Modeling Sequences within the RelView System

Rudolf Berghammer and Thorsten Hoffmann

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel, Olshausenstraße 40, D-24098 Kiel

1 Introduction

Since many years, axiomatic relational algebra (cf. [8]) has been used very successfully for formal problem specification and program derivation. Relations are well suited for modeling and reasoning about many discrete structures and computations on them. This holds in particular for those graph algorithms which manipulate sets of arcs resp. vertices since sets of arcs and relations are essentially the same and there are many simple and elegant ways to model sets of vertices by specific relations like vectors, partial identities, and injective embedding mappings. For details see [8]. Here one finds also examples for relation-algebraic derivations of graph algorithms; further derivations can be found, e.g., in [4, 7, 5].

Sets are not the only datatype used by graph algorithms. Also sequences are very important since many descriptions and algorithmic solutions of fundamental graph problems require different types of them. E.g., to calculate a path means to compute a sequence of vertices and to enumerate the strongly connected components means to compute a sequence of sets of vertices. We claim that in many cases relations can also be used for modeling sequences in a way which is well suited for calculations and formal program derivations. To prove our point we present in this paper a simple relation-algebraic model for sequences via binary direct sums which especially works for the relational manipulation and prototyping system RelView (cf. [3, 1]) and show a typical application.

2 Relation-algebraic Preliminaries

In this section we collect some basic concepts of relational algebra; for more details see [8] for example. We denote the set (or type) of all (binary) relations with domain X and range Y by $[X \leftrightarrow Y]$ and write $R : X \leftrightarrow Y$ instead of $R \in [X \leftrightarrow Y]$. If the sets X and Y are finite and of cardinality m resp. n, then 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 and also used within the RelView system, in the following we often use matrix terminology and matrix notation. The latter means that we write 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), $R \cup S$ (join), $R \cap S$ (meet), RS (composition), $R \subseteq S$ (inclusion), and the special relations O (empty relation), L (universal relation), and I (identity relation). The theoretical framework for all the well-known algebraic properties of relations is that of a relational algebra. As constants and operations of this abstract algebraic structure we have those of the set-theoretic relations; its axioms are those of a complete Boolean lattice for negation, join, meet, ordering, empty and universal relation, the axioms of a monoid for composition and identity relation, the so-called *Schröder equivalences*

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

and the so-called *Tarski rule*

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

An immediate consequence of (2) is $\mathsf{O} \neq \mathsf{L}$ which in turn implies that domain and range of each relation are non-empty. This agrees exactly with the use of relations in practice.

A relation R is said to be *reflexive* if $\mathsf{I} \subseteq R$, *transitive* if $RR \subseteq R$, and *symmetric* if $R \subseteq R^\mathsf{T}$. By an *equivalence relation* we mean a reflexive, transitive, and symmetric relation. For such a relation domain and range coincide, i.e., it is *homogeneous*, and thus the product RR is defined. In the Boolean matrix model of relations a homogeneous relation is quadratic. An arbitrary (also called *heterogeneous*) relation R is said to be *univalent* if $R^\mathsf{T}R \subseteq \mathsf{I}$ and *total* if $R\mathsf{L} = \mathsf{L}$. As usual, an univalent and total relation is said to be a (total) *mapping*. A relation R is called *injective* if R^T is univalent and *surjective* if R^T is total. An injective and surjective relation is said to be *bijective*. Another class of heterogeneous relations are *vectors*, which are defined by $v = v\mathsf{L}$ and can be used to describe subsets of a given set. If $[X \leftrightarrow Y]$ is the type of v, then $v = v\mathsf{L}$ means that whatever set Z, universal relation $\mathsf{L} : Y \leftrightarrow Z$, and element x from X we choose either $(v\mathsf{L})_{xz}$ holds for all elements z of Z or for none element z of Z. Consequently, for a vector the range is irrelevant. In the following we only consider vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\bot\}$ as range and omit the second subscript, i.e., write v_x instead of $v_{x\bot}$. Then v can be considered as a Boolean column vector and describes the subset $\{x \in X : v_x\}$ of X. A vector is said to be a *point* if it is injective and surjective. For $v : X \leftrightarrow \mathbf{1}$ these properties mean that it describes a singleton set, i.e., an element of X if we identify a singleton set with its only element. In the Boolean matrix model a point is a Boolean column vector in which exactly one component is true.

The operators on relations we will present now, are introduced in terms of the basic operations and in most cases only partially defined. Let R be a homogeneous relation. The least reflexive and transitive relation containing R is its *reflexive-transitive closure* $R^* = \bigcup_{i \geq 0} R^i$. Since we only deal with set-theoretic relations, the so-called point axiom of [8] holds. It says that for $R \neq \mathsf{O}$ there exist points p, q with $pq^\mathsf{T} \subseteq R$. As a consequence, for each vector $v \neq \mathsf{O}$ there exists a point p fulfilling $p \subseteq v$. The choice of such a point is fundamental for relational programming since it corresponds to the choice of an element from a non-empty set. For a non-empty vector v an axiomatization of the *choice* $\mathsf{point}(v)$ is given by

$$\mathsf{point}(v) \subseteq v \qquad\qquad \mathsf{point}(v) \text{ is a point}. \tag{3}$$

The symmetric quotient $\mathsf{syq}(R, S)$ of relations R and S is defined by

$$\mathsf{syq}(R, S) = \overline{R^\mathsf{T}\, \overline{S}} \cap \overline{\overline{R}^\mathsf{T} S}\ . \tag{4}$$

The right-hand side of (4) shows that $\mathsf{syq}(R, S)$ is only defined if R and S have the same domain. Then the domain of $\mathsf{syq}(R, S)$ is the range of R and the range of $\mathsf{syq}(R, S)$ is the range of S. Many properties of the symmetric quotient can be found in [8]. In the following lemma we collect some further properties.

Lemma 2.1. Let relations $R, S, p,$ and v be given. Then we have:

(i) If p is a point and $p^\mathsf{T} p = \mathsf{I}$, then $\mathsf{syq}(Rp, Rp) = \mathsf{I}$.
(ii) If p is a point, then $\mathsf{syq}(R, S)p = \mathsf{syq}(R, Sp)$.
(iii) If v is a vector, then $\mathsf{syq}(R, v)$ is a vector.
(iv) If R is an equivalence relation, then $\mathsf{syq}(S, R)R = \mathsf{syq}(S, R)$.
(v) If v is a non-empty vector and $R\mathsf{L} \subseteq \overline{v}$, then $\mathsf{syq}(R, v) = \mathsf{O}$.

Symmetric quotients are closely related to powersets. If we translate (4) in predicate logic notation using the set-theoretic definitions of the basic operations, then the result is

$$\mathsf{syq}(R, S)_{xy} \iff \forall\, z : R_{zx} \leftrightarrow S_{zy}\, . \tag{5}$$

Now we consider this equivalence for the special case of R being a membership relation $\in : X \leftrightarrow 2^X$ and S being a vector $v : X \leftrightarrow \mathbf{1}$. Then $\mathsf{syq}(\in, v)$ is of type $[2^X \leftrightarrow \mathbf{1}]$ and for each set Y from 2^X we have $\mathsf{syq}(\in, v)_Y$ if and only if $\forall\, z : z \in Y \leftrightarrow v_z$ holds. The latter shows that $\mathsf{syq}(\in, v)$ describes exactly the same set than v but as an element of 2^X instead of a subset of X as v does.

3 A Simple Relation-algebraic Model for Sequences

Our modeling of sequences within RelView is based upon a relation-algebraic characterization of binary direct sums. Within the relational framework it is natural to do the latter by means of the natural injective embedding mappings. This leads to the following relation-algebraic specification: A pair $\imath_1$ and $\imath_2$ of relations is called a *binary direct sum* if

$$\imath_1 {\imath_1}^\mathsf{T} = \mathsf{I} \qquad \imath_2 {\imath_2}^\mathsf{T} = \mathsf{I} \qquad {\imath_1}^\mathsf{T} \imath_1 \cup {\imath_2}^\mathsf{T} \imath_2 = \mathsf{I} \qquad \imath_1 {\imath_2}^\mathsf{T} = \mathsf{O}\, . \tag{6}$$

Given sets X_1 and X_2 it is easy to verify that the injective embedding mappings from these sets to the set-theoretic direct sum $X_1{+}X_2$ are a model of (6). Furthermore, by purely relation-algebraic reasoning it can be shown that the binary direct sum is uniquely characterized up to isomorphism by (6).

Based upon a binary direct sum $X_1 + X_2$ with the injective embedding mappings $\imath_1 : X_1 \leftrightarrow X_1{+}X_2$ and $\imath_2 : X_2 \leftrightarrow X_1{+}X_2$, now we define for relations $R : X_1 \leftrightarrow Y$ and $S : X_2 \leftrightarrow Y$ their *relational sum* $R + S : X_1{+}X_2 \leftrightarrow Y$ by

$$R + S = \imath_1{}^{\mathsf{T}} R \cup \imath_2{}^{\mathsf{T}} S\,. \tag{7}$$

This construction behaves like the relation R for all elements from $X_1{+}X_2$ which come from X_1 and like the relation S for all elements from $X_1{+}X_2$ which come from X_2.

Having defined the relational sum, now we are in a position to model non-empty (and finite) sequences of sets and elements in a very simple way. In Section 2 we have already shown how a subset (resp. an element) of a set X can be modeled by a vector (resp. a point) $v : X \leftrightarrow \mathbf{1}$. Therefore, it suffices to model sequences $\mathbf{v} = (v_i)_{1 \le i \le n}$ of vectors of type $[X \leftrightarrow \mathbf{1}]$ with relation-algebraic means. As we are interested in algorithms, especially executable with the RelView system, in the following we restrict us to relations $R : X \leftrightarrow Y$ with finite and enumerated carrier sets X and Y. Hence, we are allowed to consider R as a Boolean $m \times n$ matrix with m being the cardinality of X and n those of Y. In this Boolean matrix interpretation, which is also the standard way of RelView to depict a relation on its screen, R models the sequence $\mathbf{v} = (v_i)_{1 \le i \le n}$ of vectors, where $v_i : X \leftrightarrow \mathbf{1}$ corresponds to its i^{th} column ($1 \le i \le n$). Guided by this model, we define

$$R \,@\, S = (R^{\mathsf{T}} + S^{\mathsf{T}})^{\mathsf{T}}\,. \tag{8}$$

However, it must be pointed out that this concatenation of relations models the usual concatenation of sequences only within RelView because of the specific definition of the relational sum operation of the system which forms the Boolean matrix of $R + S$ by putting the Boolean matrices of R and S one upon another. Independent of the system, for a correct modeling of sequences via relation-algebraic binary direct sums we have to demand that the direct sum $X{+}Y$ of two disjoint and finite sets X and Y the elements of which are enumerated as $x_1, \ldots, x_m$ resp. $y_1, \ldots, y_n$ is enumerated as $x_1, \ldots, x_m, y_1, \ldots, y_n$.

The following lemma states some facts of the concatenation operation of (8) which are used in the next section.

Lemma 3.1. Let relations R, S, and T be given. Then we have:

(i) If $\mathsf{syq}(R,R) = \mathsf{I}$, $\mathsf{syq}(S,S) = \mathsf{I}$, and $\mathsf{syq}(R,S) = \mathsf{O}$, then $\mathsf{syq}(R \,@\, S, R \,@\, S) = \mathsf{I}$.

(ii) $\mathsf{syq}(R, S \,@\, T) = \mathsf{syq}(R,S) \,@\, \mathsf{syq}(R,T)$.

(iii) $(R \,@\, S)\mathsf{L} = R\mathsf{L} \cup S\mathsf{L}$.

This lemma does not fall out of the blue but relation-algebraically formalizes rather clear properties of sequences. We demonstrate this by means of (i): Using matrix terminology, from (5) one immediately obtains that $\mathsf{syq}(R,R) = \mathsf{I}$ holds if and only if the columns of R are pair-wise distinct and $\mathsf{syq}(R,S) = \mathsf{O}$ holds if and only if R and S have no column in common. Changing to sequence terminology, hence Lemma 3.1.i says: If the elements of a sequence $\mathbf{v}$ as well as of a sequence $\mathbf{w}$ are pair-wise distinct and $\mathbf{v}$ and $\mathbf{w}$ have no element in common, then also the elements of their concatenation are pair-wise distinct.

4 An Application: Computing Equivalence Classes

We assume $R : X \leftrightarrow X$ to be an equivalence relation on a finite set X. Our goal is to combine relational algebra and the Dijkstra-Gries program development method (see [6]) to derive formally a RelView program which enumerates the set $\mathcal{C}$ of all equivalence classes of R column-wise as a relation $C : X \leftrightarrow \mathcal{C}$.

First we have to specify the problem by relation-algebraic pre- and postconditions. As we assume the relation R to be the input of the program we want to derive, the precondition $\mathit{pre}(R)$ is obvious:

$$\mathit{pre}(R) \;\triangleq\; \mathsf{I} \subseteq R \,\wedge\, RR \subseteq R \,\wedge\, R = R^{\mathsf{T}}\,.$$

The relation C is the program's output. Using matrix terminology, therefore we have to specify that its columns are pair-wise distinct and describe all equivalence classes of R. In Section 3 we have already shown how to formalize the first of these properties with relation-algebraic means. This leads to $\mathsf{syq}(C,C) = \mathsf{I}$ as first part of the postcondition $\mathit{post}(R,C)$. To formalize also the second of the above properties within relational algebra, we modify the relation-algebraic specification of the strongly connected components given in [2] and get, with $\in : X \leftrightarrow 2^X$ as membership relation, $\mathsf{syq}(\in, R)\mathsf{L} : 2^X \leftrightarrow \mathbf{1}$ as vector for describing the equivalence classes of R as elements of 2^X. This leads to $\mathsf{syq}(\in, R)\mathsf{L} = \mathsf{syq}(\in, C)\mathsf{L}$ as second part of $post(R, C)$. In words it says that each column of C describes an equivalence class of R and, conversely, each equivalence class of R is described by a column of C. Altogether, we have the postcondition

$$post(R,C) \;\stackrel{\triangle}{=}\; \mathsf{syq}(C,C) = \mathsf{I} \wedge \mathsf{syq}(\in,R)\mathsf{L} = \mathsf{syq}(\in,C)\mathsf{L}\,.$$

Our next goal is to calculate a loop invariant and a guard from $post(R, C)$. Here we follow the most common approach of generalizing a postcondition by introducing new variables. In the present case it seems to be a good idea to compute the equivalence classes of R, i.e., the sequence of vectors modeled by C, one after the other and to use a vector for describing the elements yet to be checked. If we use $v : X \leftrightarrow \mathbf{1}$ for the latter purpose, then we arrive at

$$inv(R,C,v) \;\stackrel{\triangle}{=}\; \mathsf{syq}(C,C) = \mathsf{I} \wedge \mathsf{syq}(\in,R)\,\overline{v}\, = \mathsf{syq}(\in,C)\mathsf{L} \wedge Rv \cap C\mathsf{L} = \mathsf{O}$$

as loop invariant $inv(R, C, v)$. In words its second equation says that the columns of C describe the equivalence classes of the already checked elements (which again are described by $\overline{v}$) and its third equation says that no element of an equivalence class which is still to be computed occurs in an equivalence class which has already been computed.

It is obvious that $inv(R, C, v)$ implies $post(R, C)$ when v is empty. This leads to $v \neq \mathsf{O}$ as guard of the loop. Hence, it remains to develop an initialization of C and v which establishes $inv(R, C, v)$ and a body of the loop which maintains $inv(R, C, v)$ and ensures termination.

Let's start with the initialization. Here it seems to be a good idea to choose arbitrarily an equivalence class, i.e., to initialize C with the vector $R\mathsf{point}(\mathsf{L})$, where $\mathsf{L} : X \leftrightarrow \mathbf{1}$. Consequently, we have to initialize v with $\overline{C}$, i.e., with $\overline{R\mathsf{point}(\mathsf{L})}$. This initialization establishes the loop invariant. In the subsequent proof of this fact we abbreviate the choice $\mathsf{point}(\mathsf{L})$ as p. The first equation $\mathsf{syq}(Rp, Rp) = \mathsf{I}$ of $inv(R, Rp, \overline{Rp})$ is an immediate consequence of Lemma 2.1.i since p is a point due to (3) and from its type $[X \leftrightarrow \mathbf{1}]$ we get $p^{\mathsf{T}}p = \mathsf{I}$. A proof of the second equation of $inv(R, Rp, \overline{Rp})$ is

$$\mathsf{syq}(\in,R)\,\overline{\overline{Rp}}\, = \mathsf{syq}(\in,R)Rp = \mathsf{syq}(\in,R)p = \mathsf{syq}(\in,Rp) = \mathsf{syq}(\in,Rp)\mathsf{L}\,,$$

where we successively have applied lattice theory, Lemma 2.1.iv in combination with the precondition, Lemma 2.1.ii, and Lemma 2.1.iii. For a proof of its third equation we use that it is equivalent to $R\,\overline{Rp} \subseteq \overline{Rp}$ and prove this inclusion. Symmetry and transitivity of R show $R^{\mathsf{T}}R \subseteq R$ which in turn implies $R^{\mathsf{T}}Rp \subseteq Rp$. Now, we apply (1) and are done.

To complete the derivation, we have to work out a loop body and to verify that it maintains $inv(R, C, v)$ and ensures termination. Since C finally shall enumerate the equivalence classes of R and v describes those elements of X during the computation of C the classes of which still have to be added to C, it seems to be promising to explore the effect of the assignments $C := C \,@\, R\mathsf{point}(v)$ and $v := v \cap \overline{R\mathsf{point}(v)}$ which add a new class to C and change v accordingly. In RelView syntax, hence the final program is

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

To prove that its loop body maintains the loop invariant, we assume $v \neq \mathsf{O}$ and $inv(R, C, v)$. Furthermore, we abbreviate $\mathsf{point}(v)$ as p. To prove the first equation of $inv(R, C \,@\, Rp, v \cap \overline{Rp})$ we apply Lemma 3.1.i and must only verify its three premises. Equation $\mathsf{syq}(C, C) = \mathsf{I}$ is part of the supposed invariant $inv(R, C, v)$. A proof of $\mathsf{syq}(Rp, Rp) = \mathsf{I}$ follows from Lemma 2.1.i since p is again a point with $p^{\mathsf{T}}p = \mathsf{I}$. Finally, to prove $\mathsf{syq}(C, Rp) = \mathsf{O}$ we use $Rv \cap C\mathsf{L} = \mathsf{O}$ since this part of $inv(R, C, v)$ in combination with $p \subseteq v$ shows $C\mathsf{L} \subseteq \overline{Rp}$ which in turn implies the desired result due to the vector property of Rp, the inequation $\mathsf{O} \neq Rp$ (which follows from surjectivity of p and reflexivity of R), and Lemma 2.1.v. Here is the proof of the second equation of $inv(R, C \mathbin{@} Rp, v \cap \overline{Rp})$:

$$\begin{array}{rll}
\mathsf{syq}(\in, R)\,\overline{v \cap \overline{Rp}} & = \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, R)Rp & \\
& = \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, R)p\mathsf{L} & pre(R),\ \text{Lemma 2.1.iv},\ p = p\mathsf{L} \\
& = \mathsf{syq}(\in, R)\,\overline{v} \cup \mathsf{syq}(\in, Rp)\mathsf{L} & p \text{ point, Lemma 2.1.ii} \\
& = \mathsf{syq}(\in, C)\mathsf{L} \cup \mathsf{syq}(\in, Rp)\mathsf{L} & inv(R, C, v) \\
& = \mathsf{syq}(\in, C \mathbin{@} Rp)\mathsf{L} & \text{Lemma 3.1.iii, Lemma 3.1.ii}\,.
\end{array}$$

Finally, a proof of the third equation of $inv(R, C \mathbin{@} Rp, v \cap \overline{Rp})$ is given by

$$\begin{array}{rll}
R(v \cap \overline{Rp}) \cap (C \mathbin{@} Rp)\mathsf{L} & = R(v \cap \overline{Rp}) \cap (C\mathsf{L} \cup Rp\mathsf{L}) & \text{Lemma 3.1.iii} \\
& \subseteq Rv \cap R\,\overline{Rp} \cap (C\mathsf{L} \cup Rp) & p = p\mathsf{L} \\
& = R\,\overline{Rp} \cap ((Rv \cap C\mathsf{L}) \cup (Rv \cap Rp)) & \\
& = R\,\overline{Rp} \cap Rp & inv(R, C, v),\ p \subseteq v \\
& = \mathsf{O} & R^{\mathsf{T}}Rp \subseteq Rp,\ (1)\,.
\end{array}$$

Since X is finite and $Rp \neq \mathsf{O}$, the above program obviously terminates if $Rp \cap v \neq \mathsf{O}$. We prove this inequation by contradiction: Assume $Rp \subseteq \overline{v}$. Then we get $R^{\mathsf{T}}v \subseteq \overline{p}$ due to (1). Now $pre(R)$ implies $v \subseteq Rv = R^{\mathsf{T}}v \subseteq \overline{p}$, i.e., $p \subseteq \overline{v}$, and combining this with $p \subseteq v$ yields the contradiction $p = \mathsf{O}$.

5 Conclusion

We have presented a relation-algebraic model of sequences. It contrasts with all existing work in this domain since it is specifically tailored to the relational manipulation and prototyping system RelView without touching the system's present state. Afterwards, we have combined relational algebra with the Dijkstra-Gries program development method to derive formally a RelView program for the column-wise enumeration of equivalence classes.

Experiments have shown that, using higher-order constructs like the membership relation $\in : X \leftrightarrow 2^X$, our approach often allows to specify a problem involving sequences as an executable RelView-function which then may be used for protyping. See [2, 4, 3] for many examples. We have found it also very attractive to use such functions for producing good examples in teaching.

Strictly speaking, we only have presented a RelView-model of non-empty sequences with a concatenation operation. But it is easy to refine it to a model of stacks with the additional well-known operations first and rest using the relation-algebraic operators inj for the injective mapping generated by a vector and init for the initial element of a carrier set. Due to lack of space we have to renounce details. We have applied this refinement to solve many further problems involving sequences with RelView, e.g., graph-theoretic ones (strongly connected resp. biconnected components, vertex bases, Eulerian cycles), Petri-net problems (reachable resp. live markings of CE-nets), and problems on ordered sets (cut resp. ideal completion). To give an impression for running times, for the well-known five dining philosophers CE-net computing all reachable markings takes one second on a Sun Ultra V workstation.

References

1. Behnke R., Berghammer R., Meyer E., Schneider P.: RelView – A system for calculating with relations and relational programming. In: Astesiano E. (ed.): Proc. FASE '98, LNCS 1382, Springer, 318-321 (1998)
2. Berghammer R., Gritzner T., Schmidt G.: Prototyping relational specifications using higher-order objects. In: Heering, J. et al. (eds.): Proc. HOA '93, LNCS 816, Springer, 56-75 (1994)
3. Berghammer R., Karger B. von, Ulke C.: Relation-algebraic analysis of Petri nets with RelView. In: Margaria T., Steffen B. (eds.): Proc. TACAS '96, LNCS 1055, Springer, 49-69 (1996)
4. Berghammer R., Karger B. von: Algorithms from relational specifications. In: Brink C. et al. (eds.): Relational methods in Computer Science. Advances in Computing Sci., Springer, 131-149 (1997)
5. Berghammer R., Karger B. von, Wolf A.: Relation-algebraic derivation of spanning tree algorithms. In: Jeuring J. (ed.): Proc. MPC '98, LNCS 1422, Springer, 23-43 (1998)
6. Gries D.: The science of computer programming. Springer (1981)
7. Karger B. von, Berghammer R.: Computing kernels in directed bichromatic graphs. Inform. Proc. Let. 62, 5-11 (1997)
8. Schmidt G., Ströhlein T.: Relations and graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993)