RelView and Rath — Two Systems for Dealing with Relations

Rudolf Berghammer[1⋆], Gunther Schmidt[2⋆] and Michael Winter[2⋆]

[1] Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
D-24098 Kiel, Germany
[2] Fakultät für Informatik
Universität der Bundeswehr München
D-85577 Neubiberg, Germany

Abstract. In this paper we present two systems for dealing with relations, the RelView and the Rath system. After a short introduction to both systems we exhibit their usual domain of application by presenting some typical examples.

1 Introduction

In the area of logical reasoning, people began soon to look for subsets easier to handle than, for example, full predicate logic. This attempt resulted not least in relational reasoning. Already as early as 1915, Leopold Löwenheim postulated that one should resort to reasoning with relations in the "Gebietekalkul", and should "Schröderize" all of mathematics. This approach is certainly burdened with a loss in expressiveness. Nevertheless, such a loss has been accepted in the past by many scientists, as everything looks much simpler and it does not deteriorate expressiveness too much.

When working with relations today, one usually asks for additional computer aid. Three systems with quite different approaches have been proposed from our groups the last years. First, there may be just a specialized support in formula manipulation as in Ralf (see [7, 8]), amended even by some automated features. A second approach is completely "on the model side" as with RelView. Here, instead of working with binary predicates that may result in *true* or *false*, one works with boolean matrices. This is a paradigm shift allowing to incorporate techniques known from linear algebra. In the RelView system this has been elaborated in great detail to the extent that now something is available which might be compared to a "numerics package" — this time however for relational algebra. Thirdly, one may remain on the syntactic side, still avoiding to work in a model. This means concentrating solely on the algebraic rules valid in the relational fragment. This characterizes the Rath approach. Logical reasoning is facilitated since the Rath system offers precise type control. Negation, e.g., need

⋆ Co-operation for this paper was supported by European COST Action 274 "Theory and Applications of Relational Structures as Knowledge Instruments" (TARSKI)

not be avoided, as due to the type restriction no unacceptably large result will show up. RATH also works if some of the rules of relation algebra are abandoned focusing on Dedekind categories, division allegories, etc. All the common aspects are handled simultaneously.

Considered in the context of the newly founded COST action 274: TARSKI, all systems seem extremely well-suited to fostering mechanization. Given the observation that many people keep inventing ideas to cope with relational structures arising around real-world phenomena, there is always the task to study whether these ideas are really helpful — whether they really work. The systems offer detailed computer help in different directions. Here, we exhibit in which way they may be used. Since RALF is currently not maintained, we concentrate on RELVIEW and RATH.

2 Relation-algebraic Preliminaries

In this section, we briefly introduce the basic concepts of relation algebra, some special relations, and some relation-algebraic constructions. For more details concerning the algebraic theory of relations, see e.g., [4, 13].

Given non-empty sets X and Y, the set of all (set-theoretic or concrete) relations with domain X and range Y is denoted by $[X \leftrightarrow Y]$ and we write $R : X \leftrightarrow Y$ instead of $R \in [X \leftrightarrow Y]$. If X and Y are finite and of cardinality m and n, respectively, then we may consider R as a Boolean matrix with m rows and n columns. This matrix interpretation is well-suited for many purposes. Therefore, in this paper we frequently will use matrix concepts and notations also for relations. Especially, we will speak of rows and columns, and we will denote membership by R_{xy} instead of $(x, y) \in R$.

We assume the reader to be familiar with the basic operations on relations, viz. R^{T} (transposition), $\overline{R}$ (negation), $R \cup S$ (union), $R \cap S$ (intersection), RS (composition), $R \subseteq S$ (inclusion, subrelation test), and the special relations O (empty relation), L (universal relation), and I (identity relation). With the set-theoretic operations $\overline{\ }$, $\cup$, $\cap$, $\subseteq$ and the constants O, L such relations respectively Boolean matrices form a complete Boolean lattice. Further well-known laws for operations on relations are, for instance:

$$R^{\mathrm{T}\,\mathrm{T}} = R \qquad Q(R \cap S) \subseteq QR \cap QS \qquad (RS)^{\mathrm{T}} = S^{\mathrm{T}} R^{\mathrm{T}}$$

The theoretical framework for such laws to hold is that of a *relation algebra*. First, such an algebraic structure is a category. I.e., there is a class of objects; for every pair A, B of objects there is a class $\mathcal{R}_{AB}$ of morphisms, and for all triples $\mathcal{R}_{AB}$, $\mathcal{R}_{BC}$, $\mathcal{R}_{AC}$ there is a composition from $\mathcal{R}_{AB} \times \mathcal{R}_{BC}$ to $\mathcal{R}_{AC}$ such that associativity holds and for all $\mathcal{R}_{AB}$ there exists precisely one left identity from $\mathcal{R}_{AA}$ and one right identity from $\mathcal{R}_{BB}$. The morphisms are called (abstract) relations and for their composition and the identity relations we use here the same notation as for concrete relations. However, this category is extended by a transposition operation mapping relations from $\mathcal{R}_{AB}$ to $\mathcal{R}_{BA}$, where we use

again the notation of the concrete case. Furthermore, the following properties are demanded to hold:

1. Every class $\mathcal{R}_{AB}$ is a complete Boolean lattice with the usual operations $\overline{\ }$, $\cup$, $\cap$, the ordering $\subseteq$, and the least (empty) relation O and greatest (universal) relation L.
2. For all relations $Q \in \mathcal{R}_{AB}$, $R \in \mathcal{R}_{BC}$, and $S \in \mathcal{R}_{AC}$ the following so-called *Schröder equivalences* hold:

$$Q^{\mathrm{T}}\overline{S} \subseteq \overline{R} \iff QR \subseteq S \iff \overline{S}R^{\mathrm{T}} \subseteq \overline{Q} \tag{1}$$

Often, in particular within the RelView system, the following so-called *Tarski rule* is required as a further axiom; it is strongly connected to a generalization of the notion of simplicity known from universal algebra:

$$\mathsf{L}R\mathsf{L} = \mathsf{L} \iff R \neq \mathsf{O} \tag{2}$$

Note that for $R \in \mathcal{R}_{BC}$ in the equality of (2) there occur three possible different universal relations, viz. from $\mathcal{R}_{AB}$ and $\mathcal{R}_{CD}$ on the left-hand side and from $\mathcal{R}_{AD}$ on the right-hand, which all are denoted by the same symbol.

Let R be a (concrete or abstract) relation. Then R is called univalent (or *functional* respectively a *partial mapping*) if $R^{\mathrm{T}}R \subseteq \mathsf{I}$, and *total* if $R\mathsf{L} = \mathsf{L}$. As usual, a *mapping* is a univalent and total relation. Relation R is called *injective* if R^{T} is univalent and *surjective* if R^{T} is total. A *bijective* relation is an injective and surjective relation.

Now, let R additionally be *homogeneous*, i.e., a relation for which the specific product RR exists. (In the abstract case this is equivalent to $R \in \mathcal{R}_{AA}$ and in the concrete case this is equivalent to $R : X \leftrightarrow X$.) Then R is called *reflexive* if $\mathsf{I} \subseteq R$, *transitive* if $RR \subseteq R$, and *antisymmetric* if $R \cap R^{\mathrm{T}} \subseteq \mathsf{I}$. A *partial order* is a reflexive, antisymmetric, and transitive relation. The *transitive closure* of R is defined as $R^+ = \bigcup_{i>0} R^i$, where $R^0 = \mathsf{I}$ and $R^{i+1} = RR^i$ for all $i \in \mathbb{N}$. Using R^+, the *reflexive-transitive closure* R^* of R may be defined through $R^* = \mathsf{I} \cup R^+$. If $R^+ \subseteq \overline{\mathsf{I}}$, then R is said to be *acyclic*.

A relation v with $v = v\mathsf{L}$ is called a *(row-) vector*. In the case of a concrete relation $v : X \leftrightarrow Y$ this condition means that an element from X is either in relation to none of the elements or to all elements of Y. Hence, v equals a Cartesian product $X' \times Y$, where X' is a subset of X. As for a concrete vector the range is without relevance, we consider in the following frequently vectors $v : X \leftrightarrow \mathbf{1}$ with a singleton set $\mathbf{1} = \{\bot\}$ as range and write then v_x instead of $v_{x\bot}$, i.e., suppress the second index. Such a vector v may be considered as a Boolean matrix with exactly one column, i.e., as a Boolean column vector. It describes the set $X' = \{x \in X \mid v_x\}$.

Sets may also be described via embedding mappings. Given an injective mapping $\imath : X' \leftrightarrow X$, we may regard X' as a subset of X. Then the vector $\imath^{\mathrm{T}}\mathsf{L} : X \leftrightarrow \mathbf{1}$ describes X' in the above sense. A transition in the other direction, i.e., the construction of an injective mapping $\mathsf{inj}(v) : X' \leftrightarrow X$ from a given non-empty vector $v : X \leftrightarrow \mathbf{1}$ describing X' in such a way that $\mathsf{inj}(v)_{yx}$ if and only if $y = x$, is also

possible. Using matrix terminology, one only has to remove from the identity matrix those rows which don't correspond to an element of X'. We call $\mathsf{inj}(v)$ the *injective mapping generated by* v. A relation-algebraic axiomatization of this construction can be found in [2].

The *left residual* of S over R is defined by $S \,/\, R = \overline{\overline{S}R^{\mathrm{T}}}$ and the *right residual* of S over R is defined by $R \setminus S = \overline{R^{\mathrm{T}}\overline{S}}$. One also considers relations which share properties of left and right residuals simultaneously, viz. *symmetric quotients*. This construction is defined by $\mathsf{syq}(R,S) = (R \setminus S) \cap (R^{\mathrm{T}} \,/\, S^{\mathrm{T}})$. In the case of concrete relations we have that $(S \,/\, R)_{xy}$ if and only if R_{yz} implies S_{xz} for all z, that $(R \setminus S)_{xy}$ if and only if R_{zx} implies S_{zy} for all z, and that $\mathsf{syq}(R,S)_{xy}$ if and only if R_{zx} is equivalent to S_{zy} for all z.

3 A Short Introduction to the Systems

In this section, we want to give an impression of two computer systems, called RelView and Rath. Applications will be presented in Section 4. More details and advanced applications can e.g., be found in [2, 3, 9, 10].

3.1 The RelView-System

RelView is an interactive and graphic-oriented computer system for calculating with relations and relational programing. In it all data are represented as relations which the system visualizes in two different ways. First, for homogeneous relations it offers a representation as directed graphs, including sophisticated algorithms for drawing them nicely. Alternatively, arbitrary relations may be depicted as Boolean matrices. This second representation is very useful for visually editing and also for discovering various structural properties that are not evident from a representation of relations as directed graphs. Because RelView computations frequently use very large relations, for instance, membership, inclusion, and size comparison on powersets, the system uses a very efficient implementation of relations via reduced ordered binary decision diagrams. See [10] for its detailed description.

The RelView system can manage as many relations simultaneously as memory allows and the user can manipulate and analyse them by pre-defined operations, tests and user-defined relational functions and relational programs. The pre-defined operations on relations include e.g., `^`, `-`, `|`, `&`, and `*` for transposition, negation, union, intersection, and composition; the relational tests include e.g., `incl`, `eq`, and `empty` for testing inclusion, equality, and emptiness of relations. All that can be accessed through command buttons and simple mouse-clicks. But the usual way is to use the pre-defined operations and tests to construct relational functions and relational programs.

A declaration of a relational function in the programming language of the RelView system is done as usual in mathematics. Hence, it has the form $f(R_1, \ldots, R_n) = E$, where f is the name of the function, the R_i, $1 \leq i \leq n$, are the formal parameters (standing for relations), and E is a relation-algebraic

expression over the relations of the workspace of the RelView system that can additionally contain the formal parameters R_i. As a simple example, the following unary relational function `hasse` computes the so-called Hasse diagram $R \cap \overline{RR^+}$ of an acyclic relation R:

```
hasse(R) = R & -(R * trans(R)).
```

In this declaration, a call of the pre-defined operation `trans` yields the transitive closure of its argument.

A relational program in RelView essentially is a while-program based on the datatype of relations. Such a program has many similarities with a function procedure in languages like Pascal or Modula-2. It starts with a head line containing the name of the program and the list of formal parameters. Then the declarations of the local domains, functions, and variables follow. The last part of a program is its body, a sequence of statements which are separated by semicolons and terminated by the `RETURN`-clause. We give again a simple example. If $g = (X, R)$ is a directed graph with the set of arcs given by the relation $R : X \leftrightarrow X$ and $s : X \leftrightarrow \mathbf{1}$ is a vector describing a subset X' of X, then the vector $(R^*)^{\mathrm{T}} s : X \leftrightarrow \mathbf{1}$ describes the set of those vertices which are reachable from a vertex of X'. Without using the reflexive-transitive closure, the latter vector may be computed by the following relational program:

```
reach(R,s)
  DECL u, v
  BEG  u = s;
       v = R^ * u & -u;
       WHILE -empty(v) DO
         u = u | v;
         v = R^ * u & -u OD
       RETURN u
  END.
```

RelView can be used to solve many different tasks. First, it assists the formulation and the proof of relation-algebraic theorems. In this field, the system can help to construct examples which support the validity of a theorem or to find — via random generated relations — counter-examples to disprove the considered relation-algebraic property. Relational program development is a second very important application of RelView. Whereas relation algebra in combination with a programming logic (e.g., the Hoare calculus) forms the formal basis for ensuring correctness of the derived relational programs, RelView supports many validation tasks for the development of a relational algorithm. For example, it can be applied to check the formal relational problem specification against the informal fixed requirements. Experimenting with relations and relation-algebraic propositions, the system may also help to find loop invariants or other decisive properties necessary for a correctness proof of a relational program. As a third application, the execution of a relational program or a piece of it by means of RelView in the course of a program derivation can reveal alternative development steps and possibilities for optimization.

3.2 The RATH-System

The RATH-System presents a library of Haskell modules that allows to explore relation algebras and several weaker structures such as categories, allegories, distributive allegories, division allegories (see e.g., [5]) and Dedekind categories (see e.g., [12]) by providing tools to construct and test such structures. These modules constitute a common framework for calculational work with all the structures mentioned. It takes into account that they share concepts and properties so as to be able to, for instance, introduce the idea of division only once for division allegories and to directly reuse it for the more specific Dedekind allegories as well as for relation algebras.

For example, in RATH a parameterized data structure `Cat obj mor` representing categories with objects given by the type `obj` and morphisms given by the type `mor` may be defined as follows:

```
data Cat obj mor = Cat
  {cat_isObj   :: obj -> Bool
  ,cat_isMor   :: obj -> obj -> mor -> Bool
  ,cat_objects :: [obj]
  ,cat_homset  :: obj -> obj -> [mor]
  ,cat_source  :: mor -> obj
  ,cat_target  :: mor -> obj
  ,cat_idmor   :: obj -> mor
  ,cat_comp    :: mor -> mor -> mor}
```

Here `cat_objects` yields the list of objects and `cat_homset` yields for a pair of objects the list of morphisms. Source and target of a morphism are computed via `cat_source` and `cat_target`. Composition of morphisms is given by `cat_comp` and `cat_idmor` applied to an object yields the corresponding identity morphism. Finally, `cat_isObj` and `cat_isMor` test whether an element from `obj` respectively `mor` is indeed an element of the category. This is necessary, as one might be using the datatypes `Int` or `String` to denote the objects.

As an application of the above data structure, we want to implement the one-object category of truth values. It may be defined as an element of the data structure of categories as follows:

```
catB :: Cat () Bool
catB = Cat
  {cat_isObj   = const True
  ,cat_isMor   = const $ const $ const True
  ,cat_objects = [()]
  ,cat_homset  = const $ const [False, True]
  ,cat_source  = const ()
  ,cat_target  = const ()
  ,cat_idmor   = const True
  ,cat_comp    = (&&)}
```

There is exactly one object `()` of type `()` and two morphisms `True` and `False` of type `Bool`. Composition is given by intersection such that `True` becomes the identity. Now, the comprehensive test mechanism of RATH could be used to verify that `catB` is indeed a category. An execution of

```
performAll acat_TEST catB
```

will apply all pre-defined tests for categories on this structure. Of course, it is also possible to perform other tests on such a structure in order to find models with a specific property by using the underlying language Haskell. Later on, we will demonstrate this approach by an example.

A next step in the hierarchy of structures could be the representation of allegories. Following [5], an allegory is a category with some extra structure, in particular an intersection, a transposition operation, and an inclusion test. Within RATH, a corresponding parameterized data structure looks as follows:

```
data All obj mor = All
  {all_cat    :: Cat obj mor
  ,all_transp :: mor -> mor
  ,all_meet   :: mor -> mor -> mor
  ,all_incl   :: mor -> mor -> Bool}
```

Similarly, also other relational categories, including relation algebras, can be introduced. In doing so, it is also possible to compute different relation-algebraic expressions. Usually, such an execution is not as efficient as in RELVIEW. But in contrast with RELVIEW, the RATH system provides the possibility to switch to nonstandard relation algebras, i.e., to exchange the underlying model of the relational category in question.

RATH also provides means to construct new algebras from given ones as product algebras, subalgebras and matrix algebras. Last but not least, it is possible to generate a specific relational category by defining the corresponding operations on the set of atoms and taking the complex algebra over this atom structure (see [11]). This reduces the size of the algebra and the complexity of the operations. Using the representation of relation algebras by their atom structure, a wide variety of such algebras was generated with the system. For the moment, this variety contains 4527 different integral relation algebras (see again [11]). Besides these algebras, there are several examples of relation algebras included, which model quite simple everyday situations such as compass directions, interval interdependency, spatial information with "mereology", etc. Note that the number of available algebras is much larger since one may apply the constructions mentioned above to those integral algebras, too.

The sources of the whole RATH-System constitute executable Haskell code. A first account on RATH is given in [9]. This report contains examples of non-standard relation algebras — in particular algebras which, considered from the classical viewpoint, fail to correspond to our imagination of relations as sets of pairs. A very well-known non-standard relation algebra goes back to R. McKenzie and is described in detail in [13].

4 Applications

This section is devoted to some applications of RelView and Rath. First, we concentrate on RelView and show how to solve problems on concrete relations with its help. In the second subsection, we then use Rath to verify that a relation-algebraic proof of a property obviously holding for concrete relations requires the Tarski rule (2) as an additional axiom and that a well-known property of direct products and disjoint unions requires representability.

4.1 Computing Cut Completions and Concept Lattices

Let (X, R) be a partially ordered set, i.e., $R : X \leftrightarrow X$ be a partial order relation. Furthermore, assume $\varepsilon : X \leftrightarrow 2^X$ to be the membership relation between X and its powerset 2^X. This means that ε_{xs} if and only $x \in s$. For $s \in 2^X$, let $\mathrm{Ma}_R(s)$ denote its upper bounds wrt. R and $\mathrm{Mi}_R(s)$ denote its lower bounds wrt. R. Then $c \in 2^X$ is called a *(Dedekind) cut* of (X, R) if

$$c = \mathrm{Mi}_R(\mathrm{Ma}_R(c)), \tag{3}$$

i.e., if the first-order formula

$$\forall\, x : x \in c \leftrightarrow x \in \mathrm{Mi}_R(\mathrm{Ma}_R(c)) \tag{4}$$

holds. Obviously, formula (4) is equivalent to the formula

$$\exists\, s : \forall\, x : (x \in c \leftrightarrow x \in \mathrm{Mi}_R(\mathrm{Ma}_R(s))) \wedge c = s. \tag{5}$$

It is known that for $x \in X$ the set $(x) = \{y \in X \mid R_{yx}\}$ is a cut, called the *principal cut* generated by x. Now, let $\mathcal{C}$ denote the set of cuts of (X, R). Then $(\mathcal{C}, \subseteq)$ is a complete lattice, called the *cut completion* of (X, R), and the function mapping x to the principal cut (x) is an injective order homomorphism.

For a relation-algebraic construction of the cut completion of (X, R), we start with the definition that $y \in X$ is a lower bound of $s \in 2^X$ if and only if for all z from $z \in s$ it follows R_{yz}. Then we describe s by a vector $v : X \leftrightarrow \mathbf{1}$ and use the property of left residuals given at the end of Section 2. We obtain that the set $\mathrm{Mi}_R(s)$ is described by the vector $\mathsf{mi}(R, v) = R \,/\, v^{\mathrm{T}}$. Transposing the relation R yields $\mathsf{ma}(R, v) = R^{\mathrm{T}} \,/\, v^{\mathrm{T}}$ as the vector describing $\mathrm{Ma}_R(s)$. In the language of RelView, hence, we obtain the following two relational functions `mi` and `ma` for computing lower and upper bounds:

```
mi(R,v) = R / v^.        ma(R,v) = R^ / v^.
```

If the second argument of these functions is not a vector but an arbitrary relation, then obviously they compute lower and upper bounds column-wise.

Using relation algebra and the formulae (4) and (5) — for the characterization of cuts (5) is more suited since it immediately leads to a symmetric quotient construction $\mathsf{syq}(\varepsilon, \ldots)_{cs}$ — in combination with the three relational functions

mi, ma, and syq, we obtain the vector $\mathsf{cutvector}(R) : 2^X \leftrightarrow \mathbf{1}$ describing the elements of 2^X which are cuts, i.e., the set $\mathcal{C}$, as follows:

$$\mathsf{cutvector}(R) = (\mathsf{syq}(\varepsilon, \mathsf{mi}(R, \mathsf{ma}(R, \varepsilon))) \cap \mathsf{I})\mathsf{L}$$

This relational specification may immediately be transformed into a relational program in the language of RelView. The result is:

```
cutvector(R)
  DECL M, O, I
  BEG  M = epsi(On1(R));
       O = On1(M^);
       I = I(O * O^)
       RETURN dom(syq(M,mi(R,ma(R,M))) & I)
  END.
```

In this program `On1`, `epsi`, `I`, and `dom` are pre-defined operations. The call `On1(S)` yields the universal vector with one column and the same row number as `S`, the call `epsi(v)` yields the membership relation with the cardinality of the base set given by the row number of the vector `v`, the call `I(S)` yields the identity relation with the same dimension as `S`, and a call `dom(S)` computes the composition of `S` with a one-column universal vector.

Let $v : 2^X \leftrightarrow \mathbf{1}$ abbreviate the vector $\mathsf{cutvector}(R)$ and $\mathsf{inj}(v) : \mathcal{C} \leftrightarrow 2^X$ be the injective mapping generated by v. Furthermore, define the relation $C : X \leftrightarrow \mathcal{C}$ by $C = \varepsilon\, \mathsf{inj}(v)^{\mathrm{T}}$. Then a little reflection shows for all $x \in X$ and $c \in \mathcal{C}$ the equivalence of $x \in c$ and C_{xc}. This means that the columns of C describe the cuts of (X, R). Cuts are ordered by inclusion. Using the property of right residuals given at the end of Section 2, we get for all $c, d \in \mathcal{C}$ the equivalence of $c \subseteq d$ and $(C \setminus C)_{cd}$. Hence, the ordering on $\mathcal{C}$ equals the right residual $C \setminus C$. If we formulate the procedure just described in the language of RelView, we arrive at the following relational program:

```
cutcompletion(R)
  DECL v, C
  BEG  v = cutvector(R);
       C = epsi(On1(R)) * inj(v)^
       RETURN C \ C
  END.
```

Now, let us turn to concept analysis. Here one deals with *(formal) contexts* which are triples (G, M, I) consisting of a set G of objects, a set M of attributes, and an *incidence relation* $I : G \leftrightarrow M$. A *(formal) concept* is a pair (a, b), where $a \in 2^G$, $b \in 2^M$, $a' = b$, and $b' = a$. Here the sets a' and b' are defined as follows:

$$a' = \{y \in M \mid \forall\, x \in a : I_{xy}\} \qquad b' = \{x \in G \mid \forall\, y \in b : I_{xy}\} \tag{6}$$

If (a, b) and (c, d) are two concepts, then (a, b) is defined to be *less general or equal* than (c, d), denoted by $(a, b) \leq (c, d)$, if $a \subseteq c$ or, equivalently, $b \supseteq d$. With

this relation the set $\mathcal{K}$ of all concepts constitutes a complete lattice, in [6] called *(formal) concept lattice*. Sometimes, e.g., in [1], also the term *Galois lattice* is used. It is obvious that the concept lattice $(\mathcal{K}, \leq)$ is isomorphic to $(\mathcal{K}_G, \subseteq)$, with the carrier set defined as $\mathcal{K}_G = \{a \in 2^G \mid \exists b \in 2^M : (a, b) \in \mathcal{K}\}$, and also to $(\mathcal{K}_M, \supseteq)$, with the carrier set defined as $\mathcal{K}_M = \{b \in 2^M \mid \exists a \in 2^G : (a, b) \in \mathcal{K}\}$.

In the following, we concentrate on the computation of $(\mathcal{K}_G, \subseteq)$. Fundamental for this is the simple fact that the equations of (6) generalize the notions of upper bounds and lower bounds from partial order relations to arbitrary relations. Using also the notations $\mathrm{Ma}_I(a)$ and $\mathrm{Mi}_I(b)$ instead of a' and b', we, furthermore, get for $a \in G$ the equivalence of $a \in \mathcal{K}_G$ and

$$a = \mathrm{Mi}_I(\mathrm{Ma}_I(a)). \tag{7}$$

Property (7) is exactly the defining equation (3) for a set to be a cut. Hence, the lattice $(\mathcal{K}_G, \subseteq)$ generalizes the construction of a cut completion from a partial order relation to an arbitrary (incidence) relation. As a consequence, the relational program `cutcompletion` can also be used to compute for a context (G, M, I) the ordering of the lattice $(\mathcal{K}_G, \subseteq)$.

We have tested this approach with many examples. The following table shows the execution times (in seconds) for contexts of the specific form $(G, G, \overline{\mathrm{I}})$. They constitute the worst case since they lead to 2^n concepts, with n being the cardinality of G. The tests have been carried out on a Sun Fire-280R workstation running Solaris 7 at 750 MHz and with 8 GByte main memory.

n	10	11	12	13	14	15	16	17	18	19	20
cutvector	0.01	0.02	0.43	0.96	2.16	4.64	10.0	28.1	54.2	118	257
cutcompl.	0.19	0.22	0.74	1.89	5.84	13.3	38.9	89.1	204	400	1127

On a modern PC we even obtained better results. E.g., for $n = 19$ instead of 400 only 258 seconds are needed to compute the concept lattice of $(G, G, \overline{\mathrm{I}})$.

In many applications of context analysis, experts learn from contexts by carefully inspecting their concept lattices. Therefore, these lattices have modest size because otherwise they are hard to analyze visually. This is ideal for applying RelView, especially since in such a case the system not only allows the fast computation of the ordering of the lattice but also its nice drawing as a graph and its further interactive graphical and relation-algebraic manipulation. To a certain extent RelView can also be used for larger examples. But this requires more sophisticated relational programs which avoid the use of a membership relation. We have developed such a RelView-program. Starting with the incidence relation it stepwise generates the ordering of $(\mathcal{K}_G, \subseteq)$ by gradually inserting missing least upper bounds and greatest lower bounds. Its detailed description, however, is out of the scope of this paper.

4.2 Investigating Properties of Abstract Relations

As mentioned in Section 3.1, the RelView system may be used to construct counter-examples of a relation-algebraic property in question. Since RelView

uses just one specific model, the relation algebra of concrete relations between finite sets, it may fail. Using RATH we are able to switch to some abstract relation algebra providing the required counter-example. In this section we want to demonstrate this approach.

In any relational category one may prove that the following formula (8) is valid if one of the universal relations on the left hand side is homogeneous:

$$\mathsf{LL} = \mathsf{L} \tag{8}$$

A relational category such that the equation (8) holds in general, i.e., it is valid for all object A, B, C and universal relations from $\mathcal{R}_{AB}, \mathcal{R}_{BC}$ and $\mathcal{R}_{AC}$, respectively, is called a uniform one. Obviously, the Tarski rule (2) implies uniformity. One may ask if there exists a non-uniform relation algebra. An example was given in [14]. This algebra has two objects and at most 4 relations in the corresponding sets of morphisms.

As mentioned in Section 3.2, it is possible to define the operations on relations just on the underlying set of atoms. Therefore, we define the following data structures and lists of elements in Haskell:

```
data Obj = A | B deriving (Eq, Ord, Show)
objseq = [A, B]

data A2 = At1 | At2 deriving (Eq, Ord, Show, Read)
atoms :: Obj -> Obj -> [A2]
atoms A A = [At1, At2]
atoms _ _ = [At1]
```

Consequently, we will have 4 relations between A and A and 2 relations otherwise. Notice, that the `deriving`-clause generates an equality, a linear ordering, a `show` respectively a `read` function for `Obj` respectively `A2`. Transposition and composition of atoms are defined as follows:

```
transpTab :: Obj -> Obj -> A2 -> A2
transpTab _ _ x  = x

atComp :: Obj -> Obj -> Obj -> A2 -> A2 -> [A2]
atComp A A A At1 At1 = [At1]
atComp A A A At2 At1 = []
atComp A A A At1 At2 = []
atComp A A A At2 At2 = [At2]
atComp A A B At1 At1 = [At1]
atComp A A B At2 At1 = []
atComp B A A At1 At1 = [At1]
atComp B A A At1 At2 = []
atComp B A B At1 At1 = [At1]
atComp A B A At1 At1 = [At1]
atComp _ _ _ At1 At1 = [At1]
```

The operations and data structures are converted into a category respectively an allegory by the following declarations:

```
aCat_NUW :: ACat Obj A2
aCat_NUW = ac where
  ac = ACat
 {acat_isObj   = const True
 ,acat_isAtom  = (\ s t a -> a `elem` atoms s t)
 ,acat_objects = objseq
 ,acat_atomset = atoms
 ,acat_idmor   = acat_idmor_defaultM ac
 ,acat_comp    = atComp}

 aAll_NUW :: AAll Obj A2
 aAll_NUW = AAll
  {aall_acat = aCat_NUW
  ,aall_converse = transpTab}
```

Finally, the corresponding relation algebra is given as the complex algebra over the allegory `aAll_NUW`. This is done by the following declaration:

```
ra_NUW :: RA Obj (SetMor Obj A2)
ra_NUW = atomsetRA aAll_NUW
```

Again, the comprehensive test mechanism of RATH can be used to verify that `ra_NUW` is indeed a relation algebra. Furthermore, the pre-defined test for uniformity `ded_uniform_TEST` may be applied to this structure. An execution of

```
performAll ded_uniform_TEST (ra_ded ra_NUW)
```

shows the following test result:

```
=== Test Start ===
non-uniform
 Objects:
  A
  B
  A
 Morphisms:
  SetMor ({At1},A,B)
  SetMor ({At1},B,A)
  SetMor ({At1, At2},A,A)
  SetMor ({At1},A,A)
=== Test End   ===
```

This confirms that `ra_NUW` is not uniform. Furthermore, it gives us a counterexample for the validity of equation (8). The composition of the greatest relation `SetMor({At1},A,B)` from A to B with its transposed `SetMor({At1},B,A)`

yields `SetMor({At1},A,A)`, which, however, is not equal to the greatest relation `SetMor({At1, At2},A,A)` from A to A.

If we denote the cartesian product and the disjoint union of two sets A and B by $A \times B$ and $A + B$, respectively, we have the well-known isomorphism

$$2^A \times 2^B \;\cong\; 2^{A+B}. \tag{9}$$

Within the theory of relations there are abstract counter-parts of cartesian products, disjoint unions and powersets, called the *(relational) product*, the *(relational) sum* and the *(relational) power*. One may ask whether (9) is valid in all relational categories. In [14] it is shown that this is true if all required objects exist. Furthermore, it is shown that every relation algebra may be embedded into an algebra with relational sums and powers. On the other hand, the existence of products implies representability, i.e., the algebra may be embedded into the relation algebra of concrete relations. Since there are non-representable algebras, a relation algebra exists with an object 2^{A+B}, which is not isomorphic to the product of 2^A and 2^B. The proof sketched so far is non-constructive. We may use RATH to develop a concrete example with the required property. This was done by using the matrix algebra over the non-representable algebra of R. McKenzie. A detailed description of this model and its implementation in RATH is, however, out of the scope of this paper.

In both examples given in this sub-section the relation algebra in question was basically known. The RATH system was used to verify the corresponding properties. On the other hand, it is possible to find a specific model just by testing the required property for all relation algebras available within RATH.

5 Conclusion

In this paper we have described the two computer systems RELVIEW and RATH for dealing with relations and have exhibited their usual domain of applications by presenting some typical examples.

The current investigations based on RELVIEW and RATH are manifold. To give two examples for RELVIEW, we presently use the system to solve tasks of systems architecture and re-engineering and for computing permanents of specific matrices appearing in physics. Concerning RATH, a work in progress is, for example, the inclusion of the theory of Goguen categories (see [15] for details) into the system. This kind of a relational category constitutes a convenient theory for dealing with so-called $\mathcal{L}$-fuzzy relations. Our next aim is to get a prototype of an $\mathcal{L}$-fuzzy controller using RATH.

RELVIEW is a system for set-theoretic relations, i.e., works within the standard model of relational algebra, whereas RATH is geard towards working with non-standard models. In this sense, the systems complement each other. But there is also some overlap in their functionality. E.g., the RATH system provides some possibilities for exploration and programming with concrete relations. There is, of course, the drawback that the naive Haskell list implementation of

relations is not particularly efficient. For the future we plan to use the foreign-function interface of Haskell to connect RATH with the efficient implementation of relations in the kernel of RELVIEW. In the last months the latter has been isolated from the entire system and collected in a package called KURE (Kiel University relation package). See URL http://www.informatik.uni-kiel.de/ kure.

Much of the field of TARSKI can be circumscribed by mentioning how often words like *vague, rough, fuzzy, qualitative, uncertain* are used in presenting real world phenomena. Among these, we identified the handling of geographic information in GIS (Geographic Information Systems), dealing with lots of other vaguely defined spatial objects, their region connection etc., and the methods of automatic reasoning on spatial properties. Other activities are devoted to the difficulties of banking and investment corporations in decision making when a multitude of possibly divergent criteria must be taken into account. People work on the design of databases and information systems for large companies in industry including questions of information analysis, knowledge representation document management, and how to organize flexible querying. A diversity of intelligent systems for industry, such as data mining, work-flow design, software development with relational methods,including the demonic specification approach is studied. Times, locations, and events are handled with computer aid, employing temporal and other modal logics. Uncertainty is handled in commonsense reasoning, rough, vague, or approximate logical consequences (in human computer studies, psychology, e.g.); logical formalizations together with inference in general, proof systems from the logical side and automated reasoning as afterwards applied in artificial intelligence. A comprehensive bibliography may be found on the RelMiCS-homepage:

`http://www.relmics.org`

In any case, the two relational systems RATH and RELVIEW are intended to improve mechanization for any method proposed. They are thus really central, as it is easier to estimate usefulness of a method when it can be shown with computer aid that it really works.

While the examples in this paper are meant to demonstrate the usefulness of the systems in rather theoretical oriented examples, researchers are encouraged to try out the systems. With the help of the inventors, they seem capable to solve even intricate problems. Both systems are available via the Internet:

RELVIEW: `http://www.informatik.uni-kiel.de/~progsys/relview.shtml`
RATH: `http://ist.unibw-muenchen.de/relmics/tools/RATH/`

Acknowledgements: We are grateful to Barbara Leoniuk and Ulf Milanese who greatly contributed to the RELVIEW system and to Wolfram Kahl and Eric Offermann who did the same for the RATH system.

References

1. Barbut M., Monjardet B.: Ordre et classification: Algébre et combinatoire. Hachette (1970)

2. Behnke R., Berghammer R., Schneider P.: Machine support of relational computations. The Kiel RelView system. Bericht Nr. 9711, Institut für Informatik und Praktische Mathematik, Universität Kiel (1997)
3. Behnke R., Berghammer R., Meyer E., Schneider P.: RelView — A system for calculation with relations and relational programming. In: Astesiano E. (ed.): Proc. Conf. "Fundamental Approaches to Software Engineering (FASE '98)", LNCS 1382, Springer, 318-321 (1998)
4. Brink C., Kahl W., Schmidt G. (eds.): Relational Methods in Computer Science, Advances in Computing Science, Springer (1997)
5. Freyd P., Scedrov A.: Categories, Allegories. North-Holland (1990)
6. Ganter B., Wille R.: Formal concept analysis: Mathematical foundations. Springer (1999)
7. Hattensperger C., Berghammer R., Schmidt G.: Ralf — A relation-algebraic formula manipulation system and proof checker. Notes to a system demonstration. In: Nivat M., Rattray C., Rus T., Scollo G. (eds.): Proc. 3^{rd} Internat. Conf. "Algebraic Methodology and Software Technology (AMAST '93)", Workshops in Computing, Springer, 405-406 (1994)
8. Hattensperger C.: Rechnergestütztes Beweisen in heterogenen Relationenalgebren. Dissertation, Fakultät für Informatik, Universität der Bundeswehr München (1997)
9. Kahl W., Schmidt G.: Exploring (finite) relation algebras using tools written in Haskell. Report Nr. 2000-02, Fakultät für Informatik, Universität der Bundeswehr München (2000)
10. Leoniuk B.: ROBDD-basierte Implementierung von Relationen und relationalen Operationen mit Anwendungen. Dissertation, Institut für Informatik und Praktische Mathematik, Universität Kiel (2001)
11. Offermann E.: Konstruktion relationaler Kategorien. Dissertation, Fakultät für Informatik, Universität der Bundeswehr München (to appear)
12. Olivier J.P., Serrato D.: Catégories de Dedekind. Morphismes dans les Catégories de Schröder. C.R. Acad. Sci. Paris 290, 939-941 (1980)
13. Schmidt G., Ströhlein T.: Relationen und Graphen. Springer (1989); English version: Relations and Graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993)
14. Winter M.: Strukturtheorie heterogener Relationenalgebren mit Anwendung auf Nichtdetermismus in Programmiersprachen. Dissertation, Fakultät für Informatik, Universität der Bundeswehr München (1998)
15. Winter M.: A new algebraic approach to $\mathcal{L}$-fuzzy relations convenient to study crispness. Information Sciences 139, 233-252 (2001)