Tool-Based Verification of a Relational Vertex Coloring Program

Rudolf Berghammer[1], Peter Höfner[2,3], and Insa Stucke[1]

[1] Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Germany
[2] NICTA, Australia
[3] Computer Science and Engineering, University of New South Wales, Australia

Abstract. We present different approaches of using a special purpose computer algebra system and theorem provers in software verification. To this end, we first develop a purely algebraic while-program for computing a vertex coloring of an undirected (loop-free) graph. For showing its correctness, we then combine the well-known assertion-based verification method with relation-algebraic calculations. Based on this, we show how automatically to test loop-invariants by means of the RelView tool and also compare the usage of three different theorem provers in respect to the verification of the proof obligations: the automated theorem prover Prover9 and the two proof assistants Coq and Isabelle/HOL. As a result, we illustrate that algebraic abstraction yields verification tasks that can easily be verified with off-the-shelf theorem provers, but also reveal some shortcomings and difficulties with theorem provers that are nowadays available.

1 Introduction

Provably correct programs can be obtained in different ways. Formal program verification is one of them. It means to prove with mathematical rigor that a given program meets a given formal specification of the problem. In case of imperative programs the use of pre- and post-conditions as specifications and intermediate assertions for the verification is a widely accepted and frequently used technique. Besides proof rules for the control structures of the programming language used it requires formal specifications for the data types on which the programs are applied. Experience has shown that algebraic/axiomatic specifications or modeling by algebraic structures are most suitable for that. In the present paper we consider a graph-theoretic problem and use relation algebra for modeling undirected graphs, single vertices, sets of vertices as well as functions which assign values to vertices.

In the present paper we consider a graph-theoretic problem and use relation algebra for modeling undirected graphs. The axiomatization of relation-algebraic calculus started with [26]. The calculus is widely used and many examples in the context of program verification can be found in the literature, e.g., [2,3,4,6,7]. For the use of relation algebra in graph theory we refer again to [24,25].

Relation-algebraic proofs are precise and hence allow formal first-order reasoning, often even equational reasoning. This is a vantage point for the use of theorem provers as, for instance, demonstrated in [15,17,18]. Based on these positive experiences, in [8,9] the automated theorem prover Prover9 [20] is used for the automated verification of proof obligations appearing in the assertion-based verification of relational programs. This paper is a continuation of as well as a step further in this work. We consider a well-known graph theoretical problem, viz. vertex coloring. However, we do not restrict ourselves to the verification of the proof obligations via an automated theorem prover. We aim to gain more experience with tool support in formal verification of relational programs. Therefore, we also investigate the use of two different proof assistants tools, viz. Coq [11] and Isabelle/HOL [21], and of a specific purpose computer algebra system for relation algebra, viz. RelView [5,30]. The paper illustrates that algebraic abstraction yields verification tasks that can be verified with off-the-shelf theorem provers, but also reveals some shortcomings and difficulties with tools that are nowadays available.

One aim of the paper is to provide a guideline on how to get started with different tools with different approaches and possibilities when computations and mechanical proofs in relation algebra are desired or required. For that reason we restrict ourselves to a single and not too difficult problem. By this the general approach is easily visible and is not hidden by complex technical details. All input files and proof scripts can be found in the web [32].

2 Relation-Algebraic Preliminaries

To model undirected graphs, single vertices, sets of vertices and colorings, we will use binary relations and manipulate and calculate with such objects in a purely algebraic manner. Therefore, we recall the fundamentals of relation algebra based on the homogeneous approach of [26], its developments in [13,19,27] and the generalization to heterogeneous relation algebra in [24,25].

Set-theoretic relations form the standard model of relation algebras. We assume the reader to be familiar with the basic operations on them, viz. R^T (transposition), $\overline{R}$ (complementation), $R \cup S$ (union), $R \cap S$ (intersection), RS (composition), the predicates $R \subseteq S$ (inclusion) and $R = S$ (equality), and the special relations O (empty relation), L (universal relation), and I (identity relation). The three Boolean operations $\overline{}$, $\cup$ and $\cap$, the order $\subseteq$ and the two constants O and L form Boolean lattices. Well-known properties of set-theoretic relations are $\overline{R^\mathsf{T}} = \overline{R}^\mathsf{T}$, $(R \cup S)^\mathsf{T} = R^\mathsf{T} \cup S^\mathsf{T}$, $(R \cap S)^\mathsf{T} = R^\mathsf{T} \cap S^\mathsf{T}$, $(R^\mathsf{T})^\mathsf{T} = R$, $(RS)^\mathsf{T} = S^\mathsf{T} R^\mathsf{T}$, and the monotonicity of the transposition operation. Furthermore, union, intersection and composition are monotonic in both arguments.

The theoretical framework for these rules (and many others) to hold is that of a (heterogeneous) *relation algebra* with typed relations as elements. Typing means that each relation has a source and a target and we write $R : X \leftrightarrow Y$ to express that X is the source and Y is the target of R. We call $X \leftrightarrow Y$ the type of R. As constants and operations of a relation algebra we have those of

set-theoretic relations, where we (as usual) overload the symbols O, L and I, i.e., avoid the binding of types to them. The axioms of a relation algebra are

(1) the axioms of a Boolean lattice for all relations of the same type under the Boolean operations, the order, empty relation and universal relation,
(2) the associativity of composition and that identity relations are neutral elements w.r.t. composition,
(3) that $QR \subseteq S$, $Q^{\mathsf{T}}\overline{S} \subseteq \overline{R}$ and $\overline{S}\,R^{\mathsf{T}} \subseteq \overline{Q}$ are equivalent, for all relations Q, R, S (with appropriate types),
(4) that $R \neq \mathsf{O}$ is equivalent to $\mathsf{L}R\mathsf{L} = \mathsf{L}$, for all relations R and all universal relations (with appropriate types).

We do not require the Boolean lattice to be complete, as in [26]. In [24] the equivalences of (3) are called the *Schröder equivalences* and direction '$\Rightarrow$' of (4) is called the *Tarski rule*. Our variant of the Tarski rule is motivated by the fact that it avoids the degenerated case of a Boolean lattice with one element only. In the relation-algebraic proofs of this paper we will mention only applications of the Schröder equivalences, the Tarski rule and 'non-obvious' consequences of the axioms. Furthermore, we will assume that complementation and transposition bind stronger than composition, composition binds stronger than union and intersection, and that all expressions and formulas are well-typed. Since types are helpful for the understanding, they frequently are presented in the text surrounding the corresponding formulae.

In this paper we make use of the following classes of relations. A relation R is *univalent* if $R^{\mathsf{T}}R \subseteq \mathsf{I}$ and *total* if $R\mathsf{L} = \mathsf{L}$. As usual, a univalent and total relation is a *function*. A relation R is *injective* if R^{T} is univalent and *surjective* if R^{T} is total. Finally, a relation R is *irreflexive* if $R \subseteq \overline{\mathsf{I}}$ and *symmetric* if $R = R^{\mathsf{T}}$. In case of set-theoretic relations the equivalence of these relation-algebraic specifications and the common logical specifications can easily be derived.

Relation algebra provides different ways to model subsets and single elements of sets. In the present paper we use vectors, a special class of relations introduced in [24], and usually denoted by lower-case letters. A relation v is a *vector* if $v = v\mathsf{L}$. For a set-theoretic relation $v : X \leftrightarrow Y$ the condition $v = v\mathsf{L}$ means that v is (as set of pairs) of the specific form $V \times Y$, with a subset V of X, i.e., for all $x \in X$ and $y \in Y$ we have $(x, y) \in v$ if and only if $x \in V$. We may consider v as relational model of the subset V of its source X. For modeling an element $x \in X$ we identify the singleton set $\{x\}$ with the only element x it contains. This leads to a specific class of vectors. A *point* p is an injective and surjective vector. In the set-theoretic case and if the point $p : X \leftrightarrow Y$ is of the specific form $p = P \times Y$ with $P \subseteq X$, then injectivity of p means that P contains at most one element and surjectivity of p means that P contains at least one element. Next, we prove properties of points which are consequences of our variant of the Tarski rule.

Lemma 2.1 *If p is a point, then we have $p \neq \mathsf{O}$, and if p and q are points, then we have $pq^{\mathsf{T}} \neq \mathsf{O}$.*

Proof. Using the Tarski rule and that the point p is a surjective vector, we get

$$p \neq \mathsf{O} \iff \mathsf{L}p\mathsf{L} = \mathsf{L} \iff \mathsf{L}p = \mathsf{L} \iff \mathsf{L} = \mathsf{L}\,,$$

that is, the first claim, and using the Tarski rule twice, surjectivity of p and non-emptiness of points (i.e., the first claim), the second claim follows from

$$pq^\mathsf{T} \neq \mathsf{O} \iff \mathsf{L}pq^\mathsf{T}\mathsf{L} = \mathsf{L} \iff \mathsf{L}q^\mathsf{T}\mathsf{L} = \mathsf{L} \iff q^\mathsf{T} \neq \mathsf{O} \iff q \neq \mathsf{O}. \qquad \square$$

In the context of algorithms the choice of an element from a non-empty set is frequently used. In the same way the choice of a point from a non-empty vector is fundamental for relational programming. Therefore, we assume a corresponding operation *point* to be at hand – as in the programming language of RelView; see [30] – such that $\mathit{point}(v)$ is a point and $\mathit{point}(v) \subseteq v$, for all non-empty vectors v. Note that *point* is a (deterministic) operation in the usual mathematical sense, such that each call $\mathit{point}(v)$ yields the same point in v. However, the above requirements allow different realizations. The specific implementation of *point* in RelView uses the fact that RelView deals only with relations on finite sets, which are linearly ordered by an internal enumeration. A call $\mathit{point}(v)$ then chooses that point which describes the least element of the set described by v.

3 A Relational Program for Vertex Coloring

Graph coloring in general and vertex coloring in particular is one of the most important and most studied concepts in graph theory. It leads to many interesting applications in mathematics and computer science, e.g., in the construction of timetables. In this section we develop a relational program to compute a vertex coloring of a given undirected graph, i.e., a labeling of the vertices with colors such that two adjacent vertices are labeled with different colors.

Assume G to be an undirected (loop-free) graph with vertex set X. We model G by the *adjacency relation* $E : X \leftrightarrow X$ such that for all $x, y \in X$ it holds $(x, y) \in E$ if and only if x and y are adjacent. Since G is assumed to be undirected (and loop-free), E is symmetric and irreflexive. E is the input of the relational program we want to develop and to prove as correct. Since we tent to a while-program and the use of the inductive assertion method, this leads to

$$\mathrm{Pre}(E) \;:\Longleftrightarrow\; E = E^\mathsf{T} \wedge E \subseteq \overline{\mathsf{I}} \qquad \text{(Pre)}$$

as pre-condition. The output of our relational program should be a vertex coloring of G. Usually natural numbers are taken as colors and, thus, a vertex coloring of G would be a function $C : X \to \mathbb{N}$ such that $C(x) = C(y)$ implies $(x, y) \notin E$, for all $x, y \in X$. Functions are specific relations and so vertex colorings are relations as well. We want to stay as abstract as possible and do not want to use natural numbers as colors, but elements of an abstract set F of colors. As a consequence, a vertex coloring of G is a relation $C : X \leftrightarrow F$ that is univalent, total, and for all $x, y \in X$ if there exists $f \in F$ such that $(x, f) \in C$ and $(y, f) \in C$ this

implies $(x, y) \in \overline{E}$. It is easy to show that the third requirement is equivalent to $CC^\mathsf{T} \subseteq \overline{E}$. This yields

$$\mathrm{Post}(C, E) \;:\Longleftrightarrow\; C^\mathsf{T}C \subseteq \mathsf{I} \wedge C\mathsf{L} = \mathsf{L} \wedge CC^\mathsf{T} \subseteq \overline{E} \qquad \text{(Post)}$$

as post-condition. We call the formula $CC^\mathsf{T} \subseteq \overline{E}$ of $\mathrm{Post}(C, E)$ the *coloring property* of C w.r.t. E.

To develop a relational while-program with input E and output C which is correct w.r.t. the pre-condition $\mathrm{Pre}(E)$ and the post-condition $\mathrm{Post}(C, E)$, it seems to be reasonable to follow a greedy approach. Using a loop, the program assigns to each vertex an available color that is not already used for one of its neighbors. Such an approach means that we work with *partial colorings*. Formally, that means we use

$$\mathrm{Inv}(C, E) \;:\Longleftrightarrow\; C^\mathsf{T}C \subseteq \mathsf{I} \wedge CC^\mathsf{T} \subseteq \overline{E} \qquad \text{(Inv)}$$

as loop-invariant, and want to extend C in each run through the loop by coloring an uncolored vertex with an allowed color in the above described manner until C is total. Summing up, we have

$$\{\,\mathrm{Pre}(E)\,\}\;\ldots;\;\{\,\mathrm{Inv}(C, E)\,\}\;\textbf{while}\;C\mathsf{L} \neq \mathsf{L}\;\textbf{do}\ldots\textbf{od}\;\{\,\mathrm{Post}(C, E)\,\}$$

as program outline. Because of the definition of the loop-invariant and the post-condition we immediately obtain the implication

$$\mathrm{Inv}(C, E) \wedge C\mathsf{L} = \mathsf{L} \implies \mathrm{Post}(C, E) \qquad \text{(PO1)}$$

to be valid. Hence, by (PO1) we have the first proof obligation of program verification, viz. that the loop-invariant in conjunction with the exit-condition of the loop implies the post-condition. It remains to develop an initialization that establishes the loop-invariant and a loop-body that maintains the loop-invariant as long as $C\mathsf{L} \neq \mathsf{L}$ holds. Obviously, we have:

Lemma 3.1 *The empty relation $\mathsf{O} : X \leftrightarrow F$ is univalent and fulfills the coloring property w.r.t. E.*

As an immediate consequence of this lemma we get that the implication

$$\mathrm{Pre}(E) \implies \mathrm{Inv}(\mathsf{O}, E) \qquad \text{(PO2)}$$

is valid. If we, guided by this fact, change the above program outline by concretizing the initialization to $C := \mathsf{O}$, then (PO2) is the second proof obligation of program verification and says for the new program outline that the loop-invariant is established by the initialization if the pre-condition holds.

To develop a loop-body, we use the fact that the vector $C\mathsf{L}$ models the domain of the univalent relation $C : X \leftrightarrow F$, i.e., the set of vertices of G which are already colored. If $C\mathsf{L} \neq \mathsf{L}$, then the call $\mathit{point}(\overline{C\mathsf{L}})$ selects a point, say p, with $p \subseteq \overline{C\mathsf{L}}$ that models an uncolored vertex, say $x \in X$. Guided by the above

mentioned greedy approach, we now consider the vector Ep. A little componentwise reflection shows that it models the set of neighbors of x and that the derived vector $C^\mathsf{T}Ep$ models the image of the set of neighbors of x under the univalent relation C, that is, the set of colors already assigned to a neighbor of x. As a consequence, the complemented vector $\overline{C^\mathsf{T}Ep}$ models the set of colors that are allowed to be assigned to x without contradicting the coloring property w.r.t. E. If we define a point q as $q := \mathit{point}(\overline{C^\mathsf{T}Ep})$, then q models one of these colors, say $f \in F$, and the union $C \cup pq^\mathsf{T}$ extends the relation C by additionally assigning f to x. This yields the following complete program outline:

$$\begin{array}{l} C := \mathsf{O}; \\ \textbf{while } C\mathsf{L} \neq \mathsf{L} \textbf{ do} \\ \quad \textbf{let } p = \mathit{point}(\overline{C\mathsf{L}}); \\ \quad \textbf{let } q = \mathit{point}(\overline{C^\mathsf{T}Ep}); \\ \quad C := C \cup pq^\mathsf{T} \textbf{ od} \end{array} \tag{VC}$$

To improve readability of (VC), we use two **let**-clauses for assigning the above mentioned points p and q.

We have already verified two out of the three proof obligations needed to prove partial correctness of the relational program (VC) w.r.t. the above pre- and post-condition specification. It remains to verify the third proof obligation

$$\mathrm{Inv}(C,E) \wedge C\mathsf{L} \neq \mathsf{L} \implies \mathrm{Inv}(C \cup pq^\mathsf{T}, E) \tag{PO3'}$$

for partial correctness, where p and q are defined as in the relational program (VC). In case programs do not change the input and the precondition Pre remains unchanged, the pre-condition can be added to the loop-invariant. For the relational program (VC) this is the case and hence it suffices to show

$$\mathrm{Pre}(E) \wedge \mathrm{Inv}(C,E) \wedge C\mathsf{L} \neq \mathsf{L} \implies \mathrm{Inv}(C \cup pq^\mathsf{T}, E) \tag{PO3}$$

We prove (PO3) in two steps. First, we show that enlarging a univalent relation by the product of two points as done in line 5 of the relational program (VC) yields again a univalent relation.

Lemma 3.2 *Let C, p and q be relations such that C is univalent, p and q are points, $C\mathsf{L} \neq \mathsf{L}$, and $p \subseteq \overline{C\mathsf{L}}$. Then $C \cup pq^\mathsf{T}$ is univalent.*

Proof. Because of the equation

$$(C \cup pq^\mathsf{T})^\mathsf{T}(C \cup pq^\mathsf{T}) = C^\mathsf{T}C \cup qp^\mathsf{T}C \cup C^\mathsf{T}pq^\mathsf{T} \cup qp^\mathsf{T}pq^\mathsf{T}$$

it suffices to show the following four inclusions:

$$(1)\ C^\mathsf{T}C \subseteq \mathsf{I} \quad (2)\ qp^\mathsf{T}C \subseteq \mathsf{I} \quad (3)\ C^\mathsf{T}pq^\mathsf{T} \subseteq \mathsf{I} \quad (4)\ qp^\mathsf{T}pq^\mathsf{T} \subseteq \mathsf{I}$$

Inclusion (1) holds as C is univalent. Since $qp^\mathsf{T}C = (C^\mathsf{T}pq^\mathsf{T})^\mathsf{T}$ and $\mathsf{I} = \mathsf{I}^\mathsf{T}$, inclusion (2) is equivalent to inclusion (3) and, thus, it suffices to show that one of them holds. To prove inclusion (3), we calculate

$$p \subseteq \overline{C\mathsf{L}} \iff C\mathsf{L} \subseteq \overline{p} \iff C^\mathsf{T}p \subseteq \mathsf{O}\,,$$

where we apply one of the Schröder equivalences in the second step. So, we have $C^\mathsf{T}p = \mathsf{O}$ and this implies $C^\mathsf{T}pq^\mathsf{T} = \mathsf{O} \subseteq \mathsf{I}$. Using the vector property and the injectivity of the point q, inclusion (4) is shown by

$$qp^\mathsf{T}pq^\mathsf{T} \subseteq q\mathsf{L}q^\mathsf{T} = qq^\mathsf{T} \subseteq \mathsf{I}\,. \qquad \square$$

The following lemma states the second fact we have to prove for verifying proof obligation (PO3). We show that the enlargement maintains the coloring property.

Lemma 3.3 *Let E, C, p and q be relations such that E is symmetric and irreflexive, p and q are points, C fulfills the coloring property w.r.t. E, $C^\mathsf{T}Ep \neq \mathsf{L}$, and $q \subseteq \overline{C^\mathsf{T}Ep}$. Then $C \cup pq^\mathsf{T}$ fulfills the coloring property w.r.t. E.*

Proof. We follow exactly the proof of Lemma 3.2 and start with

$$(C \cup pq^\mathsf{T})\,(C \cup pq^\mathsf{T})^\mathsf{T} = CC^\mathsf{T} \cup pq^\mathsf{T}C^\mathsf{T} \cup Cqp^\mathsf{T} \cup pq^\mathsf{T}qp^\mathsf{T},$$

such that it suffices to show the following four inclusions:

$$(1)\ CC^\mathsf{T} \subseteq \overline{E} \qquad (2)\ pq^\mathsf{T}C^\mathsf{T} \subseteq \overline{E} \qquad (3)\ Cqp^\mathsf{T} \subseteq \overline{E} \qquad (4)\ pq^\mathsf{T}qp^\mathsf{T} \subseteq \overline{E}$$

Inclusion (1) holds since it is assumed that C fulfills the coloring property. Because of $pq^\mathsf{T}C^\mathsf{T} = (Cqp^\mathsf{T})^\mathsf{T}$ and $\overline{E} = \overline{E}^\mathsf{T}$ the inclusions (2) and (3) are again equivalent. To prove inclusion (3), we calculate

$$\begin{aligned} q \subseteq \overline{C^\mathsf{T}Ep} &\iff C^\mathsf{T}Ep \subseteq \overline{q} \iff Cq \subseteq \overline{Ep} \\ &\iff Ep \subseteq \overline{Cq} \iff Cqp^\mathsf{T} \subseteq \overline{E}\,, \end{aligned}$$

where we apply the Schröder equivalences in the second and the fourth step. Using the vector property and the injectivity of the point q and the irreflexivity of E, inclusion (4) is shown by

$$pq^\mathsf{T}qp^\mathsf{T} \subseteq p\mathsf{L}p^\mathsf{T} = pp^\mathsf{T} \subseteq \mathsf{I} \subseteq \overline{E}\,. \qquad \square$$

Combining the Lemmata 3.2 and 3.3, we immediately obtain (PO3) and, thus, altogether the partial correctness of the relational program (VC) w.r.t. the pre-condition $\mathrm{Pre}(E)$ and the post-condition $\mathrm{Post}(C, E)$. Note that only for the maintenance of the coloring property the pre-condition is required.

We are not only interested in partial correctness, but also in total correctness. Therefore, it remains to prove the proof obligation

$$\mathrm{Pre}(E) \implies \text{the relational program (VC) yields a defined value}\,. \qquad \text{(PO4)}$$

To verify (PO4), we have to verify two facts: first, we have to prove that the loop of the relational program (VC) terminates, and, secondly, that the partial operation *point* is only applied to non-empty vectors (i.e., yields a defined value). The following lemma shows that the relation C is strictly enlarged in each execution of the loop-body.

Lemma 3.4 *Let C, p and q be relations such that p and q are points, $C\mathsf{L} \neq \mathsf{L}$, and $p \subseteq \overline{C\mathsf{L}}$. Then $C \subseteq C \cup pq^\mathsf{T}$ and $C \neq C \cup pq^\mathsf{T}$.*

Proof. Inclusion $C \subseteq C \cup pq^\mathsf{T}$ is trivial. Next, we show $pq^\mathsf{T} \subseteq \overline{C}$ by

$$\begin{array}{lll} Cq \subseteq C\mathsf{L} & \iff C^\mathsf{T}\overline{C\mathsf{L}} \subseteq \overline{q} & \text{Schröder equivalences} \\ & \implies C^\mathsf{T}p \subseteq \overline{q} & \text{as } p \subseteq \overline{C\mathsf{L}} \\ & \iff p^\mathsf{T}C \subseteq \overline{q^\mathsf{T}} & \\ & \iff pq^\mathsf{T} \subseteq \overline{C} & \text{Schröder equivalences}\,. \end{array}$$

Using $pq^\mathsf{T} \subseteq \overline{C}$, the second claim $C \neq C \cup pq^\mathsf{T}$ now can be shown by contradiction: $C = C \cup pq^\mathsf{T}$ would imply $pq^\mathsf{T} \subseteq C$, such that $pq^\mathsf{T} \subseteq C \cap \overline{C} = \mathsf{O}$ follows. But the latter fact contradicts Lemma 2.1. □

From this lemma we obtain that the loop of the relational program (VC) terminates if $E : X \leftrightarrow X$ is a relation on a finite set X, i.e., if the graph G is finite. However, to verify (PO4) we also have to ensure that the partial operation *point* is only applied to non-empty vectors. In case of the call $\mathit{point}(\overline{C\mathsf{L}})$ non-emptiness of $\overline{C\mathsf{L}}$ follows from the loop-condition. However, since we do not assume specific properties for the set F of colors, in case of the call $\mathit{point}(\overline{C^\mathsf{T}Ep})$ it may happen that $\overline{C^\mathsf{T}Ep}$ is empty, viz. if there are too few colors and each color is already assigned to a neighbor of the vertex modeled by the point p. This situation can not appear if there are enough colors. Obviously $|X|$ colors suffice. So, we have the following result:

Theorem 3.1 *If E is a relation on a finite set X and F consists of at least $|X|$ colors, then the relational program (VC) is totally correct w.r.t. the pre-condition* $\mathrm{Pre}(E)$ *and the post-condition* $\mathrm{Post}(C, E)$.

The assumptions of this theorem and its proof confirm again the experience we have made so far with the assertion-based verification of relational programs: algebra is an ideal base to verify the proof obligations for partial correctness, but for showing total correctness non-algebraic arguments are necessary, typically. Usually, they concern the sizes of the carrier sets of the relations in question.

In the following sections we demonstrate how the presented proofs can be automated, or at least supported by the tools mentioned in the introduction.

4 Invariant Testing using RelView

Relation algebra has a fixed and small set of constants and operations which (in the case of finite carrier sets) can be implemented very efficiently. At the University of Kiel we have developed RelView, a special purpose computer algebra system for relation algebra. It uses BDDs for implementing relations and makes full use of a graphical user interface. Details can be found in [5,30].

Translating the relational program (VC) into the programming language of RelView yields the following code:

```
color(E)
  DECL C, p, q
  BEG  C = O(E);
       WHILE -eq(C*L(C),L(C)) DO
         ASSERT(Inv, incl(C^*C,I(C)) & incl(C*C^,-E));
         p = point(-(C*L(C)));
         q = point(-(C^*E*p));
         C = C | p*q^ OD
       RETURN C
  END.
```

In this RelView-program the symbols `-`, `^`, `|`, `&` and `*` denote the operations for complementation, transposition, union, intersection and composition, respectively. Furthermore, `eq` and `incl` are base-operations for testing the equality and inclusion of relations, respectively. All tests yield relations on a specific singleton set $\mathbf{1}$ as result, where $\mathsf{L} : \mathbf{1} \leftrightarrow \mathbf{1}$ models 'true' and $\mathsf{O} : \mathbf{1} \leftrightarrow \mathbf{1}$ models 'false'. A call of the base-operation `O` generates an empty relation, with the same type as the argument. The operations `L` and `I` perform the same for the universal relation and the identity relation, respectively. Due to the initialization of the variable `C` in `color` by the empty relation of the same type as the input `E`, hence, the vertex set X of the graph G is taken as set F of colors, implicitly. As a consequence, there are enough colors and the RelView-program `color` is totally correct w.r.t. the pre-condition $\mathrm{Pre}(E)$ and the post-condition $\mathrm{Post}(C, E)$.

Within the RelView-program `color` we also use the `ASSERT`-statement for testing the loop-invariant. If the second part of `ASSERT` (a relation-algebraic formula formulated as RelView-expression) is true, then the statement is without effect, otherwise the execution stops and RelView allows us to inspect the values of the variables via the debug window. Combining the specification of the loop-invariant in the program via `ASSERT` with RelView's feature for generating relations randomly (also with specific properties like, in our case, symmetry and irreflexivity) has the general advantage that no invariant-tests have to be done by hand (which takes time and is vulnerable to mistakes) and a lot of tests can be done in a very short time. Consequently, one gets a good feeling if a loop-invariant was chosen correctly. Because of the specific modeling of truth-values in RelView, furthermore, on the two relations $\mathsf{L} : \mathbf{1} \leftrightarrow \mathbf{1}$ and $\mathsf{O} : \mathbf{1} \leftrightarrow \mathbf{1}$ the Boolean operations $\overline{}$, $\cup$ and $\cap$ precisely correspond to the logical connectives $\neg$, $\vee$ and $\wedge$, respectively. This allows to formulate all Boolean combinations over inclusions of relations as RelView-expressions and to test them via `ASSERT`. Experience has shown that this suffices for most practical applications. It also has shown that stepwise execution and visualization via RelView are frequently helpful if invariants are not correct, e.g., too weak.

5 Verification of Proof Obligations using Prover9

Along the lines of [8,9] we now show how the correctness proof of the relational program (VC) can be supported by an automated theorem prover, i.e., we au-

tomate the proofs of Section 3 as far as possible. Intending a user-optimized approach we choose Prover9 as verification tool. This choice is based on an evaluation that shows that Prover9 performs best for automated reasoning in the context of relation algebra; see [9] for details. A further reason for the choice of Prover9 is the positive experience made in [8,9] in the automated verification of relational programs with this tool.

Prover9 [20] is a resolution- and paramodulation-based automated theorem prover for first-order and equational logic. However, it does not include a type system. Of course, types can be realized using predicates. Since this is a bit cumbersome, we have decided to restrict our experiments to homogenous relation algebra in the sense of [13,26,27], with untyped relations. This algebraic structure axiomatizes the algebra of relations on one set (the universe) and its axioms are obtained from those of Section 2 if all demands concerning types are removed. A consequence of our decision is that the sets of vertices and colors coincide, as in case of the RelView-program `color` of Section 4.

For each result of Section 3 we want to prove, we create one input file. Each file consists of three parts, where the first two parts of each file coincide. The first part contains the language options, in particular the list of operations of relation algebra. We use the symbols `^`, `'`, `\/`, `/\` and `*` for transposition, complement, union, intersection and composition, respectively, with the binding strengths of Section 2. The second part is a list of assumptions and contains the axioms of homogeneous relation algebra, some auxiliary facts which turned out to be well suited for proving relation-algebraic results, and predicates for defining properties of relations. The constants `L`, `O`, `I` and the inclusion of relations are implicitly defined via the axioms, in symbols `L`, `O`, `I` and `<=`. The encoding of the axiomatization in Prover9 is straightforward. For example, the distributivity laws can be formulated as follows:

```
x /\ (y \/ z) = (x /\ y) \/ (x /\ z).
x \/ (y /\ z) = (x \/ y) /\ (x \/ z).
```

To give another example in the notation of Prover9 the Schröder equivalences look as follows:

```
x*y <= z <-> x^*z' <= y'.
x*y <= z <-> z'*y^ <= x'.
```

Although Prover9 accepts capital letters as variable names, such as `Q`, `R` and `S`, we use the small letters `x`, `y` and `z` for variables, since variables which are denoted by these letters are automatically assumed as universally quantified. The set of auxiliary facts only lists statements which are already proven by Prover9 (e.g., in [16,17]). The predicates to specify, for instance, univalent relations or relations with the coloring property can be encoded as follows:

```
univalent(x)          <-> x^*x <= I.
coloringProperty(x,z) <-> x*x^ <= z'.
```

The goal to be proven by Prover9 is specified in the third part of the file. As mentioned, we apply algebra only for proving facts, where arguments concerning sizes of sets etc. are not necessary. This means that, besides the auxiliary lemma about points of Section 2, we apply Prover9 only for proving the lemmata of

Section 3. Doing so, we use the variables `p` and `q` for the general points p and q of Lemma 2.1 as well as for the specifically selected points $p := point(\overline{C\mathsf{L}})$ and $q := point(\overline{C^\mathsf{T} E p})$ of Section 3 and the (again automatically universally quantified) variables `x` for the relation C and `z` for the relation E, respectively. Then, e.g., the statement of Lemma 3.3 can be encoded as follows:

```
all p all q (symmetric(z) & irreflexive(z) & point(p) & point(q) &
             coloringProperty(x,z) & (x^*z)*p != L & q <= ((x^*z)*p)'
             -> coloringProperty(x \/ p*q^,z)).
```

Prover9 has no problems to derive a proof of Lemma 2.1 and requires only a couple of milliseconds. For the input files for the Lemmata 3.1, 3.2 and 3.4 Prover9 generates output files containing their proofs instantaneously as well. However, in case of Lemma 3.3 Prover9 is not able to find a proof in an appropriate time (we stopped the execution after one hour). Guided by our experience gained by previous case studies we know that in such a situation the unfolding of definitions, the subdivision of the entire task into appropriate subtasks and the removal of laws may help, since these steps reduce the size of the search space, frequently even dramatically. In the present case replacing `coloringProperty(x\/p*q^,z)` by its definition is not sufficient. Also the removal of formulae seems not to be helpful. If, however, the proof of Lemma 3.3 is divided into, first, showing that its conclusion is equivalent to the conjunction of the inclusions (1) to (4) of its proof and, secondly, that from its assumptions this conjunction follows, then for each of these tasks Prover9 needs again no time.

If Prover9 fails to find a proof, besides the unfolding of definitions, the manual change of the goal and the removal of axioms or auxiliary facts, one can use that the tool allows a weighting of formulae to specify on them an order of significance in view of the present problem. Because of our experiments with different weightings, in the case of Lemma 3.3 we believe that also a weighting of formulae does not lead to a proof in an appropriate time.

Summing up, Prover9 was able to prove the desired results and requires in one case a small user interaction only. As all automatic theorem provers, if the goals are appropriately formulated, then no interaction is needed and, hence, no deeper knowledge about (relation-)algebraic reasoning is required from the user.

6 Verification of Proof Obligations using Coq

Now we change the paradigm from 'automated' to 'user-controlled' and demonstrate how to verify Lemma 2.1 and the lemmata of Section 3 by use of the proof assistant Coq. More information about this tool can be found in [11,29].

Since the functionality of Coq is based on the *predicative calculus of inductive constructions*, each object has a type. Thus, heterogeneous relation algebra can be modeled, and it has already been done within a relation-algebra library, presented in [22] and available via the web (see [23]). This library does not only include a model for heterogeneous relation algebra but also for a large number of other algebraic structures. For this purpose, sets of operations and laws are

provided, mainly in the modules `lattice` (for lattice theory), `monoid` (for pre-ordered monoids) and `kat` (for Kleene algebra with tests). The dependencies of the structures w.r.t. the operations and laws are managed in the module `level`, i.e., one can choose which kind of structure should be used by providing the required operations and laws.

Since we want to derive the proofs of Lemma 2.1 and those of Section 3, we assume a heterogeneous relation algebra, where the constants, operations, predicates and laws are defined in the mentioned modules. At the moment, the Axioms (3) and (4), i.e., the Schröder equivalences and the Tarski rule, are not formulated in the library of [22], yet.

The Schröder equivalences can be derived via the two so-called modular laws of a Dedekind category, that is, via the law named `capdotx` and its dual law `capxdot`, which are defined in the module `monoid.v`. One of the Schröder equivalences can be encoded as follows:

```
Lemma schroe1 `{laws} `{BL+STR+CNV<<l} n m p(Q:X n m)(R:X m p)(S:X n p):
  Q*R <== S <-> Q`*!S <== !R.
```

With `` `{laws} `` and `` `{BL+STR+CNV<<l} ``, respectively, we provide the operations and axioms of relation algebra. The symbols `` ` ``, `!`, `+`, `^` and `*` are used for transposition, complement, union, intersection and composition, respectively.

The missing Tarski rule has to be added since it is necessary for the proofs of Lemma 2.1 and (implicitly) of Lemma 3.4. In contrast to the Schröder equivalences, the Tarski rule is not a consequence of the given laws, i.e., we have to provide it as an additional axiom. In Section 2, we specify the Tarski rule by the equivalence of $R \neq \mathsf{O}$ and $\mathsf{L}R\mathsf{L} = \mathsf{L}$ for all relations R and universal relations with appropriate types. Of course, constants are typed objects in Coq, too. But, if Coq can infer the type from the context, then it is not necessary to specify it. In such a case the universal relation, empty relation and identity relation are denoted with `top`, `0` and `1`, respectively. In case of non-inferable types we have to specify them by, for instance, `top' X Y` for the universal relation $\mathsf{L} : X \leftrightarrow Y$. We have a universal quantification over the three occurring universal relations in the Tarski rule. For its formulation within Coq, besides the type of R we have to specify the types of the two universal relations of the left-hand side of $\mathsf{L}R\mathsf{L} = \mathsf{L}$ only. The type of O in $R \neq \mathsf{O}$ and the type of the right-hand side of $\mathsf{L}R\mathsf{L} = \mathsf{L}$ then can be inferred from the context. Considering this typing and using a Coq-definition, the Tarski rule can be encoded as follows:

```
Definition Tarski_rule `{laws} : Prop :=
(forall a b c d (R:X b c),(top' a b)*R*(top' c d) == top <-> ~(R == 0)).
```

Note that we omit the assumption `` `{BL+STR+CNV<<l} `` about the level, because such definitions can be written without having a structure satisfying any law. We assume the definition `Tarski_rule` in each lemma whose relation-algebraic proof uses the Tarski rule. As already mentioned, this concerns Lemma 2.1 of Section 2 and Lemma 3.4 of Section 3.

As in the previous section, we define predicates specifying, e.g., relations as points or relations with the coloring property. In Coq, this can be done as follows:

```
Definition coloringProperty `{laws} {n} {m}: X n m -> X m m -> Prop :=
```

```
  fun x y => x*x‘<== !y.
Definition point ‘{laws} {n} {m}: X n m -> Prop :=
  fun p => vector p /\ p*p‘ <== 1 /\ (forall a, top’ a m == top*p).
```

Here `vector` is yet another predicate for describing vectors. Such predicates improve the readability of the encoding.

Using the defined predicates and the definition specifying the Tarski rule, the first statement of Lemma 2.1 can be encoded as follows:

```
Lemma lemma_2_1_1 ‘{laws} ‘{BL+STR+CNV<<l} m n:
tarski_rule -> forall (p:X m n), point p -> ~(p == 0).
```

Its second statement and the lemmata of Section 3 can be formulated in a similar way. For example, the Coq-version of Lemma 3.3 is given below; it looks rather similar to the version in Prover9 with typed relations though (note that the conjunction symbol `/\` of Coq corresponds to the symbol `&` in Prover9):

```
Lemma lem3_3 ‘{laws} ‘{BL+STR+CNV<<l} v f (C:X v f)(E p:X v v)(q:X f v):
  symmetric E /\ irreflexive E /\ point p /\ point q /\
  coloringProperty C E /\ ~(C‘*E*p == top) /\ q <== !(C‘*E*p)
  -> coloringProperty (C + p*q‘) E.
```

Usually, the development of proofs in Coq is done via various tactics. The proofs of the mentioned lemmata, apart from Lemma 3.4, can be managed with only a few basic tactics, such as `intro` for introducing new variables or hypotheses, `unfold` for unfolding upcoming predicates in the goal as well as in the hypotheses and `rewrite` for replacing terms. More interesting are the tactics defined in the module `normalisation.v` of the library described in [22]. This module includes three specific tactics called `ra`, `ra_simpl` and `ra_normalise` which can be used to automate parts of the proofs, for instance in case of universally quantified inclusions and equalities. The proof of Lemma 3.4 has to be handled in a different way since the occurring negated equality. For this purpose, we slightly change the proof presented in Section 3 and import a module for classical propositional logic, viz. the module `Coq.Logic.Classical_Prop`, to provide the required De Morgan's laws. The advantage of this approach is that we are able to prove the lemma with the already mentioned tactics, i.e., we avoid to deal with contradiction in Coq.

In summary, Coq offers a type system and allows to model heterogeneous relation algebra in a very natural way. Amongst others, the library we have used offers a typed model for heterogeneous relations, a large number of already proven algebraic theorems and very helpful tactics for reasoning about relation algebra.

7 Verification of Proof Obligations using Isabelle/HOL

In this section we discuss the verification of the relational program (VC) by means of Isabelle/HOL, a proof assistant that additionally offers support for automated theorem proving via the Sledgehammer tool. For more details on Isabelle/HOL, see e.g., [21].

Similar to Coq, libraries can be included in Isabelle/HOL. The development of such libraries usually takes a long time and deep insights in the theorem prover at hand. Luckily, as in the case of Coq, relation algebra has already be formalized in Isabelle/HOL and is available via the web (see [1]). However, the library of [1] formalizes homogeneous relation algebra only. A consequence of its use with regard to the verification of the relational program (VC) is again that the sets of vertices and colors coincide.

The formalization [1] of homogeneous relation algebra follows the lines of [19,26]. Besides the basic constants, operations and predicates and the axioms it includes a number of further important relation-algebraic concepts such as subidentities, vectors and points, as well as various notions associated to functions – together with numerous proven facts. For example, all facts about relation algebra listed in Section 2 have been proven. As a consequence, it seems to be an ideal basis for the verification of relational programs such as (VC). However, the current implementation does not contain the Tarski-rule (similar to Coq), so we added this rule to the set of assumptions of a lemma when necessary.

The provided libraries are included by a simple `imports`-statement; encoding of the lemmata is easy and straightforward. For example, Lemma 3.3 is encoded as follows, where $^\smile$ indicates transposition and the symbol `-` is used for both negation and complement:

```
lemma assumes "symmetric e"            and "irreflexive e"
          and "is_point p"             and "is_point q"
          and "coloringProperty x e"   and "q ≤ -(x⌣ ; e ; p)"
        shows "coloringProperty (x + p;q⌣) e"
```

It might be confusing that we use different symbols for the same operation (e.g., $^\mathsf{T}$, `^`, `‘` and $^\smile$ for transposition). However, since we use different tools we decided to stick to the notation of these frameworks. Since the GUI of Isabelle/HOL allows non-ascii symbols, transposition can be encoded as $^\smile$.

The used predicates are basically identical to the ones of Prover9 and Coq. For example, `coloringProperty` is defined as follows:

```
definition coloringProperty
     where "coloringProperty x e ≡ x ; x⌣ ≤ -e"
```

A straightforward approach would be the use of the Isabelle/Isar tool (see [28]), which basically replays the proofs given in Section 3. The advantage of this approach is that it provides a proof certificate and verifies the manual proofs. Moreover the generated proofs are easy to read. However, this strategy does not provide (much) automation and hence requires expert knowledge in relation-algebraic reasoning.

As mentioned in Section 5, automated reasoning within the relation-algebraic setting is successful if the proof-goals are appropriately formulated. In contrast to Coq, Isabelle/HOL offers support for (first-order) automated theorem provers via the integrated tool Sledgehammer (see [12] for more details), thus, allows to combine the ‘automated’ and ‘user-controlled’ paradigm. The Sledgehammer tool takes the given goal and proven facts available, feeds them to automated theorem provers, such as E and Z3, and awaits their output. In case one of the

provers is successful in finding a proof, the proof is included in the Isabelle-file; in case all theorem provers fail, the GUI continues to assist in a manual proof derivation. That means that Isabelle/HOL provides both proof-assistance and proof-automation and it seems to be the perfect combination of interaction and automation. In fact, a proof of Lemma 3.3 becomes 'nearly' automatic: after a first manual step using Isabelle's unfolding and simplification mechanisms (`simp add: unfold_defs distrib_left distrib_right, safe`) we end up with the four subgoals (1) to (4) presented in the proof of Lemma 3.3. This command unfolds automatically all predicates (`unfold_defs`) and uses the built-in simplifier, which is manually extended by the two distributivity axioms of relation algebra (`distrib_left distrib_right`).

The derived subgoals can now all be proven automatically by Sledgehammer using not only the axioms of relation algebra, but also the facts provided by the theories of relation algebra of [1]. The fact `"x;y ≤ z ⇔ y ≤ -(x⌣;-z)"`, for example, which was proven in the framework of [1], is automatically chosen to prove the second subgoal `(p;q⌣;x⌣≤ -e)` (in the proof $pq^{\mathsf{T}}C^{\mathsf{T}} \subseteq \overline{E}$). All other lemmata presented in the paper, except Lemma 3.4, can be proven in an identical way: first derive subgoals using the inbuilt simplifier, and then use Sledgehammer and the provided automated theorem provers to prove these subgoals automatically. Lemma 3.4 could not be proven by this strategy. In fact, we could only reply the proof by contradiction given in Section 3 – real proof automation was not possible.

8 Assessment and Concluding Remarks

In this paper we have developed a relational program for calculating a vertex coloring of an undirected graph, which is modeled by the adjacency relation. A relation-algebraic approach was chosen since case studies have shown that such an approach is not only very suitable for prototyping and testing programs by systems like RelView, but also for proof automation. The program verification was performed by classical reasoning about pre- and post-conditions, and loop-invariants. The proofs of the proof obligations were executed with the help of Prover9, Coq and Isabelle/HOL, which are prominent tools to support verification tasks. This repetition of mechanized proofs and the comparison with the original mathematical proofs allow us to compare these mentioned tools. As one might expect each and every tool has its pros and cons.

Prover9 does not include a type system such that the typing of relations in heterogeneous relation algebra would have to be realized by predicates. Such predicates make the encoding more complicated and decrease the readability. In our example types could be avoided, but if, e.g., incidence relations are used to model undirected graphs or hypergraphs, then types are mandatory. All but one theorems of Section 3 were proved full automatically by Prover9 in nearly no time. Unfortunately, in spite of weighting the significant rules such that they should be applied first, Prover9 was still not able to find a proof for Lemma 3.3 in an appropriate time frame. At the end, we decided not to change the weights

of the assumptions but to split the goal into subgoals. Indeed, this approach requires a kind of interaction by the user, but it yields to uniform assumptions for all theorems of Section 3 as well as to short proving times. These results are not only based on the investigations presented in this paper but additionally coincide with those discussed in [8,9,14]. Since Prover9 is fully automatic it can be used as a black box and without having a deep understanding of its functionality. From a user's point of view a big advantage of Prover9 is that the encoding of the axiomatization of homogenous relation algebra, the definition of predicates and the formulation of theorems which have to be proved is very straightforward and also comprehensible for non-experts.

The proof assistant Coq is the very opposite of Prover9. It is completely user-controlled, i.e., a purely interactive theorem prover. Coq has a sophisticated type system with type inference, which is comparable with those of functional programming languages such as Haskell, ML and OCaml. For performing our proofs we used an already existing library for relation algebra. In this case the library implements homogenous as well as heterogeneous relation algebra. Due to this we were able to reproduce all proofs of Section 3 without any restrictions on types. The used library does also include tactics, that is, strategies for proof-finding and proof execution. They support relation-algebraic reasoning. Some of them implemented decision procedures for subsets of relation algebra, which we might use in future experiments. Furthermore, the library comes with a large number of algebraic structures related to relation algebras, e.g., Dedekind categories, Kleene algebras and Kleene algebras with tests. For this reason, the library can also be used in the context of reasoning about such structures as well. We refer to [10] for an application concerning Dedekind categories. The usage of Coq requires a lot of knowledge about the internals of the tool, such as the available tactics and the hierarchy and dependencies of the modules. Besides this, in our case the user needs expertise about relation-algebraic reasoning and the used library for relation algebra to be able to derive the proofs step by step. So, from a user's point of view Coq is far more complicated than Prover9; it is suitable for advanced users only.

For the proof verification with Isabelle/HOL we were also able to build on already existing theories. Since a library for heterogeneous relation algebra does not exist yet, we used a library that implements homogenous relation algebra. Using this library we again have to avoid typed relations, although Isabelle/HOL provides a type system similar to that of Coq. Concerning proof paradigms, Isabelle/HOL bridges the gap between interactive and automated reasoning via the Sledgehammer tool. Our experiments have shown that our strategy, i.e., first using the inbuilt simplifier for deriving subgoals and then Sledgehammer to prove these subgoals automatically, was successful with all theorems of Section 3 except Lemma 3.3, again. For the latter one, we have to derive new subgoals with two manual steps. However, more intrinsic proofs (here a proof by contradiction) requires again expert knowledge in relation-algebraic reasoning. Isabelle/HOL does not offer tactics nor decision procedures, yet, for relation-algebraic reasoning specifically. With regard to usage, Isabelle/HOL is powerful enough to

support non-expert users with many (standard) tasks in case the problems in question are not very complex. But in case of more complex problems it requires experience and is then, like Coq, suitable for advanced users only.

As future work we plan to exhaust the capabilities of Prover9 w.r.t. the different options, e.g., the weighting of the given assumptions, to hopefully achieve best proving times for relation-algebraic theorems. Concerning Coq, we plan to explore the full power of the tactics and to investigate whether they are supportive in verification tasks. For Isabelle/HOL, we want to consider the implementation of tactics which are specific for relation-algebraic reasoning. Furthermore, an extension of the used library to heterogeneous relation algebra is desirable. We assume that by all this many verification tasks concerning programs on relations or related objects can be automated to a large extent – this can, however, only be verified by further and more complicated case studies. Finally, we plan to investigate in the future how tools for generating loop invariants (as Why3, see [31]) are applicable for our purposes.

Acknowledgement. We thank D. Pous for valuable remarks. We also thank the unknown referees for their constructive criticims and suggestions which helped to improve the paper. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. A. Armstrong, S. Foster, G. Struth, T. Weber: Relation algebra. Archive of Formal Proofs, 2014. `http://afp.sf.net/entries/Relation_Algebra.shtml`
2. R. Berghammer: Combining relational calculus and the Dijkstra-Gries method for deriving relational programs. Inform. Sci. 119, 155-171 (1999)
3. R. Berghammer, T. Hoffmann: Deriving relational programs for computing kernels by reconstructing a proof of Richardson's theorem. Sci. Comput, Prog. 38, 1-25 (2000)
4. R. Berghammer, T. Hoffmann: Relational depth-first-search with applications. Inform. Sci. 139, 167-186 (2001)
5. R. Berghammer, F. Neumann: RelView – An OBDD-based Computer Algebra system for relations. In: V.G. Gansha, E.W. Mayr, E. Vorozhtsov (eds.): Computer Algebra in Scientific Computing, LNCS 3718, Springer, 40-51 (2005)
6. R. Berghammer: Applying relation algebra and RelView to solve problems on orders and lattices. Acta Inform. 45, 211-236 (2008)
7. R. Berghammer, M. Winter: Embedding mappings and splittings with applications. Acta Inform. 47, 77-110 (2010)
8. R. Berghammer, G. Struth: On automated program construction and verification. In: C. Bolduc, J. Desharnais, B. Ktari (eds.): Mathematics of Program Construction. LNCS 6120, Springer, 22-41 (2010)
9. R. Berghammer, P. Höfner, I. Stucke: Automated verification of relational while-programs. In: P. Höfner, P. Jipsen, W. Kahl, M. Müller (eds.): Relational and Algebraic Methods in Computer Science, LNCS 8428, Springer, 309-326 (2014)
10. R. Berghammer, I. Stucke, M. Winter: Investigating and computing bipartitions with algebraic means. In: W. Kahl, J.N. Oliviera, M. Winter (eds.): Relational and Algebraic Methods in Computer Science (to appear)

11. Y. Bertot, P. Casteran: Interactive theorem proving and program development. Coq'Art: The calculus of inductive constructions. Texts in Theoretical Computer Science, Springer (2004)
12. J.C. Blanchette, S. Böhme, L.C. Paulson: Extending Sledgehammer with SMT solvers. In: N. Bjørner, V. Sofronie-Stokkermans (eds.): Automated Deduction – CADE 23, LNCS 6803, Springer, 116-130 (2011)
13. L.H. Chin, A. Tarski: Distributive and modular laws in the arithmetic of relation algebras. Univ. of California Publ. Math. (new series) 1, 341-384 (1951)
14. H.H. Dang, P. Höfner: First-order theorem prover evaluation w.r.t. relation- and Kleene algebra. In: R. Berghammer, B. Möller, G. Struth (eds.): Relations and Kleene Algebra in Computer Science – Ph.D. Programme at RelMiCS 10/AKA 05. Technical Report 2008-04, Institut für Informatik, Universität Augsburg, 48-52, (2008)
15. S. Foster, G. Struth, T. Weber: Automated engineering of relational and algebraic methods in Isabelle/HOL (Invited tutorial) In: H. de Swart (ed.) Relational and Algebraic Methods in Computer Science. LNCS 6663, Springer, 52-67 (2011)
16. P. Höfner, G. Struth: Automated reasoning in Kleene algebra. In: F. Pfenning (ed.): Automated Deduction – CADE 21, LNAI 4603, Springer, 279-294 (2007)
17. P. Höfner, G. Struth: On automating the calculus of relations. In: A. Armando, P. Baumgartner, G. Dowek (eds.): Automated Reasoning. LNAI 5195, Springer, 50-66 (2008)
18. W. Kahl: Calculational relation-algebraic proofs in Isabelle/Isar. In: R. Berghammer, B. Möller, G. Struth (eds.): Relational and Kleene-Algebraic Methods in Computer Science. LNCS 3051, Springer, 179-190 (2004)
19. R. Maddux: Relation algebras. Studies in Logic and the Foundations of Mathematics 150, Elsevier (2006)
20. W.W. McCune: Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9`
21. T. Nipkow, L.C. Paulson, M. Wenzel: Isabelle/HOL: A proof assistant for higher-order logic. LNCS 2283, Springer (2002)
22. D. Pous: Kleene algebra with tests and Coq tools for while programs. In: S. Blazy, C. Paulin-Mohring, D. Pichardie (eds.): Interactive Theorem Proving. LNCS 7998, Springer, 180-196 (2013)
23. D. Pous: Relation algebra and KAT in Coq. `http://perso.ens-lyon.fr/damien.pous/ra/`
24. G. Schmidt, T. Ströhlein: Relations and graphs, Discrete mathematics for computer scientists, EATCS Monographs on Theoretical Computer Science, Springer (1993)
25. G. Schmidt: Relational mathematics. Encyclopedia of Mathematics and its Applications, Cambridge University Press (2010)
26. A. Tarski: On the calculus of relations. J. Symb. Logic 6(3), 73-89 (1941)
27. A. Tarski, S. Givant: A formalization of set theory without variables. AMS Colloquium Publications, American Mathematical Society (1987)
28. M. Wenzel: Isabelle/Isar – a versatile environment for human-readable formal proof documents. Dissertation, Technische Universität München (2002)
29. Coq-homepage: `https://coq.inria.fr`
30. RelView-homepage: `http://www.informatik.uni-kiel.de/~progsys/relview/`
31. Why3-homepage: `http://why3.lri.fr/`
32. Input files and proof scripts: `http://www.hoefner-online.de/ramics15/`