Journal on Relational Methods in Computer Science, Vol. 1, 2004, pp. 50 - 72

Computation of Cut Completions and Concept Lattices Using Relational Algebra and RelView

Rudolf Berghammer

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Olshausenstraße 40, D-24098 Kiel
rub@informatik.uni-kiel.de

Abstract. Several relation-algebraic algorithms for computing the cut completion of a partially ordered set are formally developed and afterwards generalized to the case of concept lattices. It is shown how the algorithms can be translated into the programming language of RelView and executed using this system. Furthermore, it is demonstrated how to obtain and draw Hasse diagrams of concept lattices as labeled directed graphs using the RelView system.

1 Introduction

Completion via cuts is a well-known method to embed a partially ordered set into a complete lattice. Its generalization to arbitrary relations leads to concept lattices (sometimes also called Galois lattices). These structures, which constitute the basis of *formal concept analysis* [17], have been proved to be a useful tool in many computer science applications, e.g., knowledge representation and discovery, information retrieval, data mining, and program analysis. A lot of references can be found in the textbook [17].

The problem of computing cut completions and concept lattices is studied in the literature since some decades. One of the first papers seems to be [13], published 1973. In the year 1984 a remarkable simple algorithm for both tasks was presented in [15]. Some further references to algorithms again can be found in [17]. Also the present paper deals with the computation of cut completions and concept lattices. However, it does not present the algorithms in the informal way followed

Received by the editors March 23, 2004, and, in revised form, September 09, 2004.
Published on December 10, 2004.

by a "free-style" correctness proof as usual in the algorithms community but concentrates on the formal development of programs in a certain language using, in addition, the algebra of (binary) relations as main tool. Most of the previous relation-algebraic program developments deal with graph-theoretic problems or problems on structures which are closely related to graphs (like Petri nets). See [3, 4, 6–10, 19–21] to give some references. The present paper is an attempt to extend the relation-algebraic approach to orders and lattices.

When applying formal methods in programming, one usual asks for additional computer aid. We will apply the Kiel RelView system, a visual tool for calculating with relations and relation-algebraic programming. Hence, a further aim of the paper is to demonstrate the practical use of this system, its manifold possibilities, and its flexibility by means of a new domain of applications.

The remainder of the paper is organized as follows: We first present the basics of the algebra of relations in Section 2. Thereby, we also show how to model subsets and sets of subsets using relations and introduce two important choice operations. Based on relations, in Section 3 we formally develop three algorithms for the computation of cut completions which immediately can be formulated in the language of RelView. Section 4 deals with concept lattices. We show how to compute them with the help of the programs of Section 3 and how to obtain and draw their Hasse diagrams as labeled directed graphs using the RelView system. Section 5 contains some concluding remarks.

2 Relational Preliminaries

In this section we introduce some basics about the algebra of relations, show how to represent subsets and sets of subsets, and consider two choice operations on relations. We also have a short look at the RelView-system.

2.1 Algebra of Relations

We write $R : X \leftrightarrow Y$ if R is a relation with domain X and range Y, i.e., a subset of $X \times Y$. If the sets X and Y of R's *type* $X \leftrightarrow Y$ are finite and of cardinality m and n, respectively, we may consider R as a Boolean matrix with m rows and n columns. Since this Boolean matrix interpretation is well suited for many purposes, in the following we often use matrix terminology and matrix notation. Especially we speak of the rows and columns of R and write R_{xy} instead of $(x, y) \in R$. Boolean matrices are also used by the RelView system as a mean to depict relations.

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*), and the special relations O (*empty relation*), L (*universal relation*), and I (*identity relation*).

The basic operations and the special relations are very helpful for defining many properties of and further operations on relations in a component-free manner, that is, by formulae and expressions over relations involving no element relationship R_{xy}. The properties and operations used in the remainder of this paper are presented now (see e.g., [21] for more details).

A relation R is *reflexive* if $\mathsf{I} \subseteq R$, *irreflexive* if $R \subseteq \overline{\mathsf{I}}$, *transitive* if $RR \subseteq R$, *symmetric* if $R = R^\mathsf{T}$, and *antisymmetric* if $R \cap R^\mathsf{T} \subseteq \mathsf{I}$. An *equivalence relation* is a reflexive, transitive, and symmetric relation whereas a *partial order relation* is reflexive, transitive, and antisymmetric. By $ipa(R) := R \cap \overline{\mathsf{I}}$ we denote the *irreflexive part* of R. If R is a partial order relation, then $ipa(R)$ is irreflexive and transitive, i.e., a *strict order relation*. R is *univalent* if $R^\mathsf{T} R \subseteq \mathsf{I}$, *total* if $R\mathsf{L} = \mathsf{L}$, and *injective* if R^T is univalent. A univalent and total relation is a *mapping*.

The *left residual* of S over R is defined as $S \,/\, R := \overline{\overline{S}\, R^\mathsf{T}}$, while the *right residual* of R under S is given by $R \setminus S := \overline{R^\mathsf{T}\, \overline{S}}$. After translating the relation-algebraic expressions of the residuals into component-wise formulations we get that $(S \,/\, R)_{yx}$ holds if and only if R_{xz} implies S_{yz} for all z and $(R \setminus S)_{xy}$ holds if and only if R_{zx} implies S_{zy} for all z. By $syq(R, S) := (R \setminus S) \cap (R^\mathsf{T} \,/\, S^\mathsf{T})$ the *symmetric quotient* of R and S is defined. In component-wise notation this means that $syq(R, S)_{xy}$ holds if and only if R_{zx} is equivalent to S_{zy} for all z.

2.2 Modeling Sets and Sets of Sets with Relations

Relations offer some simple and elegant ways of representing subsets of a given set and sets of such subsets and for manipulating such objects.

The first representation of subsets uses *vectors* which are relations v with $v = v\mathsf{L}$. For $v : X \leftrightarrow Y$ this condition means: Whatever set Z and universal relation $\mathsf{L} : Y \leftrightarrow Z$ we choose, an element $x \in X$ is either in relationship $(v\mathsf{L})_{xz}$ to no element $z \in Z$ or to all elements $z \in Z$. As for a vector, therefore, the range is irrelevant, we consider in the following mostly vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\bot\}$ as range and omit in such cases the second subscript, that is, write v_x instead of $v_{x\bot}$. Such a vector can be considered as a Boolean matrix with exactly one column, i.e., as a Boolean column vector, and represents the subset $\{x \in X \mid v_x\}$ of X. A vector is said to be a *point* if it is non-empty and injective. These properties mean that it represents a singleton subset of its domain or an element from it if we identify a singleton set with the only element it contains. In the Boolean matrix model, hence, a point $v : X \leftrightarrow \mathbf{1}$ is a Boolean column vector in which exactly one component is 1.

We can also use injective mappings for representing subsets. Given an injective mapping $\imath : Y \leftrightarrow X$, we may consider Y as a subset of X by identifying it with its image under $\imath$. If Y is actually a subset of X and $\imath$ the identity mapping from Y to X, then the vector $\imath^{\mathsf{T}}\mathsf{L} : X \leftrightarrow \mathbf{1}$ represents Y as subset of X in the above sense. Clearly, the transition in the other direction, i.e., the generation of an injective mapping $inj(v) : Y \leftrightarrow X$ fulfilling $inj(v)_{yx}$ if and only if $y = x$ from a given vector $v : X \leftrightarrow \mathbf{1}$ representing the subset Y of X, is also possible.

As third possibility to deal with subsets of a given set X we will use in this paper the set-theoretic membership relation $\varepsilon : X \leftrightarrow 2^X$, defined by ε_{xY} if and only if $x \in Y$. Using matrix terminology, membership relations lead to a column-wise representation of sets of subsets. More specifically, if the vector $v : 2^X \leftrightarrow \mathbf{1}$ represents a subset $\mathcal{S}$ of 2^X in the above sense, then for all $x \in X$ and $Y \in \mathcal{S}$ we get the equivalence of $(\varepsilon\, inj(v)^{\mathsf{T}})_{xY}$ and $x \in Y$. This means that the elements of $\mathcal{S}$ are precisely represented by the columns of the relation $\varepsilon\, inj(v)^{\mathsf{T}} : X \leftrightarrow \mathcal{S}$.

If the subset $\mathcal{S}$ of the powerset 2^X is represented by the columns of a relation $R : X \leftrightarrow \mathcal{S}$ in the sense that R_{xY} is equivalent to $x \in Y$ for all $x \in X$ and $Y \in \mathcal{S}$, then the insertion of a new set Y into $\mathcal{S}$ can be modeled by adding to R a new column which consists of the vector representation $v : X \leftrightarrow \mathbf{1}$ of Y. For example, adding v to R from the right formally can be described as $R\,@\,v := (R^{\mathsf{T}} + v^{\mathsf{T}})^{\mathsf{T}}$, where $+$ is the relation-algebraic sum operation of the direct sum of the two sets $\mathcal{S}$ and $\{Y\}$ (cf. [5] for details).

2.3 Choice Operations

The choice of a point $point(v)$ contained in a non-empty vector v is fundamental for relational programming since it corresponds to the choice of an element from a non-empty set. An relation-algebraic axiomatization of $point(v)$ is obvious. Note that *point* is a (deterministic) function from non-empty vectors to points of the same type in the usual mathematical sense. Each call yields the same object so that especially $point(v) = point(v)$ holds. Of course, the axiomatization allows different realizations. E.g., the implementation of the function in RelView uses the fact that the system only deals with finite carrier sets which are linearly ordered by an internal enumeration. If the vector $v : X \leftrightarrow \mathbf{1}$ represents the subset Y of X, then the RelView version of $point(v) : X \leftrightarrow \mathbf{1}$ yields that point which represents its least element of Y with respect to the enumeration order.

The generalization of *point* to arbitrary non-empty relations is *atom*. A call $atom(R)$ of this function yields an atom contained in the non-empty relation R, i.e., a relation S such that $S \subseteq R$ and S contains exactly one pair. Relation-algebraically the latter property can be characterized by $S \neq \mathsf{O}$ and $S\mathsf{L}S^{\mathsf{T}} \cup S^{\mathsf{T}}\mathsf{L}S \subseteq \mathsf{I}$. The RelView version of $atom(R)$ yields that relation $S \subseteq R$ which contains the lexicographic least pair of R with respect to the internal enumeration order.

2.4 The RelView System

In the following we want to give an impression of RelView, a computer system for calculating with relations and relation-algebraic programming. More details and applications can e.g., be found in [2, 5, 8, 9].

In RelView all data are represented as relations, which the system visualizes in different ways. It offers several algorithms for pretty-printing a relation for which domain and range coincide as a directed graph. Alternatively, an arbitrary relation may be displayed as a Boolean matrix which is very useful for visual editing and also for discovering structural properties that are not evident from a graphical presentation. Because RelView, in particular when using it for the validation of relational specifications, often works on (very) large data, it uses a very efficient implementation of relations based on reduced ordered binary decision diagrams. Especially, this holds for the membership-relation $\varepsilon : X \leftrightarrow 2^X$, which in RelView is implemented via a binary decision diagram with $\mathcal{O}(|X|)$ nodes. Details can be found in [11, 18].

The RelView system can manage as many relations simultaneously as memory allows and the user may manipulate and analyze them by pre-defined operations and tests, relational functions, and relational programs. The pre-defined operations include all operations presented so far. All that can be accessed through simple mouse-clicks but also composed to functions and programs.

A declaration of a relational function is of the form $F(X_1, \dots, X_n) = t$, where F is the function name, the X_i, $1 \le i \le n$, are the formal parameters (standing for relations), and t is a relation-algebraic expression over the relations of the system's workspace that can additionally contain the formal parameters $X_i, 1 \le i \le n$. Recursion is not allowed.

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 Modula-2. It starts with a head line containing the program's name and a list of formal parameters. Then the declaration part follows, which consists of the declarations of local relational domains (direct products and sums), local relational functions, and local variables. The third part of a relational program is its body, a sequence of statements which are separated by semicolons and terminated by the return-clause. In programs recursive calls are possible.

We renounce at this place examples for relational functions and programs since many of them will be presented in the remainder of this paper.

3 Computation of Cut Completions

At the beginning of this section we recall the basic definitions of cut completion. More details can be found in [12] for example. Based on relations we then formally develop three algorithms for its computation. Each of them easily can be translated into the programming language of the RelView system.

3.1 Cut Completion

Assume (M, Q) to be a partially ordered set, i.e., $Q : M \leftrightarrow M$ to be a partial order relation. For a set $C \in 2^M$ let $Mi_Q(C) := \{x \in M \mid \forall\, y \in C : Q_{xy}\}$ denote the set of its lower bounds and $Ma_Q(C) := \{x \in M \mid \forall\, y \in C : Q_{yx}\}$ denote the set of its upper bounds with respect to Q. An element of the set

$$\mathcal{C}_Q := \{C \in 2^M \mid C = Mi_Q(Ma_Q(C))\}$$

is called a *(Dedekind) cut* of (M, Q). It is well-known that $(\mathcal{C}_Q, \subseteq)$ constitutes a complete lattice. Greatest lower bounds and least upper bounds are given by $C \sqcap D = C \cap D$ and $C \sqcup D = Mi_Q(Ma_Q(C \cup D))$, respectively.

The complete lattice $(\mathcal{C}_Q, \subseteq)$ is called the *(Dedekind-McNeille) cut completion* of the partially ordered set (M, Q) since it contains a sub-order which is order-isomorphic to (M, Q). This sub-order is given as $(\mathcal{P}_Q, \subseteq)$, where

$$\mathcal{P}_Q := \{[x] \mid x \in M\}$$

and $[x] := Mi_Q(Ma_Q(\{x\})) = \{y \in M \mid Q_{yx}\}$ is the *principal cut* generated by x. For $x, y \in M$ we have that Q_{yx} holds if and only if $[y] \subseteq [x]$. Hence, the isomorphism is established via the function $\sigma : M \to \mathcal{C}_Q$ defined by $\sigma(x) = [x]$.

It is also well-known that the cut completion is the least complete lattice containing a sub-order isomorphic to the partially ordered set (M, Q). Formally, the following theorem holds: If (M, Q) is isomorphic to a sub-order of a complete lattice $(V, \sqsubseteq)$ via the function $\rho : M \to V$, then also $(\mathcal{C}_Q, \subseteq)$ is isomorphic to a sub-order of $(V, \sqsubseteq)$ via the function $\psi : \mathcal{C}_Q \to V$ mapping a cut C to $\bigsqcup\{\rho(x) \mid x \in C\}$ and, furthermore, $\rho(x) = \psi(\sigma(x))$ holds for all $x \in M$.

Although the cut completion is the least complete lattice in which one can embed a partially ordered set (M, Q), the size of $\mathcal{C}_Q$ may be exponential in the size of M. For example, let $M := \{1, \ldots, 2 * n\}$ and Q_{xy} hold if and only if $x = y$ or $1 \leq x \leq n < y \leq 2 * n$ and $x + n \neq y$. Then Q is a partial order relation and the cut completion of (M, Q) is isomorphic to $(2^{\{1,\ldots,n\}}, \subseteq)$.

3.2 Use of Membership-Relations

In the following we develop a first relation-algebraic algorithm for computing the cut completion of a partially ordered set (M, Q). It is based on the membership relation $\varepsilon : M \leftrightarrow 2^M$ between M and its powerset 2^M.

We start with the definition that $x \in M$ is a lower bound of $C \in 2^M$ if and only if for all y from $y \in C$ it follows Q_{xy}. Then, we represent C by a vector $v : M \leftrightarrow \mathbf{1}$ and use the component-wise description of left residuals given in Section 2.1. We obtain that $Mi_Q(C)$ is represented by the vector $Q \,/\, v^\mathsf{T} : M \leftrightarrow \mathbf{1}$. Transposing the relation Q yields $Q^\mathsf{T} \,/\, v^\mathsf{T} : M \leftrightarrow \mathbf{1}$ as the vector representing $Ma_Q(C)$. Hence, we obtain the following two relation-algebraic descriptions for computing lower and upper bounds:

$$mi(Q, v) := Q \,/\, v^\mathsf{T} : M \leftrightarrow \mathbf{1} \qquad\qquad ma(Q, v) := Q^\mathsf{T} \,/\, v^\mathsf{T} : M \leftrightarrow \mathbf{1}$$

If the second argument of mi and ma is not a vector but an arbitrary relation $R : M \leftrightarrow Y$, then obviously they compute lower and upper bounds column-wise. This means e.g., that for all $x \in M$ and $y \in Y$ the relationship $mi(Q, R)_{xy}$ is equivalent to x being a lower bound of the set $\{z \in M \mid R_{zy}\}$.

Our next aim is to develop a vector which represents the set $\mathcal{C}_Q$. Using besides ε the identity relation $\mathsf{I} : 2^M \leftrightarrow 2^M$ und the vector $\mathsf{L} : 2^M \leftrightarrow \mathbf{1}$ in combination with well-known correspondences between certain kinds of logical and relation-algebraic constructions and the component-wise description of symmetric quotients of Section 2.1, we are able to calculate for all $C \in 2^M$ as follows:

$$\begin{aligned}
C \;\text{ cut of } (M, Q) &\iff \forall\, x : x \in C \leftrightarrow x \in Mi_Q(Ma_Q(C)) \\
&\iff \forall\, x : \varepsilon_{xC} \leftrightarrow mi(Q, ma(Q, \varepsilon))_{xC} \\
&\iff \exists\, D : \forall\, x : \varepsilon_{xC} \leftrightarrow mi(Q, ma(Q, \varepsilon))_{xD} \wedge C = D \\
&\iff \exists\, D : syq(\varepsilon, mi(Q, ma(Q, \varepsilon)))_{CD} \wedge \mathsf{I}_{CD} \wedge \mathsf{L}_D \\
&\iff (syq(\varepsilon, mi(Q, ma(Q, \varepsilon))) \cap \mathsf{I})\mathsf{L}_C
\end{aligned}$$

If we remove the subscript C from the last expression of this equivalence by function abstraction, we get the following relation-algebraic description of the vector representing the set of cuts:

$$cutvect(Q) := (syq(\varepsilon, mi(Q, ma(Q, \varepsilon))) \cap \mathsf{I})\mathsf{L} : 2^M \leftrightarrow \mathbf{1}$$

Now we apply the injective mapping generated by $cutvect(Q)$. As shown in Section 2.2, the columns of the following relation represent the set of cuts of (M, Q), i.e., for all $x \in M$ and $C \in \mathcal{C}_Q$ we have $x \in C$ if and only if $cutset(Q)_{xC}$:

$$cutset(Q) := \varepsilon\, inj(cutvect(Q))^\mathsf{T} : M \leftrightarrow \mathcal{C}_Q$$

The set of cuts is ordered by set inclusion. Using the component-wise description of right residuals of Section 2.1 in combination with the equivalence of $x \in C$ and $cutset(Q)_{xC}$, we are able to show for all $C, D \in \mathcal{C}_Q$ the following property:

$$\begin{aligned} C \subseteq D &\iff \forall\, x : x \in C \to x \in D \\ &\iff \forall\, x : cutset(Q)_{xC} \to cutset(Q)_{xD} \\ &\iff (cutset(Q) \setminus cutset(Q))_{CD} \end{aligned}$$

An immediate consequence of this calculation is the subsequent relation-algebraic description of the inclusion order on the set of cuts:

$$cutord(Q) := cutset(Q) \setminus cutset(Q) : \mathcal{C}_Q \leftrightarrow \mathcal{C}_Q$$

As the last step it remains to describe the embedding of (M, Q) into its cut completion, i.e., the function $\sigma : M \to \mathcal{C}_Q$ of Section 3.1, as a relation between M and $\mathcal{C}_Q$. Given $x \in M$ and $C \in \mathcal{C}_Q$, we obtain

$$\begin{aligned} [x] = C &\iff \forall\, y : y \in [x] \leftrightarrow y \in C \\ &\iff \forall\, y : Q_{yx} \leftrightarrow cutset(Q)_{yC} \\ &\iff syq(Q, cutset(Q))_{xC} \end{aligned}$$

due to $[x] = \{y \in M \mid Q_{yx}\}$ and the component-wise descriptions of the relation $cutset(Q)$ and the symmetric quotient construction, respectively. This leads to the following relation-algebraic version of the function σ of Section 3.1:

$$embedding(Q) := syq(Q, cutset(Q)) : M \leftrightarrow \mathcal{C}_Q$$

All the previous relation-algebraic descriptions are algorithms if we assume that (M, Q) is a finite partially ordered set. They can immediately be translated into the language of RELVIEW. Doing so, we get:

```
mi(Q,v) = Q / v^.

ma(Q,v) = Q^ / v^.

cutvect(Q)
  DECL E, I
  BEG  E = epsi(On1(Q));
       I = I(O1n(E)^ * O1n(E))
       RETURN dom(syq(E,mi(Q,ma(Q,E))) & I)
  END.

cutset(Q) = epsi(On1(Q)) * inj(cutvect(Q))^.
```

```
cutord(Q)
  DECL C
  BEG  C = cutset(Q)^
       RETURN C \ C
  END.
embedding(Q) = syq(Q,cutset(Q)).
```

After loading it from a file, this RelView-code can be executed via the system as it stands and after that the results are depicted on the system's screen. In the case of `cutvect` and `cutord` the code uses relational programs instead of relational functions to avoid multiple evaluations of relation-algebraic expressions.

3.3 Inductive Generation of the Set of Cuts

The column-wise enumeration of the set of cuts is the decisive step of the cut completion procedure since from it one immediately obtains the inclusion order and the embedding mapping. A serious drawback of the solution of Section 3.2 is the use of the membership relation. It leads to an exponential complexity even if the size of $\mathcal{C}_Q$ is polynomial in the size of M. In this section we develop a solution which avoids this drawback.

Guided by the description of the greatest lower bound of two cuts as their intersection, we start with the set $\mathcal{I}_Q$ inductively generated by the principal cuts $\mathcal{P}_Q$ as base and intersection as only generating function. I.e., we consider:

$$\mathcal{I}_Q := \bigcap \{X \in 2^{\mathcal{C}_Q} \mid \mathcal{P}_Q \cup \{C \cap D \mid C, D \in X\} \subseteq X\}$$

By this definition $(\mathcal{I}_Q, \subseteq)$ is the least complete lower semi-lattice which contains $\mathcal{P}_Q$, i.e., a sub-order isomorphic to (M, Q). Now we distinguish two cases: If the set $\mathcal{I}_Q$ contains the largest cut M, then $(\mathcal{I}_Q, \subseteq)$ is even the least complete lattice containing a sub-order isomorphic to (M, Q). Hence, it is the cut completion of (M, Q) because of the theorem mentioned in Section 3.1. Otherwise, the same argument shows that inserting M into $\mathcal{I}_Q$ results in the cut completion of (M, Q).

The next step of the algorithm development applies Tarski's well-known fixed point theorem (see [22]). It shows that $\mathcal{I}_Q$ is the least fixed point of the following monotone function:

$$F : 2^{\mathcal{C}_Q} \to 2^{\mathcal{C}_Q} \qquad\qquad F(X) = \mathcal{P}_Q \cup \{C \cap D \mid C, D \in X\}$$

If M is a finite set, the least fixed point of F coincides with the greatest element of the finite chain $\emptyset \subset F(\emptyset) \subset F^2(\emptyset) \subset \ldots$ which leads to the subsequent algorithm for computing the set of cuts as final value of X:

$$
\begin{array}{l}
X := \mathcal{P}_Q; \\
R := \{(C,D) \in X \times X \mid C \cap D \notin X\}; \\
\textbf{while } R \neq \mathsf{O} \textbf{ do} \\
\quad X := X \cup \{C \cap D \mid R_{CD}\}; \\
\quad R := \{(C,D) \in X \times X \mid C \cap D \notin X\} \textbf{ od}; \\
\textbf{if } M \notin X \textbf{ then } X := X \cup \{M\} \textbf{ fi}
\end{array}
$$

Correctness follows from the fact that a simple induction shows $F^{i+1}(\emptyset) = G^i(\mathcal{P}_Q)$ for all $i \geq 0$, where the monotone function G is defined as

$$G : 2^{\mathcal{C}_Q} \to 2^{\mathcal{C}_Q} \qquad\qquad G(X) = X \cup \{C \cap D \mid C, D \in X\},$$

and, furthermore, the equation

$$G(X) = X \cup \{C \cap D \mid (C,D) \in X \times X, C \cap D \notin X\} = X \cup \{C \cap D \mid R_{C,D}\}$$

holds for all $X \in 2^{\mathcal{C}_Q}$. Finally, we transform this algorithm step by step into a relation-algebraic version. Especially this means that we take X as a relation whose columns represent in each run of the loop a certain subset $\mathcal{S}$ of the powerset 2^M as described at the end of Section 2.2.

The principal cuts are exactly represented by the columns of the partial order relation Q. Hence, interpreting X as a relation we have to change line 1 of the above algorithm into the assignment

$$X := Q.$$

To obtain a relation-algebraic version of the computation of R from X, i.e., of lines 2 and 5 of the above algorithm, we apply the so-called *tupling* (sometimes also called *fork*) $[X, X] : M \leftrightarrow \mathcal{S} \times \mathcal{S}$ of X, defined by $[X, X]_{x(C,D)}$ if and only if X_{xC} and X_{xD}. Combining it with the component-wise description of symmetric quotients, we are able to calculate for all $C, D, Z \in \mathcal{S}$ as follows:

$$
\begin{array}{rcl}
C \cap D = Z & \iff & \forall\, x : x \in C \wedge x \in D \leftrightarrow x \in Z \\
& \iff & \forall\, x : X_{xC} \wedge X_{xD} \leftrightarrow X_{xZ} \\
& \iff & \forall\, x : [X, X]_{x(C,D)} \leftrightarrow X_{xZ} \\
& \iff & syq([X, X], X)_{(C,D)Z}
\end{array}
$$

Now we use – considered as relations – the projections π and ρ from the direct product $\mathcal{S} \times \mathcal{S}$ to its first and second component, respectively. Then from the above calculation we obtain for all $C, D \in \mathcal{S}$ the following equivalence:

$$\begin{aligned}
\exists\, Z : C \cap D = Z &\iff \exists\, Z : syq([X,X],X)_{(C,D)Z} \\
&\iff \exists\, P : \pi_{PC} \wedge \rho_{PD} \wedge \exists\, Z : syq([X,X],X)_{PZ} \wedge \mathsf{L}_{ZD} \\
&\iff \exists\, P : \pi^{\mathsf{T}}_{CP} \wedge (\rho \cap syq([X,X],X)\mathsf{L})_{PD} \\
&\iff (\pi^{\mathsf{T}}(\rho \cap syq([X,X],X)\mathsf{L}))_{CD}
\end{aligned}$$

A removal of the indices of the last expression followed by a negation, finally, leads to the relation-algebraic version of the computation of the relation $R : \mathcal{S} \leftrightarrow \mathcal{S}$ from the relation $X : M \leftrightarrow \mathcal{S}$ in form of an assignment as follows:

$$R := \overline{\pi^{\mathsf{T}}(\rho \cap syq([X,X],X)\mathsf{L})}$$

The relation-algebraic version of line 4 of the above algorithm consists of a few lines and can be developed rather straightforwardly. First, we store the value of X in a new variable, say V. By means of a simple loop we then take every pair (C, D) of R via a call of the operation *atom*, compute the vector representation v of the intersection $C \cap D$, and add v as a column to V if it is not yet a column of it. Of course, taking the pair (C, D) allows to neglect the reverse pair (D, C). After the loop X becomes the value of V.

Let the relation A consist exactly of the pair $(C, D) \in R$. Taking the universal vector $\mathsf{L} : M \leftrightarrow \mathbf{1}$, a little reflection shows that $XA\mathsf{L}$ is the column of X representing C and $XA^{\mathsf{T}}\mathsf{L}$ is the column of X representing D. Hence, the vector representation v of the intersection $C \cap D$ is computed by the following assignment:

$$v := XA\mathsf{L} \cap XA^{\mathsf{T}}\mathsf{L}$$

It is added as a new column to V from the right by

$$\textbf{if } syq(V,v) = \mathsf{O} \textbf{ then } V := (V^{\mathsf{T}} + v^{\mathsf{T}})^{\mathsf{T}} \textbf{ fi}.$$

The guard $syq(V, v) = \mathsf{O}$ of this conditional follows from the fact that $syq(V, v)$ is empty if and only if the vector v is not a column of the relation V (see the component-wise description of symmetric quotients given in Section 2.1) and its body already has been elaborated in Section 2.2.

It remains to treat line 6. Here we use that the set M is represented by the universal vector $\mathsf{L} : M \leftrightarrow \mathbf{1}$. Hence, the relation-algebraic version

$$\textbf{if } syq(X,\mathsf{L}) = \mathsf{O} \textbf{ then } X := (X^{\mathsf{T}} + \mathsf{L}^{\mathsf{T}})^{\mathsf{T}} \textbf{ fi}$$

of the last line of the above algorithm is an immediate consequence of the just presented test for column containment, too.

Now we are done and can combine the single developed pieces of code to the desired new relation-algebraic computation of the set of cuts. In the language of the RelView system this computation looks as follows:

```
haveint(X)
  DECL Prod = PROD(X^*X,X^*X);
       pi, rho
  BEG  pi = p-1(Prod);
       rho = p-2(Prod)
       RETURN pi^ * (rho & syq([X,X],X) * L(X^*X))
  END.

cutset(Q)
  DECL L, X, R, V, W, A, v
  BEG  L = Ln1(Q);
       X = Q;
       R = -haveint(X);
       WHILE -empty(R) DO
         V = X;
         W = R;
         WHILE -empty(W) DO
           A = atom(W);
           v = X * dom(A) & X * dom(A^);
           IF empty(syq(V,v)) THEN V = (V^+v^)^ FI;
           W = W & -A & -A^ OD;
         X = V;
         R = -haveint(X) OD;
       IF empty(syq(X,L)) THEN X = (X^+L^)^ FI
       RETURN X
  END.
```

To enhance readability, this code uses an an auxiliary program `haveint` for computing the "have an intersection relation" $\pi^{\mathsf{T}}(\rho \cap syq([X,X],X)\mathsf{L})$ from X.

3.4 Lectic Enumeration of the Set of Cuts

As we have already mentioned in the introduction, in [15] a remarkable simple algorithm for computing cut completions and concept lattices is presented. The aim of this section is to demonstrate how its decisive part easily can be formulated in RelView to compute the set of cuts of a partially ordered set (M, Q) with the help of the system.

In [15] it is assumed that the carrier set is enumerated. Using relations this means that we have besides the partial order relation $Q : M \leftrightarrow M$ a strict order relation $S : M \leftrightarrow M$ with the additional property $\overline{\mathsf{I}} \subseteq S \cup S^{\mathsf{T}}$ of *linearity*. Then a first result of [15] is that the relation $L : \mathcal{C}_Q \leftrightarrow \mathcal{C}_Q$, defined by

$$L_{CD} \ :\Longleftrightarrow\ \exists\, x : \mathit{less}(C, x, D, S),$$

is a linear strict order relation (called the *lectic order*) if the predicate *less* for $C, D \in 2^M$ and $x \in M$ is given as

$$\mathit{less}(C, x, D, S) \ :\Longleftrightarrow\ x \in D \setminus C \wedge \{y \in C \mid S_{yx}\} = \{y \in D \mid S_{yx}\}.$$

The lectic least cut is $\mathit{Mi}_Q(\mathit{Ma}_Q(\emptyset))$ and the lectic greatest cut is M. Hence, the task of computing $\mathcal{C}_Q$ can be reduced to a simple loop which starts with a list containing $\mathit{Mi}_Q(\mathit{Ma}_Q(\emptyset))$ as the only element and inserts in every run the *lectic upper neighbour* of the last list element C, i.e., the least cut that is greater than C with respect to L, until M is inserted. Using the operation

$$\oplus : \mathcal{C}_Q \times M \to 2^M \qquad\qquad C \oplus x := \mathit{Mi}_Q(\mathit{Ma}_Q(\{y \in C \mid S_{yx}\} \cup \{x\})),$$

in [15] it is shown that $C \oplus x$ is the lectic upper neighbour of C if x is the greatest element of the set $\{y \notin C \mid \mathit{less}(C, y, C \oplus y, S)\}$ with respect to S.

Now we turn to relational algebra. Let a subset C of M be represented by the vector $v : M \leftrightarrow \mathbf{1}$ and an element x of M by the point $p : M \leftrightarrow \mathbf{1}$. Then the vector $v \cap Sp : M \leftrightarrow \mathbf{1}$ is the representation of $\{y \in C \mid S_{yx}\}$. This immediately leads to the following relation-algebraic version of the predicate *less*:

$$\mathit{less}(v, p, w, S) \iff p \subseteq w \cap \overline{v} \wedge v \cap Sp = w \cap Sp$$

Using the functions *mi* and *ma* of Section 3.2, the subsequent relation-algebraic version of the operation $\oplus$ (which makes visible that it depends besides the set representation v and the element representation p also on the two relations Q and S) is an immediate consequence of the original definition, too:

$$\mathit{plus}(v, p, Q, S) = \mathit{mi}(Q, \mathit{ma}(Q, (v \cap Sp) \cup p)) : M \leftrightarrow \mathbf{1}$$

It remains to develop a relation-algebraic algorithm which yields for a vector v representing a cut C the vector representation w of the lectic upper neighbour of C. To obtain this goal, we use the simple fact that for all vectors $u : M \leftrightarrow \mathbf{1}$ the greatest element (with respect to S) of the set it represents is represented by the point $u \cap \mathit{ma}(S \cup \mathsf{I}, u)$. Then the algorithm

$$
\begin{array}{l}
u := \overline{v}\,;\\
p := u \cap \mathit{ma}(S \cup \mathsf{I}, u);\\
w := \mathit{plus}(v, p, Q, S);\\
\textbf{while } \neg \mathit{less}(v, p, w, S) \textbf{ do}\\
\quad u := u \cap \overline{p}\,;\\
\quad p := u \cap \mathit{ma}(S \cup \mathsf{I}, u);\\
\quad w := \mathit{plus}(v, p, Q, S) \textbf{ od},
\end{array}
$$

starting with a point representing the greatest element of the complement of C, is a straightforward implementation of the search for the greatest point $p \subseteq \overline{v}$ for which $less(v, p, plus(v, p, Q, S), S)$ holds.

Summing up and changing to RelView we obtain the following new code for computing a relation the columns of which represent the set of cuts. Its main program `cutset` has a second parameter for the linear strict order relation S.

```
less(v,p,w,S) =
  incl(p,w & -v) & eq(v & S * p,w & S * p).
plus(v,p,Q,S) =
  mi(Q,ma(Q, (v & S * p) | p)).
nextcut(v,Q,S)
  DECL I, u, p, w
  BEG  I = I(S);
       u = -v;
       p = u & ma(S | I,u);
       w = plus(v,p,Q,S);
       WHILE -less(v,p,w,S) DO
         u = u & -p;
         p = u & ma(S | I,u);
         w = plus(v,p,Q,S) OD
       RETURN w
  END.
cutset(Q,S)
  DECL L, X, v
  BEG  L = Ln1(Q);
       X = mi(Q,ma(Q,On1(Q)));
       v = nextcut(X,Q,S);
       X = (X^ + v^)^;
       WHILE -eq(v,L) DO
         v = nextcut(v,Q,S);
         X = (X^ + v^)^ OD
       RETURN X
  END.
```

It should be mentioned that `cutset` requires the precondition $|\mathcal{C}_Q| > 1$ for being correct. A version which works correctly for all cut sets can be obtained by an additional treatment of the case $mi(Q, ma(Q, \mathsf{O})) = \mathsf{L}$.

4 Computation of Concept Lattices

This section deals with concept lattices. First, we introduce the basics. Then, we show how concept lattices can be computed with the help of the programs of Section 3. Finally, we demonstrate how to obtain and draw their Hasse diagrams as labeled directed graphs using the RelView system.

4.1 Concept Lattices

Concept lattices are generalizations of cut completions. One starts with *(formal) contexts* which are triples (O, A, I) consisting of a non-empty set O of *objects*, a non-empty set A of *attributes*, and an *incidence relation* $I : O \leftrightarrow A$. A *(formal) concept* then is a pair (C, D) from $2^O \times 2^A$ such that the equations $C = D^\triangledown$ and $D = C^\vartriangle$ hold. Here $D^\triangledown$ and $C^\vartriangle$ are defined in analogy to the sets of lower and upper bounds in Section 3.1, that is, we have $D^\triangledown := \{x \in O \mid \forall\, y \in D : I_{xy}\}$ and $C^\vartriangle := \{x \in A \mid \forall\, y \in C : I_{yx}\}$.

If (C_1, D_1) and (C_2, D_2) are two concepts, then (C_1, D_1) is defined to be *less general or equal* than (C_2, D_2), denoted by $(C_1, D_1) \sqsubseteq (C_2, D_2)$, if $C_1 \subseteq C_2$ or, equivalently, $D_2 \subseteq D_1$. With this relation the set

$$\mathcal{B}_I := \{(C, D) \in 2^O \times 2^A \mid C = D^\triangledown \wedge D = C^\vartriangle\}$$

of all concepts constitutes a complete lattice $(\mathcal{B}_I, \sqsubseteq)$, in [17] called *concept lattice* ([1] speaks of a *Galois lattice*). By $(C_1, D_1) \sqcap (C_2, D_2) = (C_1 \cap C_2, (D_1 \cup D_2)^{\triangledown\vartriangle})$ greatest lower bounds are given; $(C_1, D_1) \sqcup (C_2, D_2) = ((C_1 \cup C_2)^{\vartriangle\triangledown}, D_1 \cap D_2)$ describes least upper bounds.

4.2 Adaptation of the Previous Algorithms

As an immediate consequence of Section 4.1 we obtain that the concept lattice $(\mathcal{B}_I, \sqsubseteq)$ of a context (O, A, I) is isomorphic to the complete lattice $(\mathcal{O}_I, \subseteq)$, with the carrier set defined as $\mathcal{O}_I := \{C \in 2^O \mid \exists\, D : (C, D) \in \mathcal{B}_I\}$, and also to the complete lattice $(\mathcal{A}_I, \supseteq)$, where $\mathcal{A}_I := \{D \in 2^A \mid \exists\, C : (C, D) \in \mathcal{B}_I\}$.

Fundamental for the following is the simple fact that $D^\triangledown$ and $C^\vartriangle$ generalize the notions of lower bounds and upper bounds from partial order relations to arbitrary relations. Using also the notations $Mi_I(D)$ and $Ma_I(C)$ of Section 3.1 instead of $D^\triangledown$ and $C^\vartriangle$ we, furthermore, get for each $C \in 2^O$:

$$\begin{aligned} C \in \mathcal{O}_I &\iff \exists\, D : (C, D) \in \mathcal{B}_I \\ &\iff \exists\, D : Ma_I(C) = D \wedge Mi_I(D) = C \\ &\iff C = Mi_I(Ma_I(C)) \end{aligned}$$

The last line of this equivalence is exactly the defining equation for a set to be a cut. Hence, the lattice $(\mathcal{O}_I, \subseteq)$ generalizes the construction of a cut completion from a partial order relation to an arbitrary (incidence) relation.

As an immediate consequence, the two calls `cutset(I)` and `cutord(I)` of the RelView-programs of Section 3.2 compute for a context (O, A, I) the column-wise representation of the set $\mathcal{O}_I$ and the inclusion order on this set, respectively. The column-wise representation of the set $\mathcal{A}_I$ is obtained as value of the RelView-expression `ma(I, cutset(I))`. This follows from the property $\mathcal{A}_I = \{Ma_I(C) \mid C \in \mathcal{O}_I\}$ and the fact that for an arbitrary second argument the RelView function `ma` computes upper bounds column-wise. From the value of `ma(I, cutset(I))`, finally, the reverse inclusion order on $\mathcal{A}_I$ can be obtained with the help of RelView's right residual and transposition operations.

Instead of the RelView-program `cutset` of Section 3.2 also the corresponding program of Section 3.4 can be used for computing the set $\mathcal{O}_I$ since the algorithm it implements originally was designed for the lectic enumeration of all object-components of the elements of $\mathcal{B}_I$, i.e., the set $\mathcal{O}_I$ (see [15, 17]).

A call `cutset(I)` with the version of Section 3.3, however, only leads to a correct result if all columns of the incidence relation I are pair-wise different (otherwise the result is not antisymmetric). A context or incidence relation with this property is called *column-clarified*. In [17] it is proved that column-clarification is without any affect on the structure of the concept lattice. Hence, to ensure correctness also for the version of Section 3.3 we only have to remove all duplicates of columns from the incidence relation in a preparatory step.

To remove all duplicates of columns from an arbitrary relation $R : X \leftrightarrow Y$, we start with the relation $S := syq(R, R)$. From the component-wise description of symmetric quotients in Section 2.1 we obtain for all $x, y \in Y$ that S_{xy} holds if and only if the x-column and the y-column of R are equal. Furthermore, $S : Y \leftrightarrow Y$ is an equivalence relation (see e.g., [21] for a relation-algebraic proof). The column-wise representation of the set Y/S of its equivalence classes is the canonical epimorphism, considered as a relation $\Phi : Y \leftrightarrow Y/S$. Now it is obvious that the step from $R : X \leftrightarrow Y$ to $R\Phi : X \leftrightarrow Y/S$ removes all duplicates of columns from R.

In [5] the following RelView-program for the column-wise representation of the equivalence classes of an equivalence relation S formally is derived:

```
classes(S)
  DECL C, v, p
  BEG  p = point(Ln1(S));
       C = S * p;
       v = -C;
       WHILE -empty(v) DO
         p = point(v);
         C = (C^ + (S * p)^)^;
         v = v & -(S * p) OD
       RETURN C
  END.
```

Combining the RelView-program `classes` with the RelView-program `cutset` of Section 3.3 and RelView's pre-defined operation `syq` for symmetric quotients, the evaluation of the RelView-expression `cutset(I * classes(syq(I,I)))` first column-clarifies I, say to I_r, and then computes the column-wise representation of the set $\mathcal{O}_{I_r}$. Instead with single attributes the columns of this relation are labeled with sets of attributes, viz. the equivalence classes of $syq(I, I)$. This is in accordance with the usual labeling of clarified incidence relations; see [17].

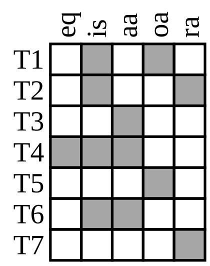
4.3 Drawing Concept Lattices With RelView

In many applications of context analysis experts learn from formal contexts by inspecting their carefully layouted concept lattices. Concept lattices of such contexts tend to have a modest size. Otherwise their drawings would be hard to analyze visually. In this section we demonstrate how the RelView system can be used for depicting such concept lattices as labeled directed graphs. We do this by means of a small example, taken from [16].

The object set O of the context (O, A, I) of the example consists of the seven triangles T1, …, T7 given in the following tabular, where each triangle is represented by the coordinates of its three corners in the Euclidian plane:

abbreviation	coordinates
T1	(0,0) (6,0) (3,1)
T2	(0,0) (1,0) (1,1)
T3	(0,0) (4,0) (1,2)
T4	(0,0) (2,0) (1,$\sqrt{3}$)
T5	(0,0) (2,0) (5,1)
T6	(0,0) (2,0) (1,3)
T7	(0,0) (2,0) (0,1)

As set A of attributes we consider five standard properties that triangles may or may not have, viz. equilateral (eq), isoceles (is), acute angled (aa), obtuse angled (oa), and right angled (ra). The following picture, finally, shows the RelView-representation of the incidence relation $I : O \leftrightarrow A$ of the context (O, A, I):

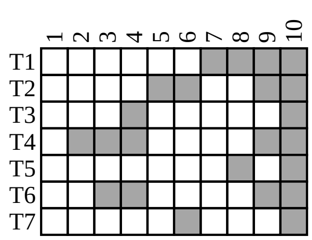


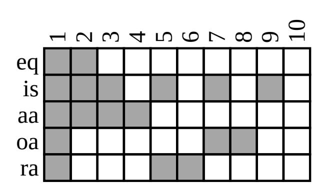
In this labeled Boolean matrix a grey square stands for the entry "true" and a white square stands for the entry "false":

To obtain the concept lattice of the triangles context, in a first step we apply the RelView system and compute two relations

$$B^O : O \leftrightarrow \mathcal{O}_I \qquad\qquad B^A : A \leftrightarrow \mathcal{A}_I$$

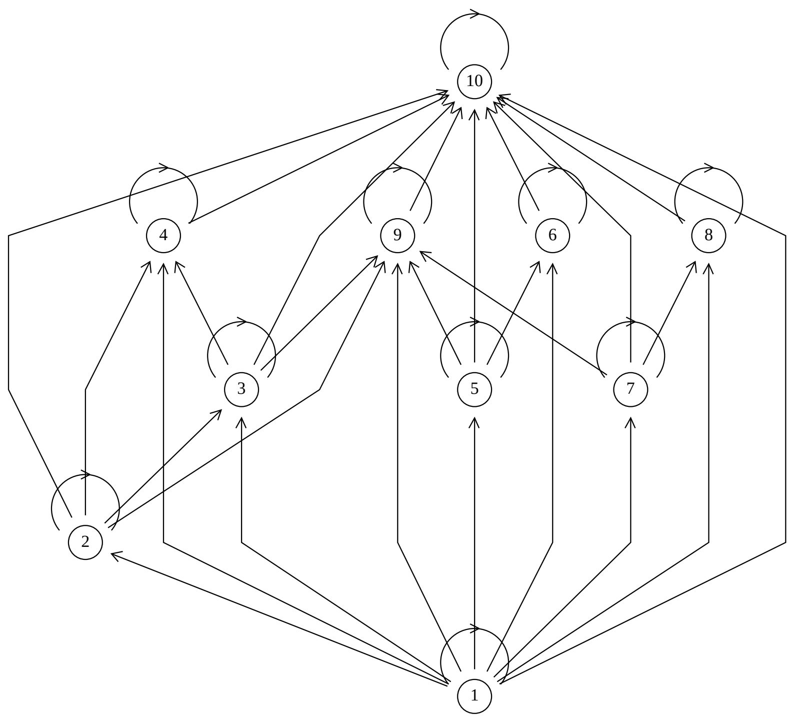
the columns of which represent the set $\mathcal{O}_I$ and the set $\mathcal{A}_I$, respectively. If we use the program `cutset` of Section 3.2 for the computation of B^O and the method of Section 4.2 to get B^A from B^O, we obtain as results the Boolean matrices shown in the following two pictures:





Hence, the triangles context leads to ten concepts. From the first columns of the matrices we obtain $(\emptyset, A)$ as first context, the second context $(\{\mathrm{T4}\}, \{\mathrm{eq}, \mathrm{is}, \mathrm{aa}\})$ is represented by the second columns and so on.

As mentioned before, the concept lattice $(\mathcal{B}_I, \sqsubseteq)$ is isomorphic to the complete lattice $(\mathcal{O}_I, \subseteq)$ and the inclusion order of the latter relation-algebraically can be described as right residual $B^O \setminus B^O$. We renounce the RelView-representation of the inclusion order as Boolean matrix. Instead we show in the next picture how the system depicts this relation as a directed graph using the hierarchical graph drawing algorithm of [14].



Such a layout of a partial order relation R is neither economic nor easy to comprehend since it contains many superfluous arcs. Therefore, it is customary to depict only the upper neighbour relationship given by R, which goes by the name *Hasse-diagram* H_R of R. In [21] it is shown that $H_R = S \cap \overline{SS}$ with S being the irreflexive part of R. A translation of this description into the language of RELVIEW is trivial and leads to the following code:

```
ipa(R) = R & -I(R).

hasse(R)
  DECL S
  BEG  S = ipa(R)
       RETURN S & -(S * S)
  END.
```

Having depicted the Hasse-diagram of a concept lattice as a "nice" directed graph with the aid of the RELVIEW system and its graph drawing features, it remains to label the vertices of the drawn graph using the system's labeling mechanism. However, labeling all vertices with the elements of the corresponding concepts would lead to a diagram which is difficult to understand because of the lot of labels. To enhance readability, therefore, in the literature it has become customary to label the vertex corresponding to a set $C \in \mathcal{O}_I$ with $x \in O$ precisely if C is a minimal set (with respect to inclusion) that contains x, that is, $x \in C$ holds and there exists no $D \in \mathcal{O}_I$ such that $D \subset C$ and $x \in D$. See [17] for example. Using the relation $B^O : O \leftrightarrow \mathcal{O}_I$ as column-wise representation of the set $\mathcal{O}_I$ and the simple facts that $x \in C$ (respectively $x \in D$) is equivalent to B^O_{xC} (respectively B^O_{xD}) and $D \subseteq C$ is equivalent to $(B^O \setminus B^O)_{DC}$, the little calculation

$$\begin{aligned}
& x \in C \wedge \neg\exists\, D : D \subset C \wedge x \in D \\
\iff\ & B^O_{xC} \wedge \neg\exists\, D : (B^O \setminus B^O)_{DC} \wedge \bar{\mathsf{I}}_{DC} \wedge B^O_{xD} \\
\iff\ & B^O_{xC} \wedge \overline{B^O((B^O \setminus B^O) \cap \bar{\mathsf{I}})}_{xC} \\
\iff\ & (B^O \cap \overline{B^O\, ipa(B^O \setminus B^O)}\,)_{xC}
\end{aligned}$$

shows that the vertex corresponding to C is labeled with x if and only if (x, C) is contained in the following *object labeling relation*:

$$B^O \cap \overline{B^O\, ipa(B^O \setminus B^O)} : O \leftrightarrow \mathcal{O}_I$$

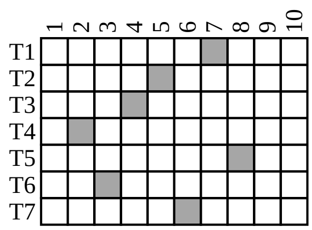
Based on a computation of the set $\mathcal{O}_I$ via a call `cutset(I)` it is obvious how to compute the object labeling relation by a RelView-program. Here is the code:

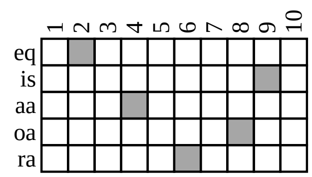
```
objlabeling(I)
  DECL BO
  BEG  BO = cutset(I)
       RETURN BO & -(BO * ipa(BO \ BO))
  END.
```

Attribute labels are saved by labeling a vertex corresponding to $C \in \mathcal{A}_I$ with $x \in A$ if and only if C is a maximal set (with respect to inclusion) that contains x. The latter means that $x \in C$ and there exists no proper superset D of C which contains x. Again it is rather trivial to develop from this description a RelView-program for computing the corresponding *attribute labeling relation*. We get:

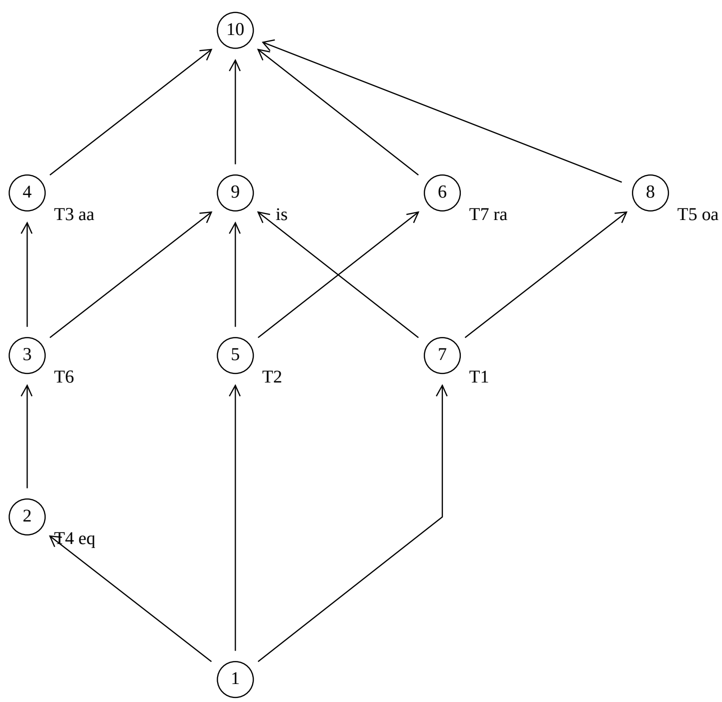
```
attrlabeling(I)
  DECL BO
  BEG  BO = cutset(I)
       RETURN BO & -(BO * ipa(BO \ BO)^)
  END.
```

The following two RelView-matrices show the object labeling relation and the attribute labeling relation for the triangles context:





If we label the hierarchical drawing of the Hasse-diagram of the triangles concept lattice according to these two relations, then the RelView system, finally, shows on its graph window the following picture:



Of course, the user of RelView interactively can change this drawing using the mouse, if it does not satisfy his aesthetical demands.

5 Conclusion

In this paper we have shown how relational algebra and the RelView system can be used for computing and depicting cut completions and concept lattices. We have tested the approach with many examples to see how the programs perform in practice. To give an example, on a modern PC (Pentium, running Linux at 1.2 GHz) we needed 76 seconds to compute the vector representation of the set $\mathcal{O}_{\bar{\mathsf{I}}}$ for a context of the specific form $(O, O, \bar{\mathsf{I}})$ with 19 objects using the program `cutvect` of Section 3.2. To get for the same context the relation of the concept lattice using the program `cutset` of Section 3.2 takes 258 seconds. It should be remarked that a so-called diagonal context $(O, O, \bar{\mathsf{I}})$ constitutes a worst case since its concept lattice is isomorphic to $(2^O, \subseteq)$; see [17].

Besides the computation of concept lattices there are many further algorithmic tasks in formal concept analysis and their relation-algebraic solutions are subject of current and future research. How to column-clarify a context has already been demonstrated in Section 4.2 and row-clarification is obtained by column-clarifying the transpose of the incidence relation. Also reduction of contexts has already been solved by combining the relation-algebraic descriptions

$$\nearrow = \overline{I} \cap \overline{\overline{I}\,((I \setminus I)^{\mathsf{T}} \cap I^{\mathsf{T}}\,\overline{I}\,)} \qquad\qquad \swarrow = \overline{I} \cap \overline{((\overline{I}\;/\;\overline{I}\,) \cap \overline{I}\,I^{\mathsf{T}})\,\overline{I}}$$

of the arrow relations $\nearrow : O \leftrightarrow A$ and $\swarrow : O \leftrightarrow A$ (which play a decisive role in reduction; see [17]) with the operation *inj* for generating injective mappings from vectors. Presently, we work on the relational-algebraic modeling of the composition of concept lattices using direct products and sums.

Acknowledgement. I wish to thank B. Ganter, G. Schmidt and M. Winter for valuable discussions on the topic of the paper.

References

1. M. Barbut and B. Monjardet, Ordre et Classification: Algèbre et Combinatoire, (Hachette, Paris, 1970).
2. R. Behnke, R. Berghammer, E. Meyer and P. Schneider, RELVIEW — A System for Calculation with Relations and Relational Programming, in: E. Astesiano, ed., Proc. 1st Conf. *Fundamental Approaches to Software Engineering*, LNCS 1382 (Springer, Berlin, 1996), 318-321.
3. R. Berghammer, Combining Relational Calculus and the Dijkstra-Gries Method for Deriving Relational Programs, Information Sci. 119 (1999), 155-171.
4. R. Berghammer and T. Hoffmann, Deriving Relational Programs for Computing Kernels by Reconstructing a Proof of Richardson's Theorem, Sci. of Comput. Progr. 38 (2000), 1-25.
5. R. Berghammer and T. Hoffmann, Modeling Sequences Within the RELVIEW System, J. Universal Comput. Sci. 7 (2001), 107-123.
6. R. Berghammer and T. Hoffmann, Relational Depth-first-search with Applications, Information Sci. 139 (2001), 167-186.
7. R. Berghammer and T. Hoffmann, Calculating a Relational Program for Transitive Reductions of Strongly Connected Graphs, in: H. de Swart, ed., Proc. 5th Int. Seminar *Relational Methods in Computer Science*, LNCS 2561 (Springer, Berlin, 2002), 258-275.
8. R. Berghammer, T. Hoffmann, B. Leoniuk and U. Milanese, Prototyping and Programming with Relations, Electronic Notes in Theoret. Comput. Sci. 44 (2003), 1-24.

9. R. Berghammer, B. von Karger and C. Ulke, Relation-algebraic Analysis of Petri Nets with RelView, in: T. Margaria and B. Steffen, eds., Proc. 2nd Workshop *Tools and Applications for the Construction and Analysis of Systems*, LNCS 1055 (Springer, Berlin, 1996), 49-69.
10. R. Berghammer, B. von Karger and A. Wolf, Relation-algebraic Derivation of Spanning Tree Algorithms, in: J. Jeuring, ed., Proc. 4th Conf. *Mathematics of Program Construction*, LNCS 1422 (Springer, Berlin, 1998), 23-43.
11. R. Berghammer, B. Leoniuk and U. Milanese, Implementation of Relational Algebra Using Binary Decision Diagrams, in: H. de Swart, ed., Proc. 6th Int. Workshop *Relational Methods in Computer Science*, LNCS 2561 (Springer, Berlin, 2002), 241-257.
12. G. Birkhoff, Lattice Theory, 3rd edition, AMS Coll. Publ. 25 (AMS, New York, 1973).
13. G. Fay, An Algorithm for Finite Galois Connections, Techn. Report, Institute for Industrial Economy, Budapest, 1973.
14. E.R. Gansner, E. Koutsofios, S.C. North and K.P. Vo, A Technique for Drawing Directed Graphs, IEEE Trans. Software Eng. 19 (1993), 214-230.
15. B. Ganter, Two Basic Algorithms in Concept Analysis, FB 4 Preprint No. 831, TH Darmstadt, 1984.
16. B. Ganter, Formal Concept Analysis: Algorithmic Aspects, Lecture Notes, Summer Semester 2002, Institut für Algebra, TU Dresden, 2002.
17. B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations (Springer, Berlin, 1999).
18. B. Leoniuk, ROBDD-based Implementation of Relational Algebra with Applications (in German), Ph.D. thesis, Inst. für Inf. und Prak. Math., Univ. Kiel, 2001.
19. J. Ravelo, Two Graph Algorithms Derived, Acta Inform. 36 (1999), 489-510.
20. J. Ravelo, Relations, Graphs, and Programs, Ph.D. thesis, Techn. Monograph PRG-125, Programming Research Group, Oxford University, 1999.
21. G. Schmidt and T. Ströhlein, Relations and Graphs, Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., (Springer, Berlin, 1993).
22. A. Tarski, A Lattice-theoretical Fixed Point Theorem and its Applications, Pacific J. Math. 5 (1955), 285-309.

Journal on Relational Methods in Computer Science, Vol. 1, 2004, pp. 50 - 72
Received by the editors March 23, 2004, and, in revised form, September 09, 2004.
Published on December 10, 2004.

This article may be accessed via WWW at `http://www.jormics.org`.