Program Verification With RELVIEW

Michael Winter

Fakultät für Informatik
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, D–85577 Neubiberg, Germany

Abstract

In this paper we show how the relation manipulating tool RELVIEW can be used for the verification of finite state programs. Based on the paradigm "programs as relations" we use the relational formulation of program semantics to automatically derive correctness properties of a sample program.

1 Introduction

The RELVIEW system is an interactive tool for computer-supported manipulation of finite relations. It consists of a graphical interface and an user definable database of relational operators.

During the last years, relation algebra has widely been used as a convenient formalism for describing semantics of programs. In this setting a program is considered as relations between input and output data. This gives us the possibility to prove program correctness in a comprehensive way within the relational calculus.

In this paper we show how RELVIEW can be used in the verification of programs with a finite set of states. Given a program together with an in- and output specification as a finite relation, the system will compute its correctness automatically.

The rest of the paper is organized as follows. In Chapter 2 we recall some fundamental concepts of relation algebras. Chapter 3 contains a brief description of the present version of RELVIEW. By looking at an example in RELVIEW we study program correctness and verification in Chapter 4.

2 Relation Algebraic Preliminaries

In the following we recall some fundamental concepts of abstract relation algebra. Further details can be found in [Schmidt Ströhlein 89].

Definition 2.1 *A (heterogeneous) relation algebra* $(\mathcal{R}, \sqcup, \sqcap, \overline{}, ^{\mathsf{T}})$ *is a self-dual category* $\mathcal{R}$ *(dualization is denoted by* $.^{\mathsf{T}}$*) such that the following holds:*

1. *Every homset* $\mathcal{R}[A, B]$ *together with* $\sqcup, \sqcap$ *and* $\overline{}$ *is a complete atomistic Boolean algebra with null (or least) element* 0*, universal (or greatest) element* L *and ordering* $\subset$*.*
2. *The Schröder equivalences*

$$QR \subset S \iff Q^{\mathsf{T}}\, \overline{S} \subset \overline{R} \iff \overline{S}\, R^{\mathsf{T}} \subset \overline{Q}$$

are valid whenever these relations are composable.

3. *For every relation the Tarski rule* $R \neq 0 \Leftrightarrow LRL = L$ *is valid.* □

According to the above definition, the elements of an abstract relation algebra are not nessecarily relations between sets. In the case of concrete relations, which are used within the RELVIEW system, we consider a relation as a Boolean matrix, which can easily be used for the graphical representation of a relation.

Of special interest are those relations, which are (partial) functions or mappings.

Definition 2.2 *A relation* R *is called*

R *univalent*	$:\iff$	$R^{\mathsf{T}}R \subset I$,
also: a (partial) function		
R *total*	$:\iff$	$RL = L \Leftrightarrow I \subset RR^{\mathsf{T}}$,
R *mapping*	$:\iff$	R *univalent and total,*
R *injective*	$:\iff$	R^{T} *univalent,*
R *surjective*	$:\iff$	R^{T} *total,*
R *bijective*	$:\iff$	R *injective and surjective.* □

A special class of relations can be seen as predicates.

Definition 2.3 *A relation* x *with* $x = xL$ *is called a predicate.* □

Later on, we want to conceive a program as a pair of graphs: the underlying flowchart and the situation graph.

Definition 2.4 *A pair* $G = (A, E)$ *of relations is called a (total) graph, if*

1. A, E *are mappings, called the outgoing incidence and ingoing incidence,*
2. $A \sqcup E$ *exists.*

The relation $B := A^{\mathsf{T}}E$ *is called the associated relation of* G*.* □

Termination of a program can be seen as a property of the underlying flowchart. If it is possible to proceed only finitely many steps along this flowchart, the program will terminate.

Definition 2.5 *Let G be a graph with associated relation B and consider predicates x. The set of all points, from which only paths of finite length emerge,*

$$J(B) := \inf\{x \mid \overline{x} \subset B\,\overline{x}\}$$

is called the initial part of G. □

In the case of bounded nondeterminism, which implies finite successor degree in all points of the underlying flowchart, we are able to compute the initial part by iteration.

Theorem 2.6 *Let G be a graph with associated relation B such that all points have finite successor degree. Then the following holds:*

$$\sup_{h \geq 0} \overline{B^h L} = J(B). \qquad \square$$

To relate the underlying flowchart and the situation graph of a program, we need a (graph) homomorphism.

Definition 2.7 *A relation Θ is called a (graph) homomorphism from the graph $G = (A, E)$ into the graph $G' = (A', E')$, if*

1. *Θ is a mapping,*
2. *$B \subset \Theta B' \Theta^{\mathsf{T}}$ holds.* □

3 The RELVIEW System

This section contains a brief description of the present version of the RELVIEW system. For further details we refer to [Abold-Thalmann et al. 89], [Landgraf Dombetzki 91] and [Herrmann et al. 93].

People working in or studying the theory of relations or graph theory very often use more or less small examples of "concrete" relations and manipulate them with pencil and paper in order to prove or disprove some property. The RELVIEW system provides machine support for the manipulation of concrete relations. Also the system allows the user to work with relational descriptions of domains. For further details see [Berghammer et al. 89] and [Zierer 91].

The RELVIEW system is a totally interactive and completely video-oriented computer system for the manipulation of concrete relations. The main components of the RELVIEW code are window handling, transition of the screen pictures into the logical structure of a relation and vice versa, implementation of operations on relations, I/O commands, and mangagement of relations and relational domains. The system is written in C and available for the X Window System.

The screen of the RELVIEW system looks as follows:

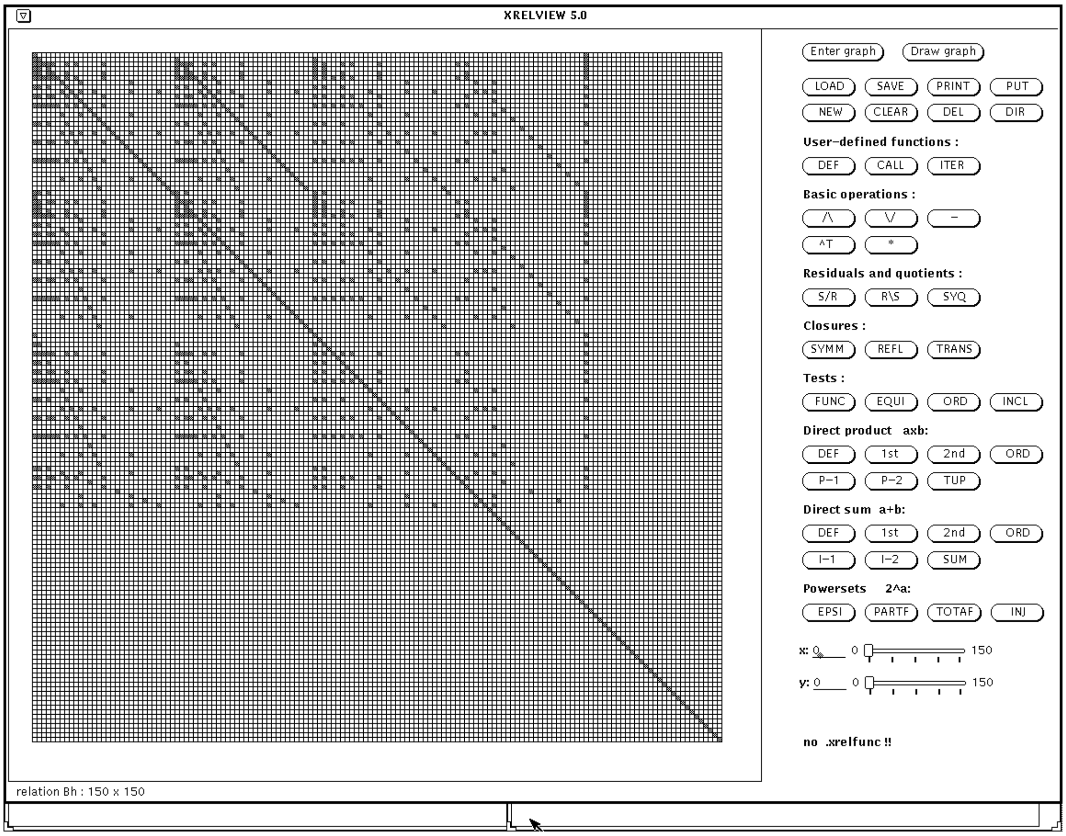

Figure 1: The RELVIEW-System

The left part of the screen is the drawing window. Using the mouse pointer, here the user can edit relations as Boolean matrices. One relation, the so-called working copy, is displayed on the screen for editing. A whole collection of relations can be kept in the working memory during a working session. Such a collection may also be saved on permanent memory, e.g. on a hard disk.

The right part of the screen contains the comand buttons and the scrollbars. The scrollbars can be used for showing a part of a relation the size of which exceeds the maximal window size.

Execution of a command consists of the choice of the respective command button, and, if necessary, the input of the arguments. The command buttons of row 1 give another possibility to enter and to visualize a relation. The "enter graph"- and the "draw graph"-button allow the user to create a relation via a directed graph and to show up every relation as such a graph. These functions are very helpful working e.g. with orderings. The buttons of row 2 and 3 serve only for data management purposes. In the next row one finds the command buttons for the definition and application of user-declared functions on relations. The remaining rows contain the command buttons for the operations and tests on relations. Row 5 and 6 contain the buttons for the basic operations on relations. In row 7 we have the command buttons for the two residuals and the symmetric quotient. The command buttons for computing closures are listed in row 8. Row 9 contains the buttons for certain tests. And, finally, the rows 10 through 14 contain the buttons which correspond to a relational description of domains. The buttons of row 10 and 11 are for the direct product, those of row 12 and 13 for the direct sum, and those of row 14 are devoted to powersets and function domains.

The Boolean matrices appearing in this paper are directly taken from the system via the "print"-button. In our example we need only a few commands, e.g. transitive closure, basic operations and inclusion test. There are many other applications for the RELVIEW system which make also use of the remaining features. For further details we refer to [Berghammer 92] and [Berghammer et al. 93].

4 Program Correctness and Verification

In this chapter we discuss partial and total correctness of a program. All concepts are shown by a finite example computed by RELVIEW. To keep it readable, this example is rather small. All proofs of theorems in this chapter can be found in [Schmidt Ströhlein 89].

Definition 4.1 *A quintuple (G, S, Θ, e, a) is called a program, if*

1. *$S = (A, E)$ is a graph, called the situation graph,*
2. *$G = (A_G, E_G)$ is a graph, called the underlying flowgraph,*
3. *Θ is a surjective homomorphism of S onto G.*
4. *e is a so-called input relation, satisfying:*

$$e^{\mathsf{T}}e = I,\ ee^{\mathsf{T}} \subset I,\ eL = \Theta\Theta^{\mathsf{T}}eL,\ \Theta^{\mathsf{T}}e = \Theta^{\mathsf{T}}eL.$$

5. *a is a so-called output relation, satisfying:*

$$a^{\mathsf{T}}a = I,\ aa^{\mathsf{T}} \sqcap \Theta^{\mathsf{T}}\Theta \subset I,\ aL = \Theta\Theta^{\mathsf{T}}aL.$$

If the associated relation B (of S) is univalent, we call $\mathcal{P}$ deterministic. □

We want to explain this definition by an example.

Example Consider the following program.

```
while x>1 do begin
  if even(x) then x := x div 2
             else x := 3*x+1;
end;
```

This program leads to the following graphs.

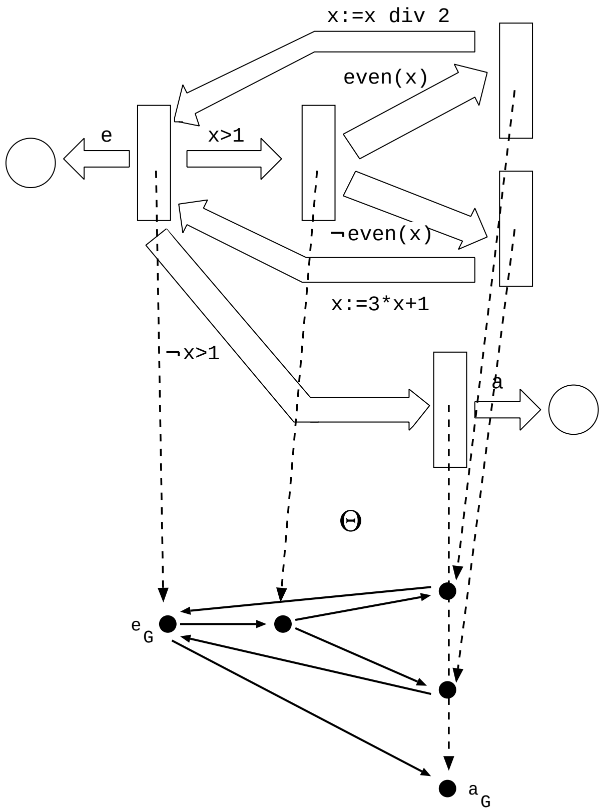

Figure 2: Situation graph and underlying flowchart

If we assume x to range over the finite subset $\{1, \ldots, 30\}$ of the natural numbers, we get the following relations in RELVIEW.

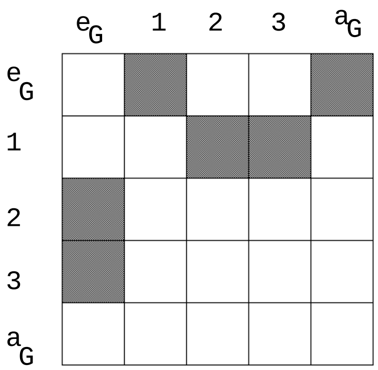

Figure 3: Associated relation of the underlying flowchart

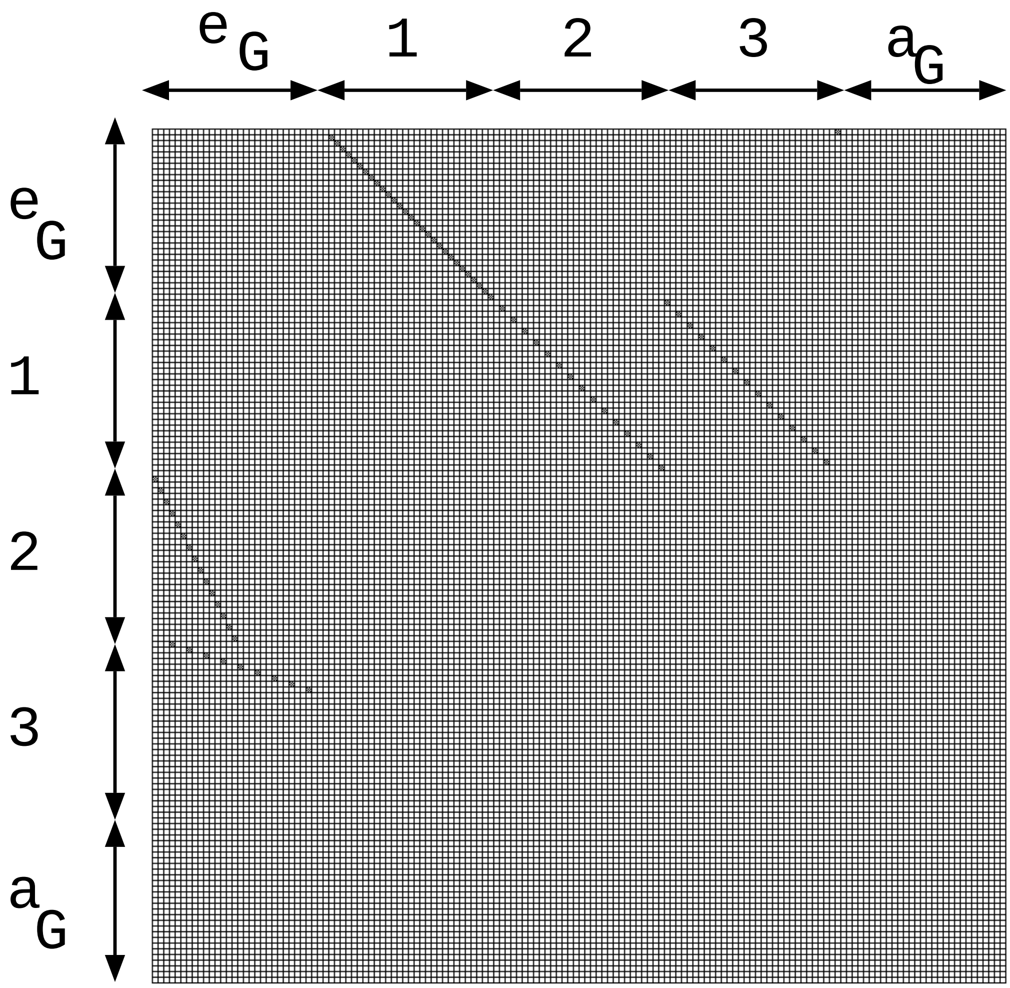

Figure 4: Associated relation of the situation graph

We can read the associated relation of the situation graph as follows. Every 1 (or a black box) in the associated relation of the underlying flowchart is replaced by a relation, which maps the internal states. For example, the entry in the third row and thirty-third column says, that the state $x = 3$ in the situation testing the loop-condition is mapped to the state $x = 3$ (the condition is true) in the situation testing the if-condition.

The input and output relation looks as follows:

$$e = \begin{pmatrix} I \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad a = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ I \end{pmatrix}$$

Here we use an abstract notation for such relations. The I stands for the identity relation. So, the input relation maps the states ($x = 1, \ldots, x = 30$) to the correponding states at the beginning of the program. In the same manner the output relation hides all internal states of the program.

Now, the question arises, which input data are mapped by the program to which output data. We define:

Definition 4.2 *In a program with input relation e, output relation a and associated relation B (of S), we call*

1. B^* *action,*

2. $C := B^* \sqcap \overline{BL}^{\mathsf{T}}$ *terminating action,*

3. $\Sigma := e^{\mathsf{T}} C a$ *effect.* □

The action of a program is thus the reachability in the situation graph. The terminating action, on the other hand, associates with a given state only the last state reachable from it, which need not belong to an output position of the program. Operations e and a extract the part leading from input to output positions. The effect of a program is the relation that relates the states before entering into the program to those after exiting it. This relation can be seen as the semantics of the program. Based on this semantic we can define partial correctness as follows.

Definition 4.3 *A program $\mathcal{P}$ with effect Σ is called partially correct with respect to a precondition $v = vL$ and a postcondition $n = nL$, if $\Sigma^{\mathsf{T}} v \subset n$ holds.* □

This condition reads as follows: If an output fulfills the postcondition, there is an input, which fulfills the precondition and a program run mapping this input to the given output. Cleary, one would like to check correctness of a program more directly by looking at its associated relation B rather than investigating the more complicated effect Σ. The essential ideas in this direction are contained in the Floyd-Hoare-Calculus [Floyd 67] and [Hoare 69]. The core of these methods is contained in the following theorem.

Theorem 4.4 (Contraction Theorem) *A program $\mathcal{P}$ with associated relation B, input relation e and output relation a is partially correct with respect to v and n, if and only if there is a predicate Q such that the following holds:*

$$B^{\mathsf{T}} Q \subset Q, \;\; v \subset e^{\mathsf{T}} Q, \;\; a^{\mathsf{T}}(Q \sqcap \overline{BL}) \subset n. \qquad \square$$

Now, we will apply the notions just introduced to our example.

Example First, we compute the effect Σ of our program by using the basic and the closure operations. We get the following relation:

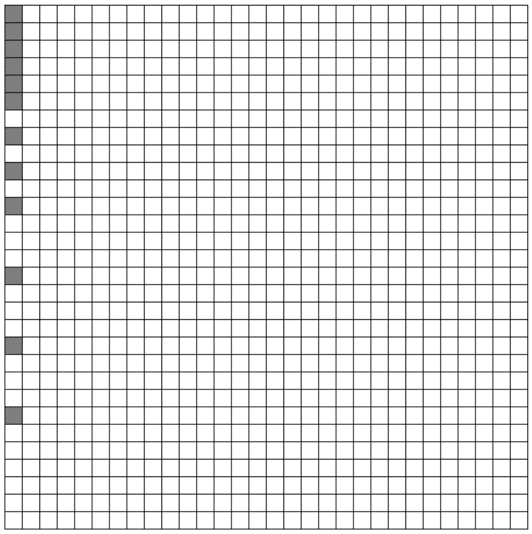

Figure 5: Effect Σ

We can read this relation as follows: The state $x = 6$ is mapped to the state $x = 1$. On the other hand, if we start in the state $x = 7$, we get no result.

By using the inclusion test, we can check partial correctness of this program with respect to given pre- and postconditions. For example, our program is partially correct with respect to the precondition L (true) and the postcondition $x = 1$.

In the rest of this paper, we will concentrate on total correctness.

Definition 4.5 *A program $\mathcal{P}$ with associated relation B, terminating action C, initial part $J(B)$ and effect Σ is called totally correct with respect to a precondition $v = vL$ and a postcondition $n = nL$, if the following holds:*

$$v \subset e^{\mathsf{T}} J(B) \sqcap e^{\mathsf{T}}\, \overline{C\, \overline{an}}\ . \qquad \square$$

So, a program $\mathcal{P}$ to be totally correct with respect to v and n means the following: Given a state satisfying v, there are no computation sequences of arbitrary length. Moreover, with the termination action C only states can be reached that satisfy the postcondition n after the output a.

If the program $\mathcal{P}$ is deterministic, this definition reduces to a simple inclusion.

Theorem 4.6 *A deterministic program $\mathcal{P}$ is totally correct with respect to v and n, if and only if $v \subset \Sigma n$ holds.* $\square$

We take a short look at our example.

Example Our example is a deterministic program. So, we can check total correctness again by a simple inclusion test. Now, we see that the program is not totally correct with respect to the precondition L and the postcondition $x = 1$, because it does not terminate in every state. But we can easily check that the program is totally correct with respect to the precondition $x < 7$ and the postcondition $x = 1$.

Analogously to partial correctness we ask whether total correctness can be directly read off from the program, i.e. the associated relation B. The answer is contained in a "Contraction Theorem" for total correctness.

Theorem 4.7 (Complement-Expansion Theorem) *Let $\mathcal{P}$ be a program with associated relation B, input relation e and output relation a. $\mathcal{P}$ is totally correct with respect to a precondition v and a postcondition n, if and only if for every predicate Q with $\overline{Q} \subset B\,\overline{Q} \sqcup \overline{an \sqcup BL}$ also $v \subset e^{\mathsf{T}}Q$ holds.* □

Finally, we introduce the weakest precondition of a given program and a given postcondition.

Definition 4.8 *Let $\mathcal{P}$ be a program with associated relation B and n a postcondition. The weakest precondition of $\mathcal{P}$ with respect to n is defined by*

$$wp(\mathcal{P}, n) := e^{\mathsf{T}}J(B) \sqcap e^{\mathsf{T}}\,\overline{C\,\overline{an}}\,. \qquad \Box$$

We defined wp as the greatest relation (the weakest predicate) such that the program is totally correct with respect to wp and the given postcondition.

Example By using the definition above we compute the following weakest precondition of our program with respect to the postcondition $x = 1$.

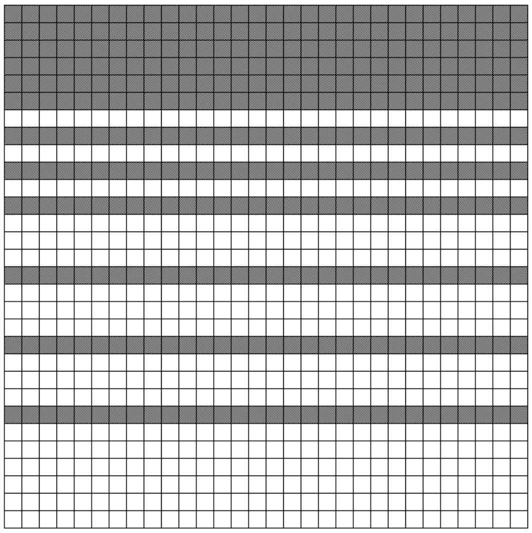

Figure 6: Weakest Precondition

In 1974 resp. 1975 Dijkstra (see [Dijkstra 74], [Dijkstra 75]) postulated a set of conditions for the weakest precondition of a program. The next result shows that our definition of wp fulfills these conditions.

Theorem 4.9 *For the weakest precondition we have*

1. $wp(skip, n) = n$,
2. $wp(abort, n) = 0$,
3. $wp(\mathcal{P}, 0) = 0$,
4. $wp(\mathcal{P}, n_1) \sqcap wp(\mathcal{P}, n_2) = wp(\mathcal{P}, n_1 \sqcap n_2)$,
5. $wp(\mathcal{P}, n_1) \sqcup wp(\mathcal{P}, n_2) \subset wp(\mathcal{P}, n_1 \sqcup n_2)$,
6. $n_1 \subset n_2 \Rightarrow wp(\mathcal{P}, n_1) \subset wp(\mathcal{P}, n_2)$. □

5 Concluding Remarks

In this paper we have shown how to use the RELVIEW system for program verification. Of course such a program has to be finite. But the possibility to compute a weakest precondition of a finite reduct of a given program can be helpfull to find an invariant or the weakest precondition for the abstract program.

Our example is rather small. But we are able to manipulate relations up to a size of 10000 times 10000. So, we can prove correctness of programs with much more states as our sample program.

As mentioned in the introduction and Chapter 3, program verification is not the only example for the use of the RELVIEW system.

References

[Abold-Thalmann et al. 89] Abold-Thalmann H., Berghammer R., Schmidt G.: Manipulation of concrete relations: The RELVIEW-system. Report Nr. 8905, Fakultät für Informatik, Universität der Bundeswehr München (1989)

[Berghammer 92] Berghammer R.: Computing the cut completion of a partially ordered set – An example for the use of the RELVIEW-system. Report Nr. 9205, Fakultät für Informatik, Universität der Bundeswehr München (1992)

[Berghammer Schmidt 91] Berghammer, R., Schmidt, G.: The RELVIEW-system. In: Choffrut C., Jantzen M. (eds.): Proc. STACS '91, LNCS 480, Springer, 535–536 (1991)

[Berghammer et al. 89] Berghammer R., Schmidt G., Zierer H.: Symmetric quotients and domain constructions. Inform. Proc. Letters 33, 3, 163–168 (1989/90)

[Berghammer et al. 93] Berghammer R., Gritzner T., Schmidt G.: Prototyping relational specifications using higher-order objects. Report Nr. 9304, Fakultät für Informatik, Universität der Bundeswehr München (1993)

[Brethauer 91] Brethauer R.: Ein Formelmanipulationssystem zur computergestützten Beweisführung in der Relationenalgebra. Diplomarbeit, Fakultät für Informatik, Universität der Bundeswehr München (1991)

[Dijkstra 74] Dijkstra E. W.: A simple axiomatic basis for programming language constructs. Indag. Math. 36, 1-15 (1974)

[Dijkstra 75] Dijkstra E. W.: Guarded commands, nondeterminacy and formal derivation of programs. Comm. ACM 18, 453-457 (1975)

[Floyd 67] Floyd R. W.: Assigning meaning to programs. In: Schwartz J. T.. (ed.): Mathematical Aspects of Computer Science. Amer. Math. Soc. Proc. Symp. in Appl. Math. XIX 1966, Providence, R. I., 19-32 (1967)

[Herrmann et al. 93] Herrmann S., Kim M., Riehn T.: XRELVIEW 4.0 - Dokumentation zur Trimesterarbeit. Trimesterarbeit, Fakultät für Informatik, Universität der Bundeswehr München (1993)

[Hoare 69] Hoare C. A. R.: An axiomatic basis for computer programming. Comm. ACM 12, 576-580, 583 (1969)

[Landgraf Dombetzki 91] Landgraf J., Dombetzki N.: Erweiterung des RELVIEW Systems. Trimesterarbeit, Fakultät für Informatik, Universität der Bundeswehr München (1991)

[Schmidt Ströhlein 89] Schmidt G., Ströhlein T.: Relationen und Graphen. Springer (1989); English version: Relations and Graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Comput. Sci., Springer (1993)

[Zierer 91] Zierer H.: Relation algebraic domain constructions. Theoret. Comput. Sci. 87, 163–188 (1991)