Applications of the RelView System

R. Behnke, R. Berghammer, T. Hoffmann, B. Leoniuk, P. Schneider

1 Introduction

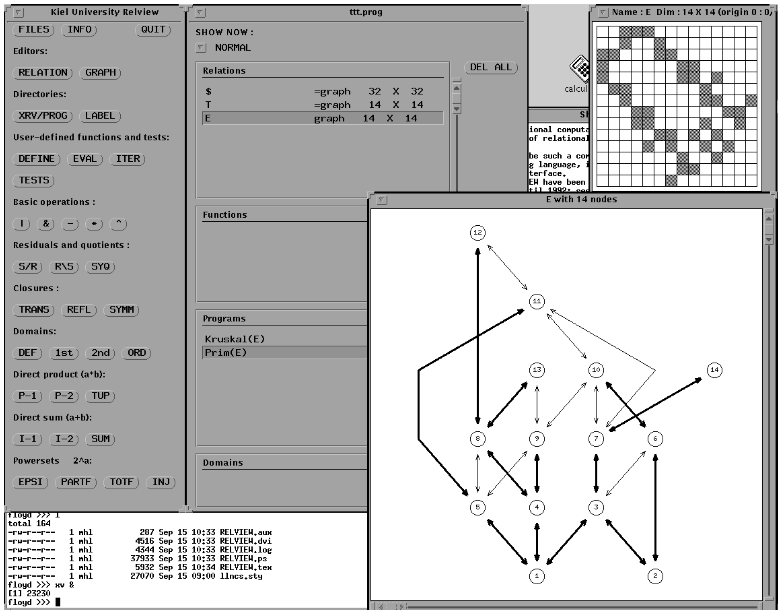
The study of relations has its roots in the second half of the 19th century with the pioneering works of Boole and de Morgan. Later on, Peirce and Schröder developed the algebra of relations. The modern axiomatic development of relational algebra starts with the fundamental work of Tarski and his co-workers. In the last two decades this formalization has widely been used by many mathematicians and computer scientists as a very convenient base for describing fundamental concepts like graphs, orders, games, and combinatorics in mathematics and like relational data bases, Petri nets, data types, and semantics of programming languages in computer science. A lot of examples and references to relevant literature can be found in (Schmidt Ströhlein 1993) and (Brink et al. 1997).

One main reason to use relations is that they can be calculated with so well; using equational reasoning one can construct proofs that are intuitive, memorable, and even machine-checkable. The other main reason is that relational algebra has a fixed and surprisingly small set of operations all of which can easily and efficiently be implemented on finite carrier sets using, for instance, Boolean arrays, linked lists, or binary decision diagrams. Thus, a computer system supporting the manipulation of relations and relational programming can easily be implemented, too. In this article, we consider such a system, called RelView. Based on experiences with a prototype at the University of the German Forces Munich (see e.g., (Berghammer 1992) and (Berghammer Schmidt 1993)), in the last four years it was developed at Kiel University. Written in the C programming language, it runs on Sun SPARC workstations (Solaris 2.5) and INTEL-based Linux systems (kernels 1.2.x, 1.3.x, and 2.0.x). It is equipped with a graphical user interface running under the X-Window system. RelView is available free of charge by FTP from host ftp.informatik.uni-kiel.de, where it is located in the directory pub/kiel/relview. See also URL http://www.informatik.uni-kiel.de/~progsys/relview.html which contains a link to the system description and user manual (Behnke et al. 1997).

The benefits of using RelView are manifold. After a short look at the system, in this article we will give an outline of four applications. First, we show how the system can support relation-algebraic reasoning. Then, we sketch some applications in relational programming, viz. specification testing and rapid prototyping. Further applications of RelView in this field can be found in (Behnke 1997), (Berghammer et al. 1996), and (Behnke et al. 1997), for example. Next, we summarize some of our experiences in using RelView as a teaching tool for relational algebra and program development in the relation-algebraic framework. Finally, we show how to use RelView to develop a more efficient version of it. We close with some work that remains to be done in the future.

2 A short look at RelView

RelView is a totally interactive and completely graphic-oriented computer system. Although the detailed appearance of its windows depends on resources defined in a configuration file, typically the four main windows (viz. the menu window, the directory window, the window of the relation editor, and the window of the graph editor) look as in the following screen snapshot:



In RelView all data are represented as (of course, finite) binary relations which the system visualizes in two different ways. An arbitrary relation may be displayed on the screen as a Boolean matrix. For a relation on one set (a *homogeneous* relation) RelView additionally offers a representation as a directed graph, including several algorithms for its nice drawing. Matrices and graphs can be stored and the system can manage in its workspace as many of these objects simultaneously as memory allows.

The main purpose of RelView is the interactive evaluation and analysis of relational terms. These are constructed from the relations of the workspace using pre-defined operations (which can be accessed through a simple mouse-click) and user-defined relational functions and programs.

Within the language of RelView all constants and basic operations of relational algebra are available. The constants are universal relations L, empty relations O, identity relations I as well as relations coming from domain constructions like projection relations on direct products or membership relations on powersets and sets of (partial) functions, respectively; the operations include meet $R \cap S$, union $R \cup S$, complement $\overline{R}$, composition RS, transposition R^{T}, closures like transitive closure R^{+}, left residual $R\,/\,S$ and right residual $R \setminus S$, and symmetric quotient $\mathsf{syq}(R, S)$. For many other kinds of problems and also to gain in efficiency, further operations on relations are necessary which cannot be introduced as relational functions based upon the basic operations. In RELVIEW, we have implemented

1. the choice of a (relational) point $\mathsf{point}(v)$ contained in a (relational) vector $v \neq \mathsf{O}$ respectively an atom $\mathsf{atom}(R)$ contained in a relation $R \neq \mathsf{O}$,
2. the embedding of the set described by a non-empty vector v by means of an injective mapping $\mathsf{inj}(v)$,
3. the generation of a (finite) set via an initial point init and a "next point function" next.

The meaning of these operations is given by relation-algebraic axioms. For example, in the case of the choice of a point contained in a non-empty vector v we have the axiomatization

$$\mathsf{point}(v) = \mathsf{point}(v)\,\mathsf{L} \qquad \mathsf{point}(v)^{\mathsf{T}}\,\bar{\mathsf{I}} = \overline{\mathsf{point}(v)^{\mathsf{T}}} \qquad \mathsf{point}(v) \subseteq v\,.$$

Here the first equation says that $\mathsf{point}(v)$ is a vector and the second equation expresses that it is injective and surjective; see (Schmidt Ströhlein 1993). Putting together both axioms, $\mathsf{point}(v)$ is a point; see again (Schmidt Ströhlein 1993). Hence, it describes a singleton set respectively an element which – due to the third axiom – is contained in the set described by v. Axiomatic characterizations like the above exactly fit into the component-free relation-algebraic framework.

As already mentioned, in RELVIEW there is only one datatype "relation". Especially, there is no datatype for truth values. Instead *true* and *false* are represented by the only two relations $\mathsf{L}_{\mathbf{1}}$ and $\mathsf{O}_{\mathbf{1}}$ on a specific singleton set $\mathbf{1}$. This allows to describe inclusion of relations via a relational function incl such that $\mathsf{incl}(R,S)$ yields a relation on $\mathbf{1}$ and

$$\mathsf{incl}(R,S) = \mathsf{L}_{\mathbf{1}} \iff R \subseteq S \qquad\qquad \mathsf{incl}(R,S) = \mathsf{O}_{\mathbf{1}} \iff R \not\subseteq S\,.$$

Based on incl, all propositional sentences over relational inclusions can easily be formulated and tested in RELVIEW since, clearly, the propositional connectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\neg$ directly correspond to the operations $\cap$, $\cup$, $\subseteq$, $=$, and $\overline{}$ on the relations $\mathsf{L}_{\mathbf{1}}$ and $\mathsf{O}_{\mathbf{1}}$. As an example, equality of relations can be tested via the relational function

$$\mathsf{eq}(R,S) = \mathsf{incl}(R,S) \cap \mathsf{incl}(S,R)\,.$$

A relational program as supported by RELVIEW has many similarities with a function procedure in Modula-2. It starts with a head line containing the program name and the formal parameters. Then, the declaration of the local relational domains, functions and variables follows. Domain declarations can be used to introduce binary relational products and sums together with some natural relations like projections and injections. A declaration of a (non-recursive) relational function is done as usual in mathematics and has the form $F(X_1, \ldots, X_n) = t$ with parameters X_i, $1 \leq i \leq n$, and a relational term t. The third part of a program is the body, a while-program over relations. As a relational program computes a value, finally, its last part consists of a return-clause followed by a relational term whose value after the execution of the body is the result.

3 Using RelView for relation-algebraic reasoning

The statement of a relation-algebraic theorem is frequently an inclusion of two relational terms. Accordingly its proof is also composed by applications of relational inclusions and leads to reductions of the form

$$t_n \subseteq e_n \Leftarrow \ldots \Leftarrow t_1 \subseteq e_1 \Leftarrow \ \mathit{true}\,.$$

The difficulty is to find the next step within such a sequence. Here it is often useful to experiment with relational properties to get an idea for it. RelView can be used for this task in a very efficent manner.

First, RelView can be applied to find counter-examples. Second, playing with the system and looking at the graphical representations of the resulting relations can help to discover new properties. One can also utilize the system to generalize properties, e.g., by substituting a relational equation for an inclusion or by replacing a relational term with a more general one. Most important, however, is the fact that the system allows to test validity of arbitrary propositional sentences over relational inclusions as shown in Sec. 2. For instance, using RelView validity of

$$P\,Q \subseteq R \iff P^{\mathsf{T}}\,\overline{R} \subseteq \overline{Q}$$

(in (Schmidt Ströhlein 1993) called *Schröder equivalence*) can be tested by the relational function

```
schroeder(P,Q,R) = eq(incl(P*Q,R),incl(P^*-R,-Q)).
```

Since the relations needed for such tests can be randomly generated, a lot of tests can be made in a short time. The advantage of this proceeding is that no tests have to be done by hand (which takes time and is vulnerable to mistakes) and one normally gets a good feeling whether a relational property could hold.

We now illustrate the just sketched proceeding by means of an example which appeared during the verification of a relational program for computing a depth-first-search tree. Let the relation x be a vector, i.e., $x = x\,\mathsf{L}$ be true. Furthermore, let the relation W be a *linear strict-ordering* which means that $W\,W \subseteq W$, $W \cap W^{\mathsf{T}} \subseteq \mathsf{O}$, and $W \cup W^{\mathsf{T}} = \overline{\mathsf{I}}$. Our task was to show that $x \neq \mathsf{O}$ implies

$$W\,\overline{W^{\mathsf{T}}\,x} \subseteq W\,(x \cap \overline{W^{\mathsf{T}}\,x}\,)\,. \tag{1}$$

We started the proof with the Schröder equivalence and obtained that (1) holds if and only if the inclusion

$$W^{\mathsf{T}}\,\overline{W\,(x \cap \overline{W^{\mathsf{T}}\,x}\,)} \subseteq W^{\mathsf{T}}\,x \tag{2}$$

is valid. Our next idea was to simplify $\overline{W\,(x \cap \overline{W^{\mathsf{T}}\,x}\,)}$. Here, we performed some random tests with RelView and they encouraged us to work with

$$\overline{W\,(x \cap \overline{W^{\mathsf{T}}\,x}\,)} = \overline{W}\,(x \cap \overline{W^{\mathsf{T}}\,x}\,)\,. \tag{3}$$

What are the reasons for this equation to be true? For an arbitrary vector v, obviously equation $\overline{R\,v} = \overline{R}\,v$ is false. This was confirmed by the first RelView test we have performed. But then further random RelView tests lead us to the conjecture that $\overline{Rv} = \overline{R}\, v$ is true if v is a point, i.e., in addition injective and surjective. Having discovered this property, its proof was easy. In our special case (3) the vector $x \cap \overline{W^\mathsf{T} x}$ indeed is a point; see (Schmidt Ströhlein 1993) for example. So, by (3) we get that (2) is equivalent to

$$W^\mathsf{T}\, \overline{W}\, (x \cap \overline{W^\mathsf{T} x}\,) \subseteq W^\mathsf{T} x\,. \tag{4}$$

Because of monotonicity of relational multiplication, (4) now suggested to test, whether the two inclusions

$$W^\mathsf{T}\, \overline{W} \subseteq W^\mathsf{T}, \qquad\qquad x \cap \overline{W^\mathsf{T} x} \subseteq x \tag{5}$$

are true since this would imply the desired result (1). Inclusion $W^\mathsf{T}\, \overline{W} \subseteq W^\mathsf{T}$ of (5) easily followed from $W\,W \subseteq W$ and $\overline{W} = \mathsf{I} \cup W^\mathsf{T}$ and validity of the second formula $x \cap \overline{W^\mathsf{T} x} \subseteq x$ of (5) is trivial.

4 Specification testing and prototyping with RelView

Relational algebra is also a powerful framework for the development of efficient algorithms. In this field, the RelView system can be used to support many different validation tasks. The starting point of a relational program development usually is a more or less formal logical and/or set theoretical description of a problem. In a first step, a so-called *explicit relational specification* is derived for that problem. Such a specification is a relational function $f(X_1, \ldots, X_n) = t$, where the X_i, $1 \leq i \leq n$, are the input parameters and t is a relational term fixing the desired result according to the given problem. The derivation is mainly based on simple transformations linking together logical and set theoretical constructs on the one side with relational constructs on the other side.

By evaluating the term t – provided that this is possible at all – RelView can be used to perform various tests on the constructed specification. First, it can be compared with the logical respectively set theoretical description to obtain additional confidence in the correctness of the transformation process itself. Moreover, it can be checked against the informal fixed requirements and even against the intuitive notion of the problem. In particular, this indirectly allows to find weaknesses or inconsistencies of the given logical or set theoretical description as well. We will illustrate this aspect by means of an example below.

Although a specification can often be regarded as an algorithm for the given problem, it usually lacks in efficiency. Therefore, in a second step the specification is normally used as a starting point for a formal program development based on, for example, the language of while-programs which is well suited for this task. RelView allows to support this development in many different ways. First of all, the constructed program can be executed, i.e. prototyped, which can yield further confidence in the correctness of the program in addition to the formal reasoning. Moreover, the execution of relational programs respectively program pieces in the course of a derivation can reveal alternative development steps and possibilities for optimizations. Last but not least, it can give hints for a final efficient implementation of the derived program in a conventional imperative programming language like Modula-2 or C.

In the following, we illustrate the use of RelView for testing a specification with the help of a small example. We consider the graph-theoretic problem of computing the set of *cutnodes* of a simple graph. To this end, firstly we consider the sub-problem to decide whether a node is a cutnode. Let $G = (N, R)$ be a finite simple graph with a non empty set $N = \{n_1, \ldots, n_k\}$ of nodes and a symmetric and irreflexive adjacency relation R on N. A node $n \in N$ is called a cutnode of G if and only if the sub-graph generated by the set $N \setminus \{n\}$ contains more connected components than G. A typical semiformal description of n being a cutnode is that there exist two nodes $m_1, m_2 \in N \setminus \{n\}$ such that each path from m_1 to m_2 contains n. This fixes the idea that a cutnode cannot be "bypassed".

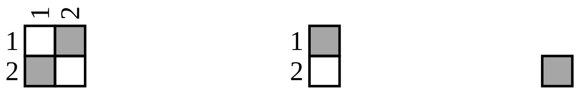
If we represent the node n by the point x (see (Schmidt Ströhlein 1993)), then by simple calculations from the just presented semiformal description we can end up with

$$x \text{ represents a cutnode} \iff (\mathsf{inj}(\,\overline{x}\,)\, R\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}})^{+} \neq \mathsf{inj}(\,\overline{x}\,)\, R^{+}\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}} \tag{6}$$

as relation-algebraic characterization. Here $\mathsf{inj}(\,\overline{x}\,)$ is the injective mapping from $X \setminus \{n\}$ to X given by the vector $\overline{x}$; see (Behnke et al. 1997). Now, we use the two relations $\mathsf{L_1}$ and $\mathsf{O_1}$ as truth values in conjunction with the relational test eq for equality. Then, (6) leads to the explicit relational specification

$$iscut(R, x) = \overline{\mathsf{eq}((\mathsf{inj}(\,\overline{x}\,)\, R\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}})^{+}, \mathsf{inj}(\,\overline{x}\,)\, R^{+}\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}})}\ . \tag{7}$$

Specification (7) can easily be tested with RelView. As an example, we consider a simple graph $G = (N, R)$ with two nodes 1 and 2 and relation R as shown below and check with the system whether the node 1 is a cutnode. All the following pictures are produced by RelView.



Relation R Point x $iscut(R, x)$

The picture of $iscut(R, x)$ on the right hand side indicates that the node 1 represented by the point x is a cutnode of G according to (7). This, however, shows that the specification does not correctly capture the property of being a cutnode as originally given, because removing the node with number 1 from G does not increase the number of connected components. Further investigations show that the semiformal description – the starting point of our relational approach – does not describe the desired property precisely enough, because it does not explicitly demand that the two nodes m_1 and m_2 have to be different and it implicitely uses a notion of a "path" which does not allow a single node to be a path. Considering these aspects, we finally obtain

$$iscut(R, x) = \overline{\mathsf{eq}((\mathsf{inj}(\,\overline{x}\,)\, R\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}})^{+} \cap \overline{\mathsf{I}}, \mathsf{inj}(\,\overline{x}\,)\, R^{+}\, \mathsf{inj}(\,\overline{x}\,)^{\mathsf{T}} \cap \overline{\mathsf{I}})} \tag{8}$$

as correct relational specification of the cutnode problem which means that $iscut(R,x)$ evaluates to $\mathsf{L_1}$ if x represents a cutnode and to $\mathsf{O_1}$ otherwise.

Now, let the relational point x_i represent the node n_i for all $i \in \{1,\ldots,k\}$. Then, a relational specification of the vector representing the set of cutnodes we are searching for is an k-ary relational direct sum of truth values, viz.

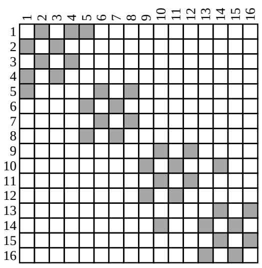
$$Cutnodes(R) = iscut(R,x_1) + \ldots + iscut(R,x_k)\,. \tag{9}$$

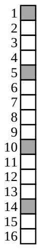
Correctness immediately follows from the fact that N in the set-theoretic sense is isomorphic to the disjoint union $\mathbf{1} + \ldots + \mathbf{1}$ of k copies of the singleton set $\mathbf{1}$ and, hence, we are allowed to identify these two sets.

In RelView all the relational operations used in (8) and (9) are implemented. Furthermore, the k-ary relational direct sum of (9) can easily be computed via a while-loop and the operations init and next for generating sets. The result is the following relational program:

```
Cutnodes(R)
  DECL ipa(R) = R & -I(R);
       del(R,x) = inj(-x) * R * inj(-x)^;
       iscut(R,x) = -eq(ipa(trans(del(R,x))),
                        ipa(del(trans(R),x)));
       x, c
  BEG  x = init(Ln1(R)); c = iscut(R,x);
       WHILE -empty(next(x)) DO
         x = next(x); c = c + iscut(R,x) OD
       RETURN c
  END.
```

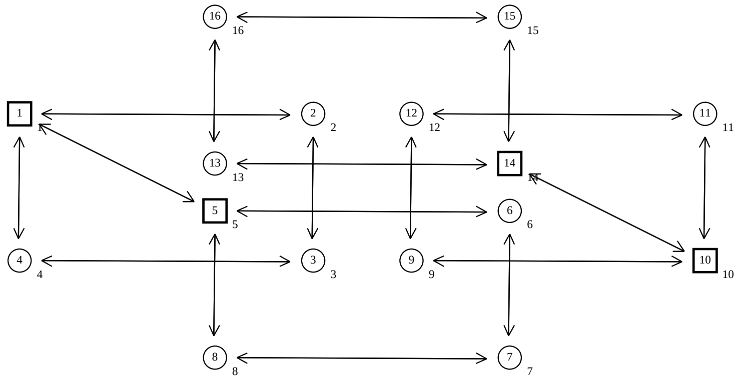
The left-hand of the next two RelView pictures shows a symmetric and irreflexive relation as a 16×16 Boolean matrix `R` which rows and colums are labeled by the numbers from 1 through 16. If we evaluate the relational term `Cutnodes(R)` with RelView and after that add the row labels of `R` also to the rows of the result, then on the window of the relation editor of RelView the right-hand of the following pictures is shown.





From the 16×1 Boolean vector `Cutnodes(R)` we see that the simple graph corresponding to the Boolean Matrix `R` has exactly four cutnodes, viz. the nodes 1, 5, 10, and 14.

In the following picture the relation `R` is depicted as a simple graph and the cutnodes are emphasized by drawing them as squares:

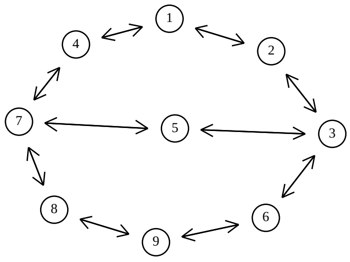


The drawing of this picture was supported by the grid facility of the RelView graph editor.

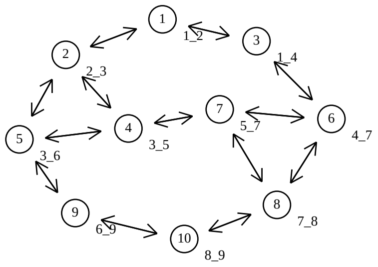
5 RelView as a teaching tool

At Kiel University, the RelView system has been extensively used over the last years in graduate lectures, the exercises accompanied with them, seminars, and diploma theses. The benefits of using the system as a teaching tool in education are manifold. For example, advanced relational operations and properties, e.g., relational domain constructions, are not so commonly known that students can be expected to understand them quickly. Here we have recognized that visualizing such concepts in RelView is very helpful during teaching them. Furthermore, since the RelView system allows computations to be executed in a stepwise fashion, experience shows that it is a very good means to demonstrate how a certain relational algorithm works and also facilitates program validation and detection of logical errors. As a third application it should be mentioned that we found it very attractive to use RelView for producing good examples in teaching – in particular when dealing with discrete structures like graphs, orders, Petri nets, finite automata – which frequently have been proven to be the key of fully understanding a concept.

To demonstrate the last application, in the following we consider as a small example the construction of the so-called *line graph* $L(G) = (E, S)$ of a simple graph $G = (N, R)$ using RelView. The line graph $L(G)$ is the simple graph obtained by taking the edge set $E = \{\{m, n\} : \langle m, n\rangle \in R\}$ of G as node set and joining two of these nodes in $L(G)$ whenever the corresponding edges in G are different and have a node in common. The following two pictures show a graph G and its line graph $L(G)$ as produced by RelView using a standard spring embedder algorithm for computing the layout of the graphs:



Graph G



Line graph $L(G)$

The computation of the adjacency relation S of $L(G) = (E, S)$ from the adjacency relation R of $G = (N, R)$ can be done in three steps. First, we "direct" the simple graph G by computing an irreflexive and asymmetric relation Q such that $R = Q \cup Q^{\mathsf{T}}$. Then, we represent the graph $G' = (N, Q)$ as an incidence structure $H = (N, E, M)$. Here the computation of the incidence relation M of H from the adjacency relation Q of G' is based on the representation of Q as a vector v with domain $N \times N$ (details can be found in Sec. 7.5 of (Schmidt Ströhlein 1993)) and described by the relational function

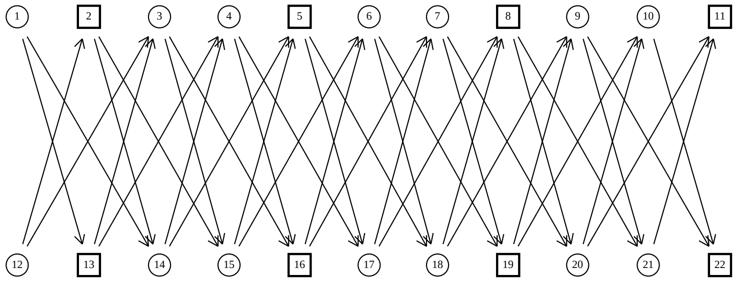
$$AdjToInc(Q) = \mathsf{inj}((\pi_1\, Q \cap \pi_2)\, \mathsf{L})\, (\pi_1 \cup \pi_2)$$

with the two relations π_1 and π_2 being the projections from the direct product $N \times N$ to the first respectively the second component. Finally, the adjacency relation S of the line graph $L(G)$ equals the so-called edge adjacency $M\, M^{\mathsf{T}} \cap \overline{\mathsf{I}}$ of the relation M. If we formulate the just described procedure in the language of RelView, we get

```
Linegraph(R)
  DECL Prod = PROD(R,R);
       Q, v, M
  BEG  Q = direct(R);
       v = dom(p-1(Prod) * Q & p-2(Prod));
       M = inj(v) * (p-1(Prod) | p-2(Prod))
       RETURN M * M^ & -I(M * M^)
  END
```

as a relational program for computing the relation of a line graph from the relation of a simple graph.

Next, we use the RelView system to demonstrate how a certain algorithm works. Assume $G = (N, R)$ to be a directed graph. A set K of nodes is called a *kernel* of G if every node outside of K has at least one successor inside K and no two nodes of K are neighbours. In the following RelView picture an example is shown. This directed graph describes the well-known game of matches with a pile of 10 matches and two players which alternately take one or two matches until the player loses who is unable to take a match since the pile is empty. Its only kernel is emphasized by square nodes.



If we represent a node set K of a directed graph $G = (N, R)$ by a vector k, then the property of K being a kernel of G becomes a fixed-point equation $k = \tau_R(k)$, where the relational function τ_R is defined as $\tau_R(x) = \overline{Rx}$. It is well-known that for G being finite and cyclefree τ_R has exactly one fixed point which is the limit of the chain

$$\mathsf{O} \subseteq \sigma_R(\mathsf{O}) \subseteq \sigma_R^2(\mathsf{O}) \subseteq \sigma_R^3(\mathsf{O}) \subseteq \ldots \tag{10}$$

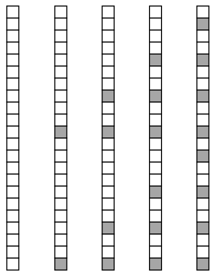
with the relational function σ_R given by $\sigma_R(x) = \tau_R(\tau_R(x))$. Details can be found, e.g., in (Schmidt Ströhlein 1993). Hence, the computation of the only kernel of a finite cyclefree graph with relation R is possible using the following relational programm:

```
Kernel(R)
  DECL tau(R,x) = -(R * x);
       sigma(R,x) = tau(R,tau(R,x));
       k
  BEG  k = On1(R);
       WHILE -eq(k,sigma(R,k)) DO
         k = sigma(R,k) OD
       RETURN k
  END.
```

How this program works, i.e., how the limit of chain (10) is computed step by step, also easily can be demonstrated using RelView. First, one draws the relation as a directed graph on the window of the graph editor and stores it in the system's workspace as `R`. Then, one defines two relational functions

```
tau_R(x) = -(R * x)        sigma_R(x) = tau_R(tau_R(x)).
```

Using a pop-up menu of the relation editor, in the third step one produces an empty Boolean vector `k` with the same number of rows than `R`. It represents the first element of chain (10). The Boolean vectors for the remaining elements are, finally, computed using the button `EVAL` for term evaluation of the menu window, where the arguments are `k` (as name of the result of the evaluation) and `sigma_R(k)` (as the term to be evaluated), as long as the value of `k` changes. In the case of the directed graph above, this leads to the sequence



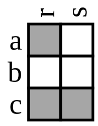
of five 16×1 Boolean vectors the last element of which represents the kernel $K = \{2, 5, 8, 11, 13, 16, 19, 22\}$.

6 Using RelView to improve RelView

In several applications of relational algebra, the use of the current RelView system is restricted to rather small input relations. The reason is that in RelView a relation is stored as a Boolean array and so every entry of a relation consumes 1 Bit of memory. For example, the membership relation on a set M and its powerset 2^M for $|M| = 25$ allocates more than 100 MByte of storage. In this particular case the memory consumption is exponential in the size of the set M. As a consequence, the evaluation of a term containing this membership relation normally takes too much time even on a modern workstation.

Our aim is to improve RelView by exchanging the array-based datatype implementation for relations to overcome these kinds of problems. For this purpose, we have carried out a number of experiments which showed that a promising alternative is given by ordered binary decision diagrams (shortly: OBDDs) as provided by R. Bryant in 1986. More information on OBDDs can be found in (Bryant 1992). In this section, we sketch how RelView has supported these experiments and the development of the new datatype implementation. Furthermore, we give some brief examples to illustrate the advantages of using OBDDs.

In the first step, we focus on the modeling of relations by this data structure. As OBDDs provide a very efficient way to represent Boolean functions, in the following we will present a description of a relation as such a function. This description is based upon binary encodings of the relation's domain and range, respectively. We will briefly illustrate this approach by giving a small example. Assume sets $M = \{a, b, c\}$ and $N = \{r, s\}$ and consider a relation R on these sets, depicted by RelView as the following 3×2 Boolean matrix:



Now, we use two binary encodings $c_1 : M \to \mathbf{B}^2$ and $c_2 : N \to \mathbf{B}$ of the domain M respective range N of R, defined by the following equations:

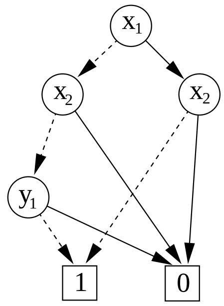
$$
\begin{array}{ll}
c_1(a) = 00 & c_2(r) = 0 \\
c_1(b) = 01 & c_2(s) = 1 \\
c_1(c) = 10 &
\end{array}
$$

Then, the relation R can be interpreted as the ternary partial Boolean function $f : \mathbf{B}^3 \to \mathbf{B}$. The left-hand of the following tables shows this interpretation of R, where the variables x_1 and x_2 come from the encoding of M and y_1 comes from the encoding of N. To describe R by an OBDD, in the next step we use the truth value 0 as a dummy result and pass from the partial function f over to the total function $f_R : \mathbf{B}^3 \to \mathbf{B}$ as indicated in right-hand of the following tables.

x_1	x_2	y_1	$f(x_1, x_2, y_1)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–

x_1	x_2	y_1	$f_R(x_1, x_2, y_1)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

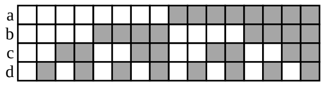
Based on a fixed variable ordering, e.g., $x_1 < x_2 < y_1$, the minimal corresponding OBDD of f_R together with a description of M and N represents R completely. Omitting these additional information, we get the following OBDD of f_R:



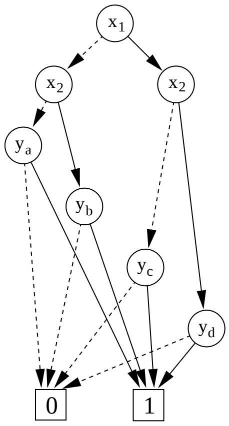
In this picture, a dashed arc describes that the value of its start variable is 0, a non-dashed arc describes that the value of its start variable is 1, and a path from the OBDD's source x_1 to a sink 0 or 1 defines the value of f_R with the n-th node of the path being the n-th argument ($1 \leq n \leq 3$).

As this small example shows, relations can easily be represented as OBDDs. To get an impression of possible advantages of this approach, one has to consider at least three important aspects. Firstly, the sizes of OBDDs describing relations typically used in practical applications of the relational framework are of interest. Secondly, the simplicity of implementing the relational operations on OBDDs has to be investigated. Finally, the efficiency of these implementations should be analysed not only theoretically in a worst-case manner but also practically by a number of tests. At this point, we want to emphasize that the current RelView system can provide excellent support for the investigation of all of these aspects. For this purpose, the implementation of an interface between RelView and an OBDD-package is essential. It consists of a small tool for the conversion of a RelView relation stored in a file into the corresponding OBDD.

The first aspect – the size of OBDDs in comparison with descriptions as Boolean arrays – can be investigated by using RelView very comfortably. We have created many relations via their corresponding Boolean matrices or directed graphs within the system and converted them into OBDDs. This way we realized that OBDDs are a convenient data structure for describing relations, especially with respect to memory consumption. We want to demonstrate this by giving as an example the already mentioned membership relation. The picture



shows the RelView representation of the membership relation between the set $M = \{a, b, c, d\}$ and the powerset 2^M. After encoding the elements of M and 2^M, we get the following OBDD:



Here the four Boolean variables y_a, y_b, y_c, and y_d encode the 16 elements of the range of this relation. At this point it is essential to note that in this case these variables correspond to the rows of the matrix as indicated by their names. As a consequence, the number of OBDD nodes is linear in the number of elements of M in contrast to the exponential size of the Boolean matrix representation of a membership relation. In the previously mentioned case of a membership relation over a set M with 25 elements we obtain an OBDD with about 1 KByte only.

In order to get further impressions of the OBDD description of relations, we have secondly investigated the development of OBDD based algorithms for several relational operations. In this step, RelView was used to carry out the operation in question. After that, both input and output relations have been converted into OBDDs by using the above mentioned interface. Observing many slightly different examples created this way, we often have obtained decisive ideas for the implementation of the desired algorithms. Afterwards, it's correctness could be tested using RELVIEW's interface to the OBDD package again since the comparison of the output of the OBDD algorithm with the OBDD obtained by the RELVIEW output in each case was a fast and easy task. Furthermore, such tests have not been restricted to simple test data. They could be performed in connection with examples of any complexity computable by RELVIEW.

Another step on the way to establish the approach for representing relations by OBDDs is the investigation of the efficiency of the developed implementation. We have experimented with relational specifications using RELVIEW as well as the recently created OBDD algorithms and have compared both performances. The new implementation proved to be superior to the current one. For example, consider the relational specification

$$Kernels(R) = (\overline{\in \cup R \in} \setminus \mathsf{O}) \cap ((\in \cap R \in) \setminus \mathsf{O}), \tag{11}$$

where the right residual $R \backslash S$ of R and S abbreviates the term $\overline{R^{\mathsf{T}} \, \overline{S}}$. Berghammer et al. (1994) have shown that this function enumerates all kernels of a directed graph $G = (N, R)$. Based on (11), the Boolean array implementation of the RELVIEW system can only enumerate all kernels of a graph with up to 25 nodes because of the use of a membership relation $\in$ between N and 2^N. In contrast, the implementation based on OBDDs allows to deal with graphs with approximately 40 nodes. Considering sparse graphs, which appear in many applications, the limit can be extended to even more than 150 nodes.

Although the first OBDD implementation still has to be optimized, the advantages of this new approach are already obvious. It must be stated that in the ongoing work on the OBDD version of RELVIEW a central role has been taken by the system's current version. So, RELVIEW has proved to be the main tool for its own further development.

7 Concluding remarks

RELVIEW is a manipulation system for relations and a prototyping system for relational functions and programs. Its main attraction is its easy and flexible operation by the user since this allows to deal and experiment with (new) relational concepts as well as relational specifications and programs in a very problem-oriented way hereby avoiding unnecessary overhead.

In Sec. 4 and Sec. 5 we have only presented graph-theoretic applications of RELVIEW. But the system may be used for much more discrete structures based on relations, e.g., orders and lattices, Petri nets, and finite automata. Using sequential algebra, a slight generalization of relational algebra, the scope of relational methods and of applications of RELVIEW even can be extended to problems where a purely relational approach leads to unacceptable complexity. The main idea here is the represention of the elements of sequential algebras by relations and the transfer of the sequential operations and their algebraic laws to relations. For details see (Behnke 1997) and (Behnke 1998).

RelView is used by many research groups around the world. For example, at Cape Town University, South Africa, specific relational algebras are computed with its help (P. Jipsen), at Oxford University, UK, it is used in relational program development (J. Ravelo), and at Waterloo University, Canada, it is applied to extract and visualize architectures of large software systems (J. Keast). The latter application uses a tool for converting relations and graphs contained in a RelView-file into ASCII format and vice versa.

Concerning future work on RelView, our main goal is to complete the OBDD version mentioned above. But also some minor work remains to be done. E.g., an improvement is possible in the graph layout. Also the error messages of the parser are not always as helpful as they should be. A third future extension concerns the interface with other systems, e.g., with the relational formula manipulation system and proof checker RALF (Hattensperger et al. 1993).

References

Behnke R. (1997): Extending relational specifications by sequential algebras – Prototyping with RelView. In: Proc. of a Colloquium on Programming languages and fundamentals of programming, Avendorf, Germany, Report 9717, Institut für Informatik und Praktische Mathematik, Universität Kiel, pp. 12-22

Behnke R. (1998): Transformationelle Programmentwicklung im Rahmen relationaler und sequentieller Algebren. Dissertation, Institut für Informatik und Praktische Mathematik, Universität Kiel

Behnke R., Berghammer R., Schneider P. (1997): Machine support of relational computations – The Kiel RelView system. Report 9711, Institut für Informatik und Praktische Mathematik, Universität Kiel

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

Berghammer R., Gritzner T., Schmidt G. (1994): Prototyping relational specifications using higher-order objects. In: Proc. of HOA '93, Amsterdam, The Netherlands, Springer, Berlin, pp. 56-75 (Lecture notes in computer science, vol. 816)

Berghammer R., von Karger B., Ulke C. (1996): Relation-algebraic analysis of Petri nets with RelView. In: Proc. of TACAS '96, Passau, Germany, Springer, Berlin, pp. 49-69 (Lecture notes in computer science, vol. 1055)

Berghammer R., Schmidt G. (1993): RelView – A computer system for the manipulation of relations. In: Proc. of AMAST '93, Enschede, The Netherlands, Springer, London, pp. 405-406 (Workshops in computing)

Brink C., Kahl W., Schmidt G. (1997): Relational methods in Computer Science. Springer, Wien (Advances in computing science)

Bryant R.E. (1992): Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys 24, pp. 293-318

Hattensperger C., Berghammer R., Schmidt G. (1993): RALF – A relation-algebraic formula manipulation system and proof checker. In: Proc. of AMAST '93, Enschede, The Netherlands, Springer, London, pp. 407-408 (Workshops in computing)

Schmidt G., Ströhlein T. (1993): Relations and graphs. Springer, Berlin, (EATCS Monographs on theoretical computer science)