A Mechanised Proof System for Relation Algebra using Display Logic

Jeremy E. Dawson * and Rajeev Goré **

Department of Computer Science and Automated Reasoning Project,
Australian National University, Canberra ACT 0200, Australia
jeremy@arp.anu.edu.au, rpg@arp.anu.edu.au

Abstract. We describe an implementation of the Display Logic calculus for relation algebra as an Isabelle theory. Our implementation is the first mechanisation of any display calculus. The inference rules of Display Logic are coded directly as Isabelle theorems, thereby guaranteeing the correctness of all derivations. Our implementation generalises easily to handle other display calculi. It also provides a useful interactive proof assistant for relation algebras.
We describe various tactics and derived rules developed for simplifying proof search, including an automatic cut-elimination procedure, and example theorems proved using Isabelle. We show how some relation algebraic theorems proved using our system can be put in the form of structural rules of Display Logic, facilitating later re-use. We then show how the implementation can be used to prove results comparing alternative formalizations of relation algebra from a proof-theoretic perspective.

Keywords: proof systems for relation algebra, non-classical logics, automated deduction, display logic, description logics

1 Introduction

Relation algebras are extensions of Boolean algebras; whereas Boolean algebras model subsets of a given set, relation algebras model binary relations on a given set. Thus relation algebras have relational operations such as composition and converse. As each relation is itself a set (of pairs), relation algebras also have the Boolean operations such as intersection (conjunction) and complement (negation). Relation algebras form the basis of relational databases and of the specification and proof of correctness of programs. Recently, relation algebras and their extensions, Peirce algebras, have also been shown to from the basis of description logics [5]. Just as Boolean algebras can be studied as a logical system (classical propositional logic), relation algebras can also be studied in a logical, rather than an algebraic, fashion. In particular, relation algebras can be formulated using Display Logic [12], and in several other ways [16].

Display Logic [1] is a syntactic proof system for non-classical logic, based on the Gentzen sequent calculus [9]. Its advantages include a generic cut-elimination

* Supported by the Defence Science and Technology Organization
** Supported by an Australian Research Council QEII Fellowship

theorem, which applies whenever the rules for the display calculus satisfy certain conditions. It is a significant general logical formalism, applicable to many logics; its mechanisation is therefore an important challenge for automated reasoning.

In this paper we describe an implementation of $\boldsymbol{\delta}$**RA** [12], a display logic formulation of relation algebras, using the Isabelle theorem prover. In this implementation the rules of $\boldsymbol{\delta}$**RA** form the axioms of an Isabelle object logic – $\boldsymbol{\delta}$**RA** is not built upon one of the standard Isabelle object logics (as is the case, for example, with RALL [17]). The ease with which this display calculus can be implemented in Isabelle highlights the generality of both Isabelle and Display Logic. We demonstrate how Isabelle can be used to show the relationship between $\boldsymbol{\delta}$**RA** and two other formalizations of relation algebras.

1.1 Other mechanised proof systems for relation algebras

RALL [17] is a theorem proving system for relation algebra, based on Isabelle. It uses the atomicity of relation algebras; every relation algebra can be embedded in an atomic relation algebra. Although RALL provides automated proof search, this feature is heavily dependent on the atomization of the relation algebra. It is still not clear to us to what extent RALL is applicable to relation algebras which are not themselves atomic. RALL is built upon the `HOL` Isabelle theory, whereas our implementation of $\boldsymbol{\delta}$**RA** is built directly upon Isabelle's metalogic.

RALF [2] is a graphically oriented relation algebra formula manipulation system and proof assistant. It contains a large number of hard-coded transformation rules, and the super-user can add others. However the rules do not form a formal calculus. Thus it does not demonstrate that the results it derives follow from any formalization of relation algebra. In fact it may be seen as complementing a system such as that described in this paper since an interesting avenue for further work would be to obtain the transformation rules of RALF as $\boldsymbol{\delta}$**RA** derived rules, giving a rigrorous basis to RALF and a graphical front-end to $\boldsymbol{\delta}$**RA**.

2 Relation Algebras

A **relation** on a set U is a set of ordered pairs (a, b) of elements of U, ie a subset of $U \times U$. Since a relation is itself a set (of ordered pairs), the operations on relations include the operations of Boolean algebras. Other relational operations are (as defined in [12]) **composition** $R \circ S = \{(a, b) \mid \exists c.\ (a, c) \in R \text{ and } (c, b) \in S\}$, its **identity** $\{(a, a) \mid a \in U\}$, and **converse** $\smile R = \{(a, b) \mid (b, a) \in R\}$.

Relation algebras are an abstraction of the notion of binary relations on a set U; reference to the actual elements of the set is abstracted away, leaving the relations themselves as the objects under consideration. Chin & Tarski [6] give a finite equational axiomatization of relation algebras. The following definition, equivalent to that in [6], is taken from [5] and [12].

A relation-type algebra $\mathcal{R} = \langle R, \vee, \wedge, \neg, \bot, \top, \circ, \smile, \mathbf{1} \rangle$ (ie, a free algebra on the set of variables R with binary operators $\vee$, $\wedge$ and $\circ$, unary operators $\neg$ and $\smile$, and constants $\bot$, $\top$ and $\mathbf{1}$) is a **relation algebra** if it satisfies the following axioms for each $r, s, t \in R$

(RA1) $(R, \vee, \wedge, \neg, \bot, \top)$ is a Boolean algebra

(RA2) $r \circ (s \circ t) = (r \circ s) \circ t$ (RA3) $r \circ \mathbf{1} = r = \mathbf{1} \circ r$

(RA4) $\smile\smile r = r$ (RA5) $(r \vee s) \circ t = (r \circ t) \vee (s \circ t)$

(RA6) $\smile (r \vee s) = (\smile r) \vee (\smile s)$ (RA7) $\smile (r \circ s) = (\smile s) \circ (\smile r)$

(RA8) $\smile r \circ \neg(r \circ s) \leq \neg s$

Further operators can be defined: $A \leq B$, as used in (RA8), means $(A \wedge B) = A$. Duals of $\mathbf{1}$ and $\circ$ are defined: $\mathbf{0}$ is $\neg\mathbf{1}$, and $A + B$ is $\neg(\neg A \circ \neg B)$. We call this theory of relation algebras **RA**. A *proper* relation algebra consists of a set of relations on a set; note that not all relation algebras are isomorphic to a proper relation algebra ([16], p.444-5).

3 Relation Algebras in Display Logic

A number of different logical systems can be formulated using the method, or style, of Display Logic [1]. These include several normal modal logics [21], and intuitionistic logic [11]. Display Logic resembles the Gentzen sequent calculus **LK**, but with significant differences. For example, the rules for introducing the connective '$\vee$' (on the right) in **LK** and Display Logic are

$$\frac{\Gamma \vdash \Delta, P, Q}{\Gamma \vdash \Delta, P \vee Q}(\mathbf{LK}\text{-} \vdash \vee) \qquad \text{and} \qquad \frac{X \vdash P, Q}{X \vdash P \vee Q}(\mathrm{DL}\text{-} \vdash \vee)$$

Whereas, in **LK**, Γ and Δ denote comma-separated lists of formulae, in Display Logic, X denotes a Display Logic structure, which can involve several *structural* operators (one of which is ','). (See [12] or [7] for a full explanation of *structures* and *formulae*). In Display Logic, unlike in **LK**, the introduced formula (here, $P \vee Q$) stands by itself on one side of the turnstile. However there are also rules (the "display postulates")which effectively allow moving formulae from one side to the other. In [12] there is a Display Logic formulation of relation algebras, called $\boldsymbol{\delta}$**RA** whose structural operators are ',' ';' '$*$' '$\bullet$' 'I' 'E' . As in **LK**, the use of ',' to stand for either '$\wedge$' or '$\vee$' (depending on the position) reflects the duality between them. Likewise in $\boldsymbol{\delta}$**RA** ';' stands for either '$\circ$' or '$+$', whose duality was noted above. Each structural operator stands for one, or two (depending on the position), formula operators. Thus I stands for truth or falsity, E for the identity relation or its complement, $*$ for Boolean negation and $\bullet$ for the relational converse. The rules of $\boldsymbol{\delta}$**RA** are sound and complete with respect to the equational axiomatization of Chin & Tarski [12].

Like **LK**, Display Logic is suited to backwards search for proof of a sequent, provided that a sequent is provable without using the cut rule. In terms of backwards proof, the logical introduction rules eliminate the logical operators and replace them with corresponding structural operators. The cut-elimination theorem of **LK** applies in all Display Logics, provided that the rules satisfy certain conditions, which are relatively easily checked ([1]). Indeed the procedure for eliminating cuts from a proof has been automated, see §5.4.

Unlike **LK**, Display Logic has bi-directional rules, the display postulates. Therefore any proof search technique must avoid a naive application of these *ad infinitum*. Although this is a difficulty in developing a purely automatic proof search technique, this is not insurmountable, and requires further work. For example, the techniques described below are sufficient to obtain a decision procedure for the classical propositional calculus (see [7], §2.6.1 for details). In any case it is known that the theory of relation algebras is undecidable (see [16], p.432).

Note that although Display Logic is suited to backwards proof, we will always write a proof tree, or a rule, with the premises at the top and the conclusion at the bottom (except that a double line in a rule means that it is bi-directional), and we will refer to the order of steps in a proof as though it is done in a forward direction, from premises to conclusion. As an alternative to writing a rule with one premise $\mathcal{P}$ and a conclusion $\mathcal{C}$ separated by a horizontal line, we often write $\mathcal{P} \Longrightarrow \mathcal{C}$, and for a bi-directional rule we often write $\mathcal{P} \iff \mathcal{C}$. For a full explanation of Display Logic and $\boldsymbol{\delta}$**RA** see [12].

4 Isabelle

Isabelle is an interactive computer-based proof system, described in [18]. Its capabilities include higher-order unification and term rewriting. It is written in Standard ML; when it is running, the user can interact with it by entering further ML commands, and can program complex proof sequences in ML. Isabelle provides a number of basic proof steps for backwards proof (*tactics*), as well as *tacticals* for combining these. Isabelle also supports forward proof.

It has a simple meta-logic, an intuitionistic higher-order logic. The user then has to augment this by defining an object-level logic, though normally one would use one of several which are packaged with the Isabelle distribution [19].

Thus a logical system such as $\boldsymbol{\delta}$**RA** can be implemented in Isabelle so that the only proof rules available are those of $\boldsymbol{\delta}$**RA**. This contrasts with an implementation in other proof systems, where the $\boldsymbol{\delta}$**RA** rules would be added to the rules of the logic on which that proof system was based. (Of course, systems can be implemented in this way in Isabelle too; thus RALL [17] is implemented on top of the `HOL` Isabelle theory).

Isabelle theorems (including the axioms of the theory) may but need not involve Isabelle's metalogical operators. In the Isabelle implementation of Display Logic, an Isabelle theorem (of type `thm`) may be either a simple sequent of $\boldsymbol{\delta}$**RA**, or a sequent *rule*. Whereas a sequent of $\boldsymbol{\delta}$**RA** becomes an Isabelle theorem of the form $X \vdash Y$, a sequent rule becomes an Isabelle theorem containing also the operators `==>` (implication) or `==` (equality). For example, the $(\vdash \vee)$ rule (shown in §3) appears as

```
"$?X |- ?A, ?B ==> $?X |- ?A v ?B"
```

in Isabelle. (The '`?`' denotes a variable which can be instantiated, and the '`$`' denotes a structural variable or constant.) The variant $A \vdash A$ of the (id) rule (see §5.4) appears as `"?A |- ?A"`. The bi-directional `dcp` rule, shown in Fig. 1, appears as `"$?X |- $?Y == * $?Y |- * $?X"`.

Isabelle can be used to show the equivalence of $\boldsymbol{\delta}$**RA** to the various other systems; in most cases this serves only to confirm proofs appearing in the literature, and in many cases it can only provide proofs of the various inductive steps of a proof that requires structural induction. However it shows that $\boldsymbol{\delta}$**RA** is effective for proving the axioms or rules of the other systems.

5 Implementation in Isabelle

The syntax of $\boldsymbol{\delta}$**RA** has been implemented in Isabelle. The source files are presently available at `http://arp.anu.edu.au:80/~jeremy/files`, and further details may be found in [7]. The syntax distinguishes structure and formula variables; externally, at the user level, a structure variable is preceded by '`$`'. The method used for this was taken directly from the Isabelle theory `LK` ([19], Ch 6).

The $\boldsymbol{\delta}$**RA** rules have been implemented in Isabelle. Some particularly useful derived rules are shown in Fig. 1. It will be noticed that the two unary structural operators commute, and both distribute (in a sense) over both the binary structural operators. These distributive rules are used, for example, in §7.1. Further, there is a one-directional distributive rule for ';' over ','. The rules `blcEa` and `blcEs`, in particular, are elegant results which are much easier to prove for the case of a proper relation algebra.

$$\begin{array}{c} X \vdash * * Y \\ \hline\hline X \vdash Y \end{array}\ (\texttt{rssS}) \qquad \begin{array}{c} X \vdash Y \\ \hline\hline *Y \vdash *X \end{array}\ (\texttt{dcp}) \qquad \begin{array}{c} X \vdash \bullet \bullet Y \\ \hline\hline X \vdash Y \end{array}\ (\texttt{rbbS}) \qquad \begin{array}{c} X \vdash Y \\ \hline\hline \bullet X \vdash \bullet Y \end{array}\ (\texttt{blbl})$$

$$\begin{array}{c} *(X, Y) \vdash Z \\ \hline\hline *Y, *X \vdash Z \end{array}\ (\texttt{stcdista}) \qquad \begin{array}{c} *(X ; Y) \vdash Z \\ \hline\hline *X ; *Y \vdash Z \end{array}\ (\texttt{stscdista})$$

$$\begin{array}{c} \bullet(X ; Y) \vdash Z \\ \hline\hline \bullet Y ; \bullet X \vdash Z \end{array}\ (\texttt{blscdista}) \qquad \begin{array}{c} (\bullet X, \bullet Y) \vdash Z \\ \hline\hline \bullet(X, Y) \vdash Z \end{array}\ (\texttt{blcdista})$$

$$\frac{(X ; Y), (X ; Z) \vdash W}{X ; (Y, Z) \vdash W}\ (\texttt{sccldista}) \qquad \frac{(X ; Z), (Y ; Z) \vdash W}{(X, Y) ; Z \vdash W}\ (\texttt{sccrdista})$$

$$\begin{array}{c} X ; Y \vdash Z \\ \hline\hline \bullet Y ; \bullet X \vdash \bullet Z \end{array}\ (\texttt{tagae}) \qquad \begin{array}{c} \bullet * X \vdash Y \\ \hline\hline * \bullet X \vdash Y \end{array}\ (\texttt{bsA}) \qquad \begin{array}{c} E \vdash X \\ \hline\hline *E \vdash X \end{array}\ (\texttt{sEa}) \qquad \begin{array}{c} I \vdash X \\ \hline\hline *I \vdash X \end{array}\ (\texttt{sIa})$$

$$\begin{array}{c} \bullet X, E \vdash Y \\ \hline\hline X, E \vdash Y \end{array}\ (\texttt{blcEa}) \qquad \begin{array}{c} Y \vdash \bullet X, E \\ \hline\hline Y \vdash X, E \end{array}\ (\texttt{blcEs})$$

$$\frac{A \vdash X \quad B \vdash X}{A \vee B \vdash X}\ (\texttt{orA}) \qquad \frac{X \vdash A \quad X \vdash B}{X \vdash A \wedge B}\ (\texttt{andS})$$

Fig. 1. Selected Derived Rules

Of the $\boldsymbol{\delta}$**RA** logical introduction rules, some can be applied always (in a backwards proof) when the relevant formula is displayed. Others, namely $(\vdash \wedge)$, $(\vee \vdash)$, $(\vdash \circ)$ and $(+ \vdash)$, are subject to the "constraint" that the structure on the other side of $\vdash$ be of a particular form. The rules `andS` and `orA` are equivalent to $(\vdash \wedge)$ and $(\vee \vdash)$. `andS` and `orA` can be applied always when the relevant formula is displayed. Unfortunately the introduction rules for '$\circ$' and '$+$' have no such equivalents. Thus the introduction of '$\circ$' on the right of $\vdash$ is always subject to the "constraint" that the structure on the left of $\vdash$ be of a particular form; similarly for introducing '$+$' on the left.

As in classical sequent calculus, the *modus operandi* for proving a sequent is to start the proof from the bottom, ie, from the goal, and work "backwards". The first step is, where possible, to remove all the formula (logical) operators, using the logical introduction rules. In Display Logic (unlike classical sequent calculus) this requires that the formula which is to be "broken up" must first be displayed. As any formula can be displayed, this is not a difficulty, except in that it is tedious to select display postulates one-by-one. The tactics described in the next two subsections were first written to help in this style of proof.

5.1 Display Tactics

The functions described below were written to help in the process of displaying a chosen substructure.

`disp_tac : string -> int -> tactic` replaces a subgoal by one in which a selected structure is displayed. The structure selected is indicated by a string argument, of which the first character is '`|`' or '`-`', to indicate the left or right side of the turnstile. Remaining characters are

'`l`' or '`r`', to indicate the left or right operand of the ',' operator

'`L`' or '`R`', to indicate the left or right operand of the ';' operator

'`*`' or '`@`', to indicate the operand of the '$*$' or the '$\bullet$' operator

For example, in the (sub)goal $R \vdash ((*T; \bullet * S), \bullet * Q); Y$ the string `"-L"` denotes the sub-structure $((*T; \bullet * S), \bullet * Q)$. Thus `disp_tac "-L"` performs the first (bottom) step of the proof example shown below.

`fdisp : string -> thm -> thm` is the corresponding function for forward proof, displaying the selected part of the conclusion of the given theorem.

`d_b_tac : string -> (int -> tactic) -> int -> tactic`

`d_b_tac` *str tacfn sg* takes subgoal *sg*, displays the part selected by *str* (as for `disp_tac`) and applies *tacfn* to the transformed subgoal. It then applies the reverse of the display steps to each resulting subgoal. This requires that the subgoals produced by *tacfn* are in a suitable form to permit the display steps used to be reversed. This will be the case if *tacfn* only affects the displayed sub-structure, and leaves the rest of the sequent alone. This was the purpose of deriving the rules `andS` and `orA` (which satisfy this requirement) from `ands` and `ora` (which do not).

For example, if we start with the subgoal $R \vdash ((*T; \bullet * S), \bullet * Q); Y$ then `d_b_tac "-L" (rtac mrs)` applies the monotonicity rule $(\vdash M)$ (`mrs`) to the sub-structure $(*T; \bullet * S), \bullet * Q$ by the following steps (read upwards)

$$\dfrac{\dfrac{\dfrac{R \vdash (*T; \bullet * S); Y}{R; * \bullet Y \vdash *T; \bullet * S}\ (\text{reverse of } \texttt{disp\_tac "-L"})}{R; * \bullet Y \vdash (*T; \bullet * S), \bullet * Q}\ (\texttt{mrs})}{R \vdash ((*T; \bullet * S), \bullet * Q); Y}\ (\texttt{disp\_tac "-L"})$$

`fdispfun : string -> (thm -> thm) -> thm -> thm`
is a corresponding operation for forward proof. `fdispfun` *str thfn th* displays the part of theorem *th* indicated by string *str*, applies transformation *thfn* to the result, then reverses the display steps used.

5.2 Search-and-Replace Tactics

Tactics were written to display all structures in a sequent in turn, and apply any appropriate transformations (selected from a given set) to the structure. These can be used to rewrite all substructures of a particular form, for example, to rewrite all occurrences of "$X, (Y, Z)$" to "$(X, Y), Z$".

`glob_tac :`
`(term -> int -> (int -> tactic) list) -> int -> tactic`
The first argument of `glob_tac` is a function *actfn*, which is called as *actfn tm asp*, where *tm* is a subterm, and *asp* has the value 0 or 1 to indicate that *tm* is displayed as the antecedent or succedent part of a sequent. *actfn tm asp* should return either `[]` or `[`*tacf*`]`, where, when *sg* is a subgoal number, *tacf sg* is a tactic. When *tm* is displayed as the antecedent (if $asp = 0$) or succedent (if $asp = 1$), *tacf* should be a tactic which changes only *tm*.
Then `glob_tac` *actfn sg* displays in turn every subterm of subgoal *sg*, and applies the tactic, if any, given by *actfn tm asp*, (where *tm* is the subterm displayed on the side indicated by *asp*) to the sequent thus produced. All the display steps are eventually reversed, in the same way as for `d_b_tac`.
`glob_tac` uses a top-down strategy in that it first tests the whole structure with the function *actfn*, then, recursively, each sub-structure. If *actfn* returns a tactic function (ie, not the empty list) then the changed structure is tested again.

`bup_tac` is like `glob_tac`, but uses a bottom-up strategy, displaying and testing the smallest sub-structures first.

`fgl_fun, fbl_fun :`
`(term -> int -> (thm -> thm) list) -> thm -> thm`
These functions are like `glob_tac` and `bup_tac`, but for doing forward proof. Their first argument is again a function *actfn*, called as *actfn tm asp*. Here *actfn tm asp* should return either `[]` or `[`*thtr*`]`, where *thtr* is a theorem transformation, ie a function of type `thm -> thm`, which changes only the term *tm* displayed on the side indicated by *asp*.

The first implementation of these tactics and functions was described in [7], §2.5.3. A later implementation gives the user great flexibility in programming the search strategy, enabling him/her to specify (for example) whether to traverse the structure top-down or bottom-up, whether or not to attempt further changes to a rewritten substructure, whether or not to repeat the whole procedure when any substructure is rewritten, etc. This implementation is described in [8, §4.4].

5.3 Application of the Search-and-Replace Tactics

We now describe some uses of the search-and-replace tactics. Provided the function *actfn*, as described in §5.2, returns an action which changes only the displayed subterm *tm*, these tactics perform local "rewrites" on subexpressions.

Examples of the use of these tactics are (1) to eliminate $**$ and $\bullet\bullet$, (2) to use the various derived distributive laws to push $*$ and $\bullet$ down, and (3) to eliminate logical operators, where possible, using a set of logical introduction rules including `andS` and `orA` (see Fig. 1) – this is only for backward proof.

As these examples illustrate, we can achieve the power of a procedure which rewrites using equalities or equivalences, such as `rewrite_tac` and `rewrite_rule` in Isabelle and `REWRITE_TAC` and `REWRITE_RULE` in HOL, even though the $\boldsymbol{\delta}\mathbf{RA}$ formalization does not contain any general rewriting facility. However $\boldsymbol{\delta}\mathbf{RA}$ requires more than simple rewriting due to the logical introduction rules with "constraints".

The fact that `pr_intr` is available only for backward proof illustrates that these tactics can be used with uni-directional rules, such as the logical introduction rules, as well as with bi-directional rules. This useful aspect of the tactics contrasts with the more typical rewriting tactics, such as in Isabelle and HOL, where the rules used to rewrite an expression *must* be equalities.

In [12], §4, a function τ which translates structures to formulae – for example, to convert $B; \bullet A \vdash \bullet D, *C$ to $B \circ \smile A \vdash \smile D \vee \neg C$ – is crucial in the proof of soundness of $\boldsymbol{\delta}\mathbf{RA}$. Its implementation in a forward proof is another example of the use of a search-and-replace function. For this we can use some of the logical introduction rules, in their original form (as given in [12]), using `fbl_fun`. Note that this must be done bottom up, ie, on the smallest sub-structures first, because only at the bottom are the sub-structures also formulae.

5.4 Other Tactics and Methods

idf_tac In the identity rule (id) $p \vdash p$, p stands for a primitive proposition, not an arbitrary formula. It is, however, true that $A \vdash A$ for any formula A; this is proved by induction (see [12], Lemma 2). The restriction to primitive propositions is not reflected in the Isabelle implementation, which contains the rule $A \vdash A$, where A is a variable standing for any formula. However the tactic `idf_tac` will convert an identity subgoal such as $q \circ (\smile p \vee \neg r) \vdash q \circ (\smile p \vee \neg r)$ into three separate subgoals $\quad q \vdash q \quad p \vdash p \quad r \vdash r$. The tactic can therefore provide a proof, from the rule $p \vdash p$, of any given instance of the general theorem (which is provable only by induction) that $A \vdash A$ for any formula A.

This tactic uses a set of six derived results (one for each logical operator, of which two examples are shown).

$$\frac{A_1 \vdash A \quad B_1 \vdash B}{A_1 \circ B_1 \vdash A \circ B}\,(\texttt{cong\_comp}) \qquad \frac{A \vdash A_1}{\neg A_1 \vdash \neg A}\,(\texttt{cong\_not})$$

Structural operators from logical ones It is also possible to implement the function τ in a backwards proof. We can "create" logical connectives (as we read a proof upwards); this can on occasions help in managing a complex proof. The tactic `tau_tac` does this. It uses variants of the logical introduction rules where all the premises are "structure-free" and "connective-free", ie of the form $A \vdash X$ or $X \vdash A$. Structural or logical operators appear only in their conclusions.

The inverse of τ is implemented for forward proof. This use of the search-and-replace method can only be done top down (using `fgl_fun`), and it uses the cut rule. Not all formula operators will necessarily be changed to structural ones: for example, '$\circ$' or '$\wedge$' in a succedent position and '$+$' or '$\vee$' in an antecedent position cannot be converted to structural operators.

Flipping theorems It may be observed in Fig. 1 of [12] that many of the rules in the two columns are symmetric about the turnstile, for example

$$(A;\vdash)\ \frac{X;(Y;Z) \vdash W}{\overline{(X;Y);Z \vdash W}}\,(\texttt{arA}) \qquad (\vdash A;)\ \frac{W \vdash X;(Y;Z)}{\overline{W \vdash (X;Y);Z}}\,(\texttt{arS})$$

In fact most rules in the right-hand column of that figure could be derived from the corresponding rules in the left-hand column by "flipping" them about the turnstile, ie, interchanging the parts before and after $\vdash$. Likewise, "flipped" versions were derived for those rules in Fig. 1 which use only structural operators. The procedure for doing this has been formalized in the theorem transformation functions `flip_st_p` and `flip_st_c`, of type `thm -> thm`. These use the theorem `dcp` (see Fig. 1) to swap the sides of each sequent (by putting a $*$ in front of each side), use the distributive laws to push the $*$ down as far as possible, and rename variables to absorb the $*$. The transformations require the rules to be in structural form (see §6.1 below). This applies generally in display logics – having proved one theorem or rule in structural form, you then get one "free", with each part moved to the other side of the turnstile. The theorem transformations `flip_bl_p` and `flip_bl_c` do the same thing, but with $\bullet$ instead of $*$. These transform a theorem to a corresponding one in terms of the converse relations.

Automated cut-elimination Belnap's original proof of cut-elimination is operational; see the proof sketch in [12, Appendix]. It was implemented as follows.

- The rules used in a proof are stored in the nodes of a tree using an appropriate ML datatype, where each node represents a single proof step. Thus, given the original sequent, we can reproduce the original proof.
- Functions were written in ML to turn an Isabelle proof, using the various compound tactics described in §5.2, into a proof consisting of single steps, and then to represent that proof as a tree, as above.

- Functions were written to allow us to rearrange the nodes of such proof trees to push a cut upwards until the cut-formula becomes principal in both premises of that cut (either due to an introduction rule or an axiom $p \vdash p$).
- Such a cut is either eliminated completely or is converted to cuts on smaller formulae using a look-up table that gives a recipe of how to do so for each logical connective. Display logic gurantees that such a recipe always exists.
- This procedure is applied recursively to remove all cuts from the proof tree.

Full details are given in [7, Ch. 5].

One novel use of this procedure would be to convert each proof of Chin and Tarski [6] into a $\boldsymbol{\delta}$**RA** proof as follows. Simply begin to convert the proof by following the text. Sooner or later, the text "stores" the current reasoning as a lemma. At this point, we let the converted text constitute a premise of cut. We then continue to convert the text, until the lemma is used. At this point we insert a cut in the converted proof. Now applying the automated cut-elimination procedure will give a new, purely cut-free proof. Often, this proof is shorter, although, in general, it will be exponentially longer.

6 Results from the δRA implementation

6.1 Theorems proved using Isabelle

We give some examples of theorems and derived rules that have been proved in $\boldsymbol{\delta}$**RA**, using Isabelle. They show how some relation algebraic results can be turned into structural derived rules, ie, rules in which all the operators are structural, and all the variables denote arbitrary structures. This (together with the display postulates) enables them to be reused in doing proofs in structural form.

Chin & Tarski [6] give a number of results whose proof is far from simple. For example, their Theorem 2.7 and the corresponding structural rule are

$$(a \circ b) \wedge c \vdash a \circ ((\smile a \circ c) \wedge b) \qquad \frac{X ; ((\bullet X ; Z), Y) \vdash W}{(X ; Y), Z \vdash W}$$

Chin & Tarski [6] Theorem 2.11 and its corresponding structural rule are

$$(r \circ s) \wedge (t \circ u) \vdash (r \circ ((\smile r \circ t) \wedge (s \circ \smile u))) \circ u \quad \frac{(U ; ((\bullet U ; W), (V ; \bullet X))) ; X \vdash Z}{(U ; V), (W ; X) \vdash Z}$$

The Dedekind rule ([17], §5.3; [2], §2.1) and its corresponding structural rule are

$$(q \circ r) \wedge s \vdash (q \wedge (s \circ \smile r)) \circ (r \wedge (\smile q \circ s)) \qquad \frac{(X, (Z ; \bullet Y)) ; (Y, (\bullet X ; Z)) \vdash W}{(X ; Y), Z \vdash W}$$

All these results were proved in $\boldsymbol{\delta}$**RA**, using Isabelle.

Some interesting theorems were useful in various areas of [7]. Again, they can be proved in the form of structural rules. These show an interesting similarity between the Boolean and relational "times" operators, ',' and ';', and their identities, I and E. Firstly, we have

$$\frac{((X;Y),E);Z \vdash W}{\overline{((X,\bullet Y);I),Z \vdash W}} \,(\texttt{thab})$$

Next we have two results which can be proved from it.

$$\frac{(X,E);(Y,Z) \vdash W}{\overline{((X,E);Y),Z \vdash W}} \,(\texttt{th78sr}) \qquad \frac{(X;I),(Y;Z) \vdash W}{\overline{((X;I),Y);Z \vdash W}} \,(\texttt{th56sr})$$

These results have the following corollaries, giving examples of cases where the Boolean and relational products of two quantities are equal.

$$\frac{(X,E);(E,Z) \vdash W}{\overline{(X,E),(E,Z) \vdash W}} \,(\texttt{cor78i}) \qquad \frac{(X;I),(I;Z) \vdash W}{\overline{(X;I);(I;Z) \vdash W}} \,(\texttt{cor56i})$$

6.2 Proving the RA axioms

Using Isabelle we can mechanize the formal proofs of completeness of $\boldsymbol{\delta}$**RA** with respect to **RA**. Here, we sketch how this is done. We show, informally, how any result provable in **RA** can be proved in $\boldsymbol{\delta}$**RA**. This completeness result is weaker than Theorem 6 of [12] in that we only show how to derive, in $\boldsymbol{\delta}$**RA**, any **RA**-equation (expressed in terms of *formulae*); by contrast, [12] also deals with $\boldsymbol{\delta}$**RA**-sequents involving *structures* whose translation into a formula inequality is valid in **RA**.

In [12] a (Lindenbaum) relation algebra is constructed from $\boldsymbol{\delta}$**RA**. Here we consider the proof system which corresponds to **RA**. By the completeness theorem of Birkhoff [3], discussed in [14], the proof rules for **RA** consist only of substituting terms for variables in the relevant equations (defining a Boolean algebra and (RA2) to (RA8)), and replacing equals by equals.

Firstly, following [12], Theorem 6, we let $A = B$ mean $A \vdash B$ and $B \vdash A$; more formally, we introduce the following rules

$$\frac{A \vdash B \quad B \vdash A}{A = B}(\texttt{eqI}) \qquad \frac{A = B}{A \vdash B}(\texttt{eqD1}) \qquad \frac{A = B}{B \vdash A}(\texttt{eqD2})$$

This equality was easily proved to be reflexive, symmetric and transitive. We define $A \leq B$ as $(A \wedge B) = A$, as in **RA**. It is then proved that $A \leq B \iff A \vdash B$.

Then the **RA** axioms can be proved; (RA2) to (RA8) are proved using Isabelle. Axiom (RA1), which says that the logical operators form a Boolean algebra, follows from the fact that $\boldsymbol{\delta}$**RA** contains classical propositional logic.

In using the **RA** axioms we would implicitly rely on the equality operator $=$ being symmetric and transitive, which we proved, as indicated above. The proof system for **RA** also uses the fact, as noted above, that equality permits substitution of equals. That is, suppose $C[A]$ and $C[B]$ are formulae such that $C[B]$ is obtained from $C[A]$ by replacing some occurrences of A by B. If we now prove in **RA** that $A = B$ and $C[A] = D$ then we could deduce $C[B] = D$ by substituting B for A.

To show that this can be done in $\boldsymbol{\delta}$**RA** we have to use induction on the structure of $C[\cdot]$. Each induction step uses one of six derived results, of which two are shown below.

$$\frac{A_1 = A \quad B_1 = B}{A_1 \wedge B_1 = A \wedge B}\ (\texttt{eqc\_and}) \qquad \frac{A_1 = A}{\smile A_1 = \smile A}\ (\texttt{eqc\_conv})$$

Alternatively, any given instance of this result could be proved with the help of the tactic `idf_tac`, described in §5.4. Details are given in [7].

7 Comparing Different Logical Calculi for RA Using $\boldsymbol{\delta}$RA

In this section we describe how we used the implementation of $\boldsymbol{\delta}$**RA** in Isabelle to demonstrate the soundness of some other logical systems for relation algebra. (As described in §6.2, we did the same for **RA**; as we use **RA** as a "reference point" we treated this as a demonstration of the completeness of $\boldsymbol{\delta}$**RA**).

7.1 Gordeev's term rewriting system for RA

Gordeev [10] has given a system **NRA** of rules for rewriting relational equations of the form $F = \top$ (where F is a relational formula and $\top$ is the unique largest element). A derivation of F in **NRA** is a sequence of permitted rewrites ("reductions") $F \longrightarrow \cdots \longrightarrow \top$. Thus **NRA** proves equations of the form $F = \top$. However it is known that every Boolean combination of equations is equivalent to an equation of the form $F = \top$ (see [16], pp. 435–6, pp. 439–440). Gordeev has shown that F is derivable in **NRA** iff $F = \top$ is derivable in **RA**.

It is expected that details of **NRA** will be published in due course; the actual rules are not pertinent here, since we simply outline how Isabelle was used as the basis of a proof of the soundness of **NRA**.

NRA, uses "literals" (L) and "positive terms" (A, B, C, D). Literals are of the shape P, $\neg P$, $\smile P$, $\neg \smile P$, $\top$, $\bot$, **0** or **1**, for any given primitive relation symbol P. Positive terms are built up from literals using the binary operators. Any **RA** term is reducible to a positive term by the distributive rules which show how the unary operators distribute over the binary operators (see Fig. 1 for the structural counterparts).

In **NRA** there are 12 rewriting rules (NRA1)-(NRA12), but, given a formula F, some may only be used to rewrite F or a conjunct of F. The permitted rewrites, or "reductions", are

- $F[G'] \longrightarrow F[G'']$, where $G' \longrightarrow G''$ is an instance of any rule (NRAi) except for (NRA9) and (NRA10)
- $G_1 \wedge G_2 \wedge \ldots \wedge G'_j \wedge \ldots \wedge G_k \longrightarrow G_1 \wedge G_2 \wedge \ldots \wedge G''_j \wedge \ldots \wedge G_k$ where $1 \leq j \leq k$ and $G'_j \longrightarrow G''_j$ is an instance of (NRA9) or (NRA10)

We take it that $F[\cdot]$ above must be such that G is a 'sub-positive-term' of $F[G]$, ie, that G is not a literal properly contained in a larger literal within $F[G]$ (for example, $F[G]$ may not be $\neg G \vee H$). This point was not spelt out in the brief communication [10], and we realized the need for it only in doing the proof in Isabelle. This gives an example of the value of a computer-based theorem prover in ensuring such details are not overlooked.

The following lemma explains the **NRA** reduction step in terms of $\boldsymbol{\delta}$**RA**.

Lemma 1. *Whenever $F' \longrightarrow F''$ in **NRA**, then $\top \vdash F'' \Longrightarrow \top \vdash F'$ in **δRA**.*

Proof. For rules other than (NRA9) and (NRA10), we show that $F'' \vdash F'$ in **δRA**. Let $F' = F[G']$ and $F'' = F[G'']$, where the **NRA** rule is $G' \longrightarrow G''$. For each rule the **δRA** theorem $G'' \vdash G'$ was proved in Isabelle.

From $G'' \vdash G'$ we show $F[G''] \vdash F[G']$ by induction on the structure of $F[\cdot]$. As $F[\cdot]$ uses only literals and the binary operators, the inductive steps rely on the four **δRA**-theorems `cong_or`, `cong_and`, `cong_comp` and `cong_rs` (see §5.4), and the identity rule $A \vdash A$.

Since $F'' \vdash F'$, if we assume $\top \vdash F''$, then the (cut) rule gives $\top \vdash F'$.

We now look at rules (NRA9) and (NRA10). For these rules, where the **NRA** rule is $G' \longrightarrow G''$, we have proved the **δRA** derived rule $\top \vdash G'' \Longrightarrow \top \vdash G'$ in Isabelle. So from this we need to show $\top \vdash F[G''] \Longrightarrow \top \vdash F[G']$, where G' is a conjunct of $F[G']$. This is shown by induction on the number of conjuncts in $F[\cdot]$. The inductive step uses the following **δRA**-theorem.

$$\frac{\top \vdash G1'' \Longrightarrow \top \vdash G1' \qquad \top \vdash G2'' \Longrightarrow \top \vdash G2'}{\top \vdash G1'' \wedge G2'' \Longrightarrow \top \vdash G1' \wedge G2'} \; (\texttt{G910meth})$$

$\square$

`G910meth` is a theorem whose use of Isabelle's metalogic is more complex than others seen so far, with nested occurrences of the metalogical operator $\Longrightarrow$.

Lemma 1 implies that for every **NRA**-derivation of F there is a **δRA**-derivation of $\top \vdash F$. Thus **NRA** is sound with respect to **RA**.

7.2 A point-variable sequent calculus

In [7], Ch. 4, we described the sequent calculus system of Maddux [15], $\mathcal{M}$, for relation algebras, and showed how its sequents and proofs correspond to sequents and proofs in **δRA**. We characterized the sequents in $\mathcal{M}$ which can be translated into **δRA**by describing the pattern of relation variables and point variables in the $\mathcal{M}$-sequent as a graph, and showing that whether or not an $\mathcal{M}$-sequent can be translated into **δRA** depends on the shape of the graph.

As a sequent in $\mathcal{M}$ can often be translated to any of several sequents in **δRA**, we showed that these several sequents are equivalent in **δRA**.

The result relating proofs in the two systems was that if an $\mathcal{M}$-sequent $\mathcal{S}$ can be translated into a **δRA**-sequent $\mathcal{S}'$, then any $\mathcal{M}$-proof of $\mathcal{S}$ can be translated into a **δRA**-proof of $\mathcal{S}'$, provided that the $\mathcal{M}$-proof uses at most four point variables. This required showing how such a proof in $\mathcal{M}$ can be divided into parts each of which could be turned into a part of a proof in **δRA**.

As not all $\mathcal{M}$-sequents containing at most four point variables can be translated into **δRA**, not all the intermediate stages of such a proof could be translated into **δRA**, that is, not every individual step of the $\mathcal{M}$-proof could be converted into a corresponding step of a proof in **δRA**. Rather, a larger portion of the $\mathcal{M}$-proof had to be treated as a unit; we showed that we could always rearrange the $\mathcal{M}$-proof to get these portions of a particular form, which conformed to a theorem that we had proved in Isabelle for **δRA**.

This result provided a constructive proof of one direction of the result of Maddux ([15], Thm. 6) that the class of relation algebras is characterized by the $\mathcal{M}$-sequents provable using four variables.

8 Conclusion

We described an implementation, in Isabelle, of the display calculus $\boldsymbol{\delta}$**RA** for relation algebras. This is the first implementation of any display calculus. We intend to generalise it to other display calculi in future work. Although a $\boldsymbol{\delta}$**RA** proof involved frequent and tedious use of the display postulates, we provided a number of tactics and other functions to make this aspect of constructing proofs easier. We also provided tactics to search for certain patterns and replace them wherever they are found, giving the power of a rewrite rule in a term rewriting system. Most, but not all, of the rules for introducing logical connectives could be used in this manner. Certain other tactics and functions, such as those which transform between structural and logical forms of sequents, have proved convenient in proofs on occasions. There is scope for further work to devise more automatic techniques than those described here.

The use of display postulates to "display" a subterm resembles the use of window inference [20],[13] to "focus" upon a subterm. Further work is needed to connect these two modes of reasoning.

Some useful derived rules were proved, for example those showing how the unary operators distribute over the binary operators. We showed how the $\boldsymbol{\delta}$**RA** system can justify inferences in the (axiomatic) theory of relation algebras. Although this result appears as Theorem 6 of [12], the published proof omits most of the detail. The detail is provided in the Isabelle proofs, giving an example of the value of a mechanized theorem prover to confirm the details of proofs which are too voluminous to publish or even too tedious to check by hand.

Some important (and difficult) theorems from the literature were also proved, for example some from [6]. Other interesting theorems were discovered and proved, some of which were needed for the work outlined in §7.2.

In §7.1 we described an equational theory due to Gordeev for relation algebras, and showed how its inferences can be justified in $\boldsymbol{\delta}$**RA**. As mentioned there, we obtained a corresponding result for the sequent calculus system of Maddux. This showed how Isabelle could be used to demonstrate the relationship between different logical systems for relation algebra.

Theorems 3.6 and 3.8 of [5] show that Peirce algebras can be embedded inside relation algebras in various ways. The authors also show that Peirce algebras form the basis of most of the common Knowledge Representation languages like KL-ONE [4]. Minor modifications of our system should give mechanised proof systems for these logics.

References

1. Nuel D. Belnap, Display Logic, Journal of Philosophical Logic 11 (1982), 375–417.
2. Rudolf Berghammer & Claudia Hattensperger, Computer-Aided Manipulation of Relational Expressions and Formulae Using RALF, preprint.
3. Garrett Birkhoff, On the Structure of Abstract Algebras, Proc. Cambridge Phil. Soc. 31 (1935), 433-454.
4. R.J. Brachman & J.G. Schmolze, An overview of the KL-ONE knowledge representation system, Cognitive Science 9(2) (1985), 171–216.
5. Chris Brink, Katarina Britz & Renate A. Schmidt, Peirce Algebras, Formal Aspects of Computing 6 (1994), 339–358.
6. Louise H. Chin & Alfred Tarski, Distributive and Modular Laws in the Arithmetic of Relation Algebras, University of California Publications in Mathematics, New Series, I (1943–1951), 341–384.
7. Jeremy E. Dawson, Mechanised Proof Systems for Relation Algebras, Grad. Dip. Sci. sub-thesis, Dept of Computer Science, Australian National University. Available at `http://arp.anu.edu.au:80/~jeremy/thesis.dvi`
8. Jeremy E. Dawson, Simulating Term-Rewriting in LPF and in Display Logic, submitted. Available at `http://arp.anu.edu.au:80/~jeremy/rewr/rewr.dvi`
9. Jean H. Gallier, Logic for Computer Science : Foundations of Automatic Theorem Proving, Harper & Row, New York, 1986.
10. Lev Gordeev, personal communication.
11. Rajeev Goré, Intuitionistic Logic Redisplayed, Automated Reasoning Project TR-ARP-1-95, ANU, 1995.
12. Rajeev Goré, Cut-free Display Calculi for Relation Algebras, Computer Science Logic, Lecture Notes in Computer Science 1249 (1997), 198–210. (or see `http://arp.anu.edu.au/~rpg/publications.html`).
13. Jim Grundy, Transformational Hierarchical Reasoning, The Computer Journal 39 (1996), 291–302.
14. Gérard Huet & Derek C. Oppen, Equations and Rewrite Rules – A Survey, in Formal Languages: Perspectives and Open Problems, R.V. Book (ed), Academic Press (1980), 349–405.
15. Roger D. Maddux, A Sequent Calculus for Relation Algebras, Annals of Pure and Applied Logic 25 (1983), 73–101.
16. Roger D. Maddux, The Origin of Relation Algebras in the Development and Axiomatization of the Calculus of Relations, Studia Logica 50 (1991), 421–455.
17. David von Oheimb & Thomas F. Gritzner, RALL: Machine-supported proofs for Relation Algebra, Proceedings of CADE-14, Lecture Notes in Computer Science 1249 (1997), 380–394.
18. Lawrence C. Paulson, The Isabelle Reference Manual, Computer Laboratory, University of Cambridge, 1995.
19. Lawrence C. Paulson, Isabelle's Object-Logics, Computer Laboratory, University of Cambridge, 1995.
20. Peter J. Robinson & John Staples, Formalizing a Hierarchical Structure of Practical Mathematical Reasoning, J. Logic & Computation, 3 (1993), 47–61.
21. Heinrich Wansing, Sequent Calculi for Normal Modal Propositional Logics, Journal of Logic and Computation 4 (1994), 124–142.