of these simplification rules preserve n-variable provability and can thus be used for SSA- and RA-proofs.

Input Language. The ARA system is capable of proving theorems of the form $\{E_1, \ldots E_n\} \vdash E$, where E and E_i are relation algebraic equations or inclusions.[3] Each equation in turn may use the connectives for relative and absolute sum

[3] Instead of an equation E a formula F may also be used, which is interpreted as the equation $F = 1$, where 1 stands for the absolute unit predicate.

and product, negation, conversion, and arbitrary predicates, among them the absolute and relative unit and the absolute zero as predefined predicates.

The translation of a relational conjecture of the form $\{E_1, \ldots E_n\} \vdash E$ to first-order logic is done in accordance with Tarski's deduction theorem for $\mathcal{L}^\times$ ([TG87], 3.3):

$$\tau(\{E_1, \ldots E_n\} \vdash E) = \tau(E_1) \wedge \ldots \wedge \tau(E_n) \rightarrow \tau(E)$$

To give an impression of how the actual input looks like, we show the representation of Dedekind's rule $(Q \odot R) \cdot S \leq (Q \cdot (S \odot R^\smile)) \odot (R \cdot (Q^\smile \odot S))$ as a conjecture for ARA:

```
|- (Q@R)*S < (Q*(S@R^))@(R*(Q^@S));
```

Literal and Reduction Tracking. To guarantee completeness of the deterministic proof search introduced by reduction strategies, we employ the technique of reduction tracking in our implementation. The idea is as follows: While successively constructing the reduction chain, record the first appearance of each reduction possibility[4] and track the changes performed on it. A strategy is complete, if each reduction possibility is eventually considered.

Literal tracking is used by the LP reduction strategy described later. It keeps track of the positions of certain literal occurrences during part of the proof.

Reduction Strategies. We implemented a trivial strategy based on the reduction tracking idea described above and several variants of a literal tracking strategy. The latter select a pair of complementary literals (and therefore are called LP strategies) that can be disposed of by a series of reduction steps. In order to find such pairs, an equation system is set up (similar to unification) that is solvable iff there is a reduction sequence that moves the literals (or one of their descendants) into a common disjunction. Then, RPC reductions are selected according to the equation system to make the literal pair vanish.

Additional Simplification Rules. To improve proof search behavior we added some simple rules and strategies to the RPC_n calculus that preserve n-provability:

1. Give priority to shortening rules $(R1)$, $(R2)$ and $(R3)$.
2. Remove quantifiers that bind no variables.
3. Minimize quantifier scopes.
4. Subgoal generation: To prove $F \wedge G$ prove first F, and then G.
5. Additional $\forall$-rule: $\forall x A \vee B \longrightarrow \forall y(A[x/y] \vee B)$ if $y \notin \mathrm{Fr}(A) \cup \mathrm{Fr}(B)$, which is used with priority over $(R6)$ and $(R6')$.
6. Delete pure literals.
7. Replace $F \vee \tilde{F}$ resp. $F \wedge \tilde{F}$ by F, if $\tilde{F}$ is a bound renaming of F.

[4] A reduction possibility consists of a position in the formula and, in case of the RPC-rules $(R5)$, $(R6)$ and $(R6')$, an additional reduction term.

Simplification rule 3 is applied only initially, before the actual proof search starts. Rules 5 and 7 are used to keep formula sizes smaller during proof search, where rule 5 is a special form of ($R6$) that has the purpose to accelerate introduction of so far unused variables.

Moreover, there are two additional simplification rules that, however, may change n-provability: (1) partial Skolemization to remove $\forall$-quantifiers and (2) replacement of free variables by new constants.

4 Experimental Results

We made some experiments with our implementation on a Sun Enterprise 450 Server running at 400 MHz. The Glasgow Haskell Compiler, version 4.04, was used to translate our source files. In Table 1 the results of our tests are summarized. The problem class directly corresponds to the number of variables used for the proof, as indicated at the end of Section 2.

Table 1. ARA run times for some relation algebra problems.

problem	source	class	strat.	proofs	steps	time
3.2(v)	[TG87]	SSA	LI	2	46	130
3.2(vi)	[TG87]	SSA	LI	2	41	100
3.2(xvii)	[TG87]	SSA	LI	3	22	40
3.1(iii)(ϵ)	[TG87]	RA	AI	6	104	200
3.2(xix)	[TG87]	RA	LA	3	25	50
Thm 2.7	[CT51]	RA	AI	1	12	40
Thm 2.11	[CT51]	RA	AI	1	19	50
Cor 2.19	[CT51]	RA	AI	1	77	75170
Dedekind	[DG98]	RA	AI	1	37	90
Cor 2.19	[CT51]	RRA	AI	1	38	140

In the last three columns the following information is given: the number of proofs that the problem consists of, the total number of RPC-reductions[5], and the total proof time in milliseconds. The first letter in the strategy column is "L" for the normal LP reduction strategy and "A" for the LP strategy with priority for the additional $\forall$-simplification rule. The second letter corresponds to the selection of disjunctive subformulae, i.e., A in rule ($R4$) and B in rule ($R6$). The strategy indicated by letter "I" selects a minimal suitable disjunction, "A" a maximal one.

The proof of Corollary 2.19 from [CT51] reveals an unexpectedly long runtime, which is reduced considerably by allowing more variables for the proof and thus switching to RRA.

[5] Only reductions with one of the rules ($R4$), ($R5$), ($R6$) and ($R6'$) are considered.

5 Conclusion and Future Work

By using ARA, many small and medium-sized theorems in various relation algebras could be proved. Formulae not containing the relative unit predicate can be handled quite efficiently, other formulae suffer from the fact that neither the RPC_n calculi nor the ARA prover offer a special treatment of equality.

Compared with other implementations [HBS94, vOG97, DG98], the most obvious differences are the automatic proof procedure using reduction strategies and the translation to the RPC_n calculi. The RALF system [HBS94] offers no automatic proof search, but has particular strengths in proof presentation. RALL [vOG97] is based on HOL and Isabelle, and thus is able to deal with higher order constructs. It also offers an experimental automatic mode using Isabelle's tactics. δ**RA** is a Display Logic calculus for relation algebra, and its implementation [DG98] is based on Isabelle's metalogic. It also offers an automatic mode using Isabelle's tactics.

ARA can also be used to generate proofs in ordinary first-order logic and in restricted variable logics. As ARA is the initial implementation of a new calculus, we expect that further progress is very well possible. Implementation of new strategies or built-in equality may be viable directions for improvement.

Availability. The ARA system is available as source and binary distribution from `www-sr.informatik.uni-tuebingen.de/~sinz/ARA`.

References

[CT51] L. H. Chin and A. Tarski. Distributive and modular laws in the arithmetic of relation algebras. *University of California Publications in Mathematics, New Series*, 1(9):341–384, 1951.

[DG98] J. Dawson and R. Goré. A mechanized proof system for relation algebra using display logic. In *JELIA'98*, LNAI 1489, pages 264–278. Springer, 1998.

[Gor95] L. Gordeev. Cut free formalization of logic with finitely many variables, part I. In *CSL'94*, LNCS 933, pages 136–150. Springer, 1995.

[Gor99] L. Gordeev. Variable compactness in 1-order logic. *Logic Journal of the IGPL*, 7(3):327–357, 1999.

[HBS94] C. Hattensperger, R. Berghammer, and G. Schmidt. RALF - a relation-algebraic formula manipulation system and proof checker. In *AMAST'93*, Workshops in Computing, pages 405–406. Springer, 1994.

[Mad83] R. Maddux. A sequent calculus for relation algebras. *Annals of Pure and Applied Logic*, 25:73–101, 1983.

[TG87] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, 1987.

[vOG97] D. von Oheimb and T. Gritzner. RALL: Machine-supported proofs for relation algebra. In *Automated Deduction - CADE-14*, LNAI 1249, pages 380–394. Springer, 1997.