Prototyping Relational Specifications Using Higher-Order Objects

Rudolf Berghammer* Thomas F. Gritzner** Gunther Schmidt*

* Fakultät für Informatik
Universität der Bundeswehr München
Werner–Heisenberg–Weg 39
D–85577 Neubiberg
GERMANY

** Fakultät für Informatik
Technische Universität München
Arcisstraße 21
D-80290 München
GERMANY

Abstract: An approach is described for the generation of certain mathematical objects (like sets, correspondences, mappings) in terms of relations using higher-order objects. It shows also, how to perform the resulting executable relational specifications using the computer system RELVIEW and how to develop more efficient algorithms from the (almost always) inefficient specifications within the calculus of binary relations.

Contents

1 Introduction **3**

2 Relation Algebraic Preliminaries **4**
2.1 Basic Operations on Relations and Basic Relation Algebraic Laws 4
2.2 Special Relations 5
2.3 Quotients, Bounds, and Closures 5
2.4 Relational Domain Construction 6

3 Relational Problem Specification and Prototyping **9**
3.1 Point Sets of a Directed Graph 10
3.2 Point Sets of a Partially Ordered Set 12
3.3 Point Sets of a Simple Graph 14
3.4 Point and Edge Sets of a Hypergraph 15
3.5 Sets of Places of a Petri Net 15
3.6 Correspondences on a Directed Graph 16
3.7 Sets of Mappings 17

4 Using the RELVIEW System For Rapid Prototyping **19**
4.1 A Short Look at RELVIEW 19
4.2 Examples 20
4.3 Development of Efficient Algorithms Using Abstract Relation Algebra 24

5 Concluding Remarks **29**

References **31**

1 Introduction

During the last two decades, axiomatic relational calculus of Tarski ([Tarski 41]) has widely been used by computer scientists who view it as a convenient formalism for describing fundamental concepts of programming. The development starts with the work of de Bakker and de Roever in the early 70's; see [de Bakker de Roever 73, de Roever 74] for example. In the following decade, e.g., Hoare and He ([Hoare He 86]) related the work of Birkhoff on residuals with Dijkstra's weakest precondition approach to programming, a group in Munich (see [Schmidt 81, Berghammer Zierer 86, Zierer 91]) constructed semantic domains by relation-algebraic means and, thus, was able to treat also languages with higher-order functions, and a group in Eindhoven (see [Backhouse et al. 92]) developed a theory of data types based on the calculus of relations. At this point also the approach of a group in Rio should be mentioned which was motivated mainly by the development of a relational programming calculus not bounded by lack of expressiveness; compare [Veloso Haeberer 91].

In order that such investigations do not stay completely on the theoretical side, several aspects need special consideration. Firstly, relational methods are not so commonly known that competent discussion with researchers from other approaches is easy. So, there should be a tool at hand to facilitate and visualize work with relations. Secondly, some tool for rapid prototyping of program specifications expressed in a relational style should be developed. And, finally, program development methods starting from such relational specifications have to be considered. Since relational specifications have a high degree of precision and formal structure it seems promising to apply transformational techniques.

In program development, the relational framework has already been used. E.g., in a series of articles a group around Desharnais and Mili proposed a relational approach to the formal derivation of imperative programs from its specifications. See [Mili 83, Mili et al. 87, Desharnais et al. 93] for example. Recently, Möller ([Möller 91]) used n-ary relations between nested tuples as elements of an applicative program development language, and also the Rio group developed various case studies on formal program construction using relations (cf. [Veloso et al. 93]).

Also this paper deals with relation algebra as a formal programming tool. However, our main objective is not program transformation. Instead, we concentrate on the development of executable relational specifications out of non-constructive first-order problem descriptions. Here some special functionals on relations in conjunction with a relational description of higher-order objects play an important rôle. Furthermore, we describe a computer system which facilitates and visualizes work with relations and demonstrate how it can be used to execute specifications.

The rest of the paper is organized as follows. In Section 2, we briefly recall some basic properties from the algebra of relations and introduce some special relations. Furthermore, we consider some special functionals on relations like residuals and symmetric quotients and give monomorphic characterizations of some domain constructions with relation-algebraic means. Section 3 is devoted to relational problem specification and prototyping. Using the correspondence between certain kinds of first-order formulae and certain relation-algebraic constructions (resp. operations), we show how the computation of objects like sets, correspondences, and mappings can be described in terms of relations. Most of the presented examples are borrowed from graph theory. The resulting algorithms can be seen as executable relational specifications. They are built up very quickly and their proofs of correctness are very simple. However, they may be fairly inefficient compared to hand-made ones. Hence, we have the typical situation of

rapid prototyping. In Section 4, first, a short description of a totally interactive and completely video-oriented computer system for the manipulation of relations – called RELVIEW – is given. Using the system, we then show how to perform executable relational specifications as those developed in Section 3. In Section 4 we demonstrate also the development of more efficient algorithms from the original inefficient specifications within abstract relation algebra and, finally, sketch some further experiences with the RELVIEW system.

2 Relation Algebraic Preliminaries

In this section, we briefly introduce the basic concepts of the algebra of relations, some special relations, and relation-algebraic constructions. For more details concerning the algebraic theory of relations, see e.g., [Chin Tarski 51, Jónsson Tarski 52, Schmidt Ströhlein 89].

2.1 Basic Operations on Relations and Basic Relation Algebraic Laws

For two sets X and Y, a subset R of the Cartesian product $X \times Y$ is a relation between the *domain* X and the *range* Y. We call it *homogeneous* if $X = Y$; otherwise we call it *heterogeneous*. Considering the corresponding characteristic predicates instead of the set representations, a relation R between X and Y becomes a function $R : X \times Y \to \mathbb{B}$, where $\mathbb{B}$ denotes the set $\{0, 1\}$ of truth-values. Therefore, 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 of relations is well-suited for a graphical representation and also used within the RELVIEW system described below. Following the notation of the specification language Z (see [Spivey 89]), we write $R : X \leftrightarrow Y$ if R is a relation between X and Y. Furthermore, we use matrix notation and 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), $R \subset S$ (inclusion), and the special relations O (empty relation), L (universal relation), and I (identity relation). The set-theoretic operations $\overline{\ }$, $\cup$, $\cap$, the ordering $\subset$, and the constants O and L are related as usual. Some further well-known rules concerning relations are

$$
\begin{array}{ll}
(R^{\mathrm{T}})^{\mathrm{T}} = R & R \subset S \implies R^{\mathrm{T}} \subset S^{\mathrm{T}} \\
R^{\mathrm{T}} S^{\mathrm{T}} = (SR)^{\mathrm{T}} & \overline{R}^{\mathrm{T}} = \overline{R^{\mathrm{T}}} \\
R \subset S \implies QR \subset QS & R \subset S \implies RQ \subset SQ \\
Q(R \cap S) \subset QR \cap QS & Q(R \cup S) = QR \cup QS \\
(R \cap S)^{\mathrm{T}} = R^{\mathrm{T}} \cap S^{\mathrm{T}} & (R \cup S)^{\mathrm{T}} = R^{\mathrm{T}} \cup S^{\mathrm{T}},
\end{array}
$$

where the last two lines also hold if binary meet and join are replaced by arbitrary join or meet, respectively. The theoretical framework for all these rules to hold is that of an abstract relation algebra. The axioms of this algebraic structure are the axioms of a complete atomic Boolean algebra for $\overline{\ }$, $\cup$, $\cap$, $\subset$, O, L, the axioms of a monoid for composition and I, the *Dedekind rule*

$$(QR \cap S) \subset (Q \cap SR^{\mathrm{T}})(R \cap Q^{\mathrm{T}} S),$$

and the *Tarski rule*

$$R \neq \mathsf{O} \implies \mathsf{L}R\mathsf{L} = \mathsf{L}.$$

An alternative form of the Dedekind rule are the *Schröder equivalences*, viz.

$$QR \subset S \iff Q^{\mathrm{T}}\overline{S} \subset \overline{R} \iff \overline{S}R^{\mathrm{T}} \subset \overline{Q}.$$

2.2 Special Relations

The basic operations and constants of the last subsection are very helpful for defining simple properties on relations in a component-free manner. In the remainder of the paper we need the following special relations:

Functions: Let $R : X \leftrightarrow Y$ be a relation. R is said to be a *partial function* or, briefly, to be *functional* if $R^{\mathrm{T}}R \subset \mathrm{I}$, and R is said to be *total* if $R\mathsf{L} = \mathsf{L}$, which is, in turn, equivalent to $\mathrm{I} \subset RR^{\mathrm{T}}$. As usual, a functional and total relation is said to be a *(total) function*. A relation R is called *injective* if R^{T} is functional, and R is called *surjective* if R^{T} is total. An injective and surjective relation is said to be *bijective*.

Partial orderings: Assume $E : X \leftrightarrow X$ to be a homogeneous relation. E is *reflexive* if $I \subset E$, *transitive* if $EE \subset E$, and *antisymmetric* if $E \cap E^{\mathrm{T}} \subset \mathrm{I}$. A *partial ordering* is a reflexive, antisymmetric, and transitive relation. If E is a partial ordering, then $E \cap \overline{\mathrm{I}}$ denotes its *irreflexive part*.

Vectors: A relation $v : X \leftrightarrow Y$ with $v = v\mathsf{L}$ is called a *(row-constant) vector* or a *predicate*. This condition means that an element x from X is either in relation v to none of the elements of Y or to all elements of Y. Relations of this kind may be considered as subsets of the set X. This becomes obvious if we use the common set-theoretic notation for relations, since then v equals a Cartesian product $X' \times Y$, where X' is a subset of X. As for a vector the range is without relevance, we consider in the following only vectors $v : X \leftrightarrow \mathsf{1}$ with a singleton set $\mathsf{1} = \{u\}$ as their range and write v_x instead of v_{xu}. Such a vector can be considered as a Boolean matrix with exactly one column, i.e., as a Boolean column vector, and describes the subset $\{x \in X : v_x\}$ of X.

Points: A vector $v : X \leftrightarrow \mathsf{1}$ is said to be a *point* if it is bijective. If $p : X \leftrightarrow \mathsf{1}$ is a point, then its transpose is a (constant) function $p^{\mathrm{T}} : \mathsf{1} \leftrightarrow X$. For points p and q we have that pq^{T} models the one-point partial function only mapping p to q if we interpret the subset $R^{\mathrm{T}}p$ as the result of the application of R to p, which is true for the matrix model.

2.3 Quotients, Bounds, and Closures

In this subsection, we consider some special mappings from relations to relations. To distinguish such "meta-level functions" from the relation-algebraic (or: object level) notion of functions, they are also called *functionals*. In most cases, they happen to be partial functionals, since the basic operations besides the unary ones are only partially defined.

Residuals: Residuals are the greatest solutions of certain inclusions. The *left residual* of S over R (in symbols $S \,/\, R$) is the greatest relation Q such that $QR \subset S$; the *right residual* of S over R (in symbols $R \setminus S$) is the greatest relation Q such that $RQ \subset S$. Both residuals may also be represented using the basic operations: Let $R : X \leftrightarrow Z$, $S : Y \leftrightarrow Z$, $R' : Z \leftrightarrow X$, and $S' : Z \leftrightarrow Y$ be given, then, from the Schröder equivalences we obtain

$$S \,/\, R = \overline{\overline{S}R^{\mathrm{T}}} \qquad\qquad R' \setminus S' = \overline{R'^{\mathrm{T}}\overline{S'}}.$$

Note also that the two residuals are linked together by the relationships $R \setminus S = (S^{\mathrm{T}} / R^{\mathrm{T}})^{\mathrm{T}}$ and $\overline{R} \setminus \overline{S} = R^{\mathrm{T}} / S^{\mathrm{T}}$. Translating the relation-algebraic expressions into component-wise formulations, for left residual $S / R : Y \leftrightarrow X$ and right residual $R' \setminus S' : X \leftrightarrow Y$ we have the equivalences

$$(S / R)_{yx} \iff \bigwedge_z R_{xz} \to S_{yz} \qquad (R' \setminus S')_{xy} \iff \bigwedge_z R'_{zx} \to S'_{zy}.$$

In particular, we have the two following correspondences for single first-order universal quantification:

$$(S / \mathsf{L})_y \iff \bigwedge_z S_{yz} \qquad (\overline{R'} \setminus \mathsf{O})_x \iff \bigwedge_z R'_{zx}.$$

Symmetric Quotients: In the following, we will frequently need relations which are left and right residuals simultaneously, viz. symmetric quotients. The *symmetric quotient* $\mathsf{syq}(R, S)$ of two relations R and S is defined as the greatest relation Q such that $RQ \subset S$ and $QS^{\mathrm{T}} \subset R^{\mathrm{T}}$. If $R : Z \leftrightarrow X$ and $S : Z \leftrightarrow Y$, then we have $\mathsf{syq}(R, S) : X \leftrightarrow Y$ as

$$\mathsf{syq}(R, S) = \overline{R^{\mathrm{T}}\,\overline{S}} \cap \overline{\overline{R^{\mathrm{T}}}\,S} = (R \setminus S) \cap (R^{\mathrm{T}} / S^{\mathrm{T}}).$$

In component-wise notation, the symmetric quotient $\mathsf{syq}(R, S)$ satisfies the equivalence

$$\mathsf{syq}(R, S)_{xy} \iff \bigwedge_z R_{zx} \leftrightarrow S_{zy}.$$

Bounds and extremal elements: Let $E : X \leftrightarrow X$ be a partial ordering. Due to later applications, we ask for some order-theoretic concepts such as the set of lower (resp. upper) bounds of a subset wrt. E or the set of minimal (resp. maximal) elements of a subset wrt. E. We define four functionals dependent on E and a further relation $R : X \leftrightarrow Y$ as follows:

Lower bounds:	$\mathsf{mi}(E, R) = \overline{\overline{E}R}$
Upper bounds:	$\mathsf{ma}(E, R) = \overline{\overline{E^{\mathrm{T}}}R}$
Minimal elements:	$\mathsf{min}(E, R) = R \cap \overline{(E \cap \overline{\mathrm{I}})^{\mathrm{T}}R}$
Maximal elements:	$\mathsf{max}(E, R) = R \cap \overline{(E \cap \overline{\mathrm{I}})R}$

Looking at the corresponding component-wise descriptions and assuming R to be a vector, it is easy to see that $\mathsf{mi}(E, R)$ (resp. $\mathsf{ma}(E, R)$) yields the subset of lower (resp. upper) bounds of R wrt. partial ordering E, while $\mathsf{min}(E, R)$ (resp. $\mathsf{max}(E, R)$) computes to the subset of minimal (resp. maximal) elements of R wrt. partial ordering E. If R is not a vector, then the functionals compute bounds and extremal elements column-wise.

Closures: Assume $R : X \leftrightarrow X$ to be a homogeneous relation. The *reflexive closure* of R, i.e., the least reflexive relation containing R, simply computes to $R \cup \mathrm{I}$. The least transitive relation containing R is called the *transitive closure* of R and denoted by R^+, while the least reflexive and transitive relation containing R is called the *reflexive-transitive closure* of R and denoted by R^*.

2.4 Relational Domain Construction

To deal with composed and higher-order objects like tuples, sets or functions, we have to explain how the corresponding domain constructions can be performed with relational algebra.

Note that the constructions described in the following, after the definitions of homomorphisms (resp. isomorphisms), may or may not exist in an arbitrary model of relational algebra; however, this problem does not occur at the concrete matrix model underlying this paper.

Homomorphisms: Let $R : X \leftrightarrow Y$ and $S : X' \leftrightarrow Y'$ be two relations and consider a pair $\mathfrak{H} = (\Phi, \Psi)$ of functions $\Phi : X \leftrightarrow X'$ and $\Psi : Y \leftrightarrow Y'$. $\mathfrak{H}$ is called a *homomorphism* from R to S if $R \subset \Phi S \Psi^{\mathrm{T}}$ holds. If, in addition, the pair $\mathfrak{H}^{\mathrm{T}} = (\Phi^{\mathrm{T}}, \Psi^{\mathrm{T}})$ is a homomorphism from S to R, then $\mathfrak{H}$ is said to be an *isomorphism* between R and S. Therefore, an isomorphism $\mathfrak{I} = (\Phi, \Psi)$ between R and S is a pair of bijective functions $\Phi : X \leftrightarrow X'$ and $\Psi : Y \leftrightarrow Y'$, which satisfies the condition $R\Psi = \Phi S$. If R and S are homogeneous, then Φ is briefly called a homomorphism (isomorphism) if the pair (Φ, Φ) is a homomorphism (isomorphism).

Direct products: Within the framework of abstract relation algebra it is natural to characterize direct products by means of the natural projections, see [Schmidt Ströhlein 89] or [Zierer 91]. Then one obtains the following specification: Let $\mathfrak{P} = (\pi_k : PX \leftrightarrow X_k)_{1 \leq k \leq n}$ be an n-tuple of $n > 0$ relations. We call $\mathfrak{P}$ an *(n-ary) direct product* if and only if

$$(P_1)\ \ \pi_k^{\mathrm{T}} \pi_k = \mathrm{I} \qquad (P_2)\ \ j \neq k \implies \pi_j^{\mathrm{T}} \pi_k = \mathsf{L} \qquad (P_3)\ \ \bigcap_{k=1}^{n} \pi_k \pi_k^{\mathrm{T}} = \mathrm{I}.$$

It is easy to verify that the natural projections from a Cartesian product $\Pi_{i=1}^{n} X_i$ to the components X_k are a model of (P_1) through (P_3) if the placeholder PX is replaced by $\Pi_{i=1}^{n} X_i$. By purely relation-algebraic reasoning, furthermore, it can be shown that the direct product is uniquely characterized up to isomorphism: If $\mathfrak{Q} = (\rho_k : PY \leftrightarrow Y_k)_{1 \leq k \leq n}$ is another model of (P_1) through (P_3) and $(\Psi_k : X_k \leftrightarrow Y_k)_{1 \leq k \leq n}$ is a family of bijective functions, then we can establish an isomorphism between π_k and ρ_k by the pair $\mathfrak{I}_k = (\Phi, \Psi_k)$, where the bijective function $\Phi : PX \leftrightarrow PY$ is defined as $\Phi = \bigcap_{i=1}^{n} \pi_i \Psi_i \rho_i^{\mathrm{T}}$.

Based on the binary direct products $\mathfrak{P} = (\pi, \rho)$ and $\mathfrak{Q} = (\sigma, \tau)$ we define the following two functionals, where the generalization to n-ary direct products ($n > 2$) is straightforward, but not needed in the remainder of the paper:

Tupling:	$[R, S]_{\mathfrak{P}} = R\pi^{\mathrm{T}} \cap S\rho^{\mathrm{T}}$
Parallel composition:	$R \mathbin{\mathfrak{P}\|\|_{\mathfrak{Q}}} S = \pi R \sigma^{\mathrm{T}} \cap \rho S \tau^{\mathrm{T}}.$

Direct sums: The direct sum can be defined in largely the same fashion as the direct product. Dually to the natural projections the natural injections are used, see [Zierer 91]. Then one obtains the following specification: Let $\mathcal{S} = (\imath_k : SX \leftrightarrow X_k)_{1 \leq k \leq n}$ be an n-tuple of $n > 0$ relations. We call $\mathcal{S}$ an *n-ary direct sum* if and only if

$$(S_1)\ \ \imath_k \imath_k^{\mathrm{T}} = \mathrm{I} \qquad (S_2)\ \ j \neq k \implies \imath_j \imath_k^{\mathrm{T}} = \mathsf{O} \qquad (S_3)\ \ \bigcup_{k=1}^{n} \imath_k^{\mathrm{T}} \imath_k = \mathrm{I}.$$

Given sets X_k, $1 \leq k \leq n$, it is easy to verify that the injections from these sets to the direct sum $\Sigma_{k=1}^{n} X_k$ are a model of (S_1) through (S_3). Again by purely relation algebraic reasoning it can be shown that by the laws the direct sum is uniquely characterized up to isomorphism and the relations $\imath_k$, $1 \leq k \leq n$, are injective functions.

Powersets: A relation-algebraic characterization of the powerset 2^X of a set X can conveniently be done using the "is-element-of" relation, see [Berghammer et al. 89]. Formally, we call $\varepsilon : X \leftrightarrow SX$ a *powerset relation* if and only if

$$(PS_1)\ \ \mathsf{syq}(\varepsilon, \varepsilon) \subset \mathrm{I} \qquad (PS_2)\ \ \bigwedge_{R} \mathsf{L}\, \mathsf{syq}(\varepsilon, R) = \mathsf{L}.$$

Since every relation-algebraic equation using ε is translated into a formula with higher-order quantification (over sets), in (PS_2) the higher-order quantification (over relations) does not surprise. Again it can be shown by purely relation-algebraic reasoning that the powerset relation is uniquely characterized up to isomorphism. Indeed, if $\varepsilon' : Y \leftrightarrow SY$ is another powerset relation, $\Phi : X \leftrightarrow Y$ is a bijective function, and one defines the bijective function $\Psi : SX \leftrightarrow SY$ by $\Psi = \mathsf{syq}(\varepsilon, \Phi\varepsilon')$, then $\mathfrak{I} = (\Phi, \Psi)$ is an isomorphism between ε and ε'.

Now, assume the concrete case of the "is-element-of" relation $\varepsilon : X \leftrightarrow 2^X$. Then (PS_1) corresponds to the extensionality axiom, whereas (PS_2) says that every vector (set) $v : X \leftrightarrow \mathsf{1}$ has a corresponding point $\mathsf{vp}(v) := \mathsf{syq}(\varepsilon, v) : 2^X \leftrightarrow \mathsf{1}$ in the powerset. This shows that the usual "is-element-of" relation is a powerset relation. The functional vp is injective and its left-inverse on points is $\mathsf{vp}^{-1}(p) = \varepsilon p : X \leftrightarrow \mathsf{1}$. Hence, vp establishes some kind of isomorphism between subsets of X and elements of 2^X. For details, compare [Berghammer et al. 89].

Based on the relation $\varepsilon : X \leftrightarrow 2^X$, the relational specification of sets merely by equations can be established. Namely, we have:

Empty set:	$\mathbb{E} : 2^X \leftrightarrow \mathsf{1}$	$\mathbb{E} = \mathsf{syq}(\varepsilon, \mathsf{O}) = \varepsilon \setminus \mathsf{O}$
Universal set:	$\mathbb{U} : 2^X \leftrightarrow \mathsf{1}$	$\mathbb{U} = \mathsf{syq}(\varepsilon, \mathsf{L}) = \overline{\varepsilon} \setminus \mathsf{O}$
Singleton embedding:	$\mathbb{S} : X \leftrightarrow 2^X$	$\mathbb{S} = \mathsf{syq}(\mathrm{I}, \varepsilon)$
Complementation:	$\mathbb{C} : 2^X \leftrightarrow 2^X$	$\mathbb{C} = \mathsf{syq}(\varepsilon, \overline{\varepsilon})$
Meet of sets:	$\mathbb{M} : 2^X \times 2^X \leftrightarrow 2^X$	$\mathbb{M} = \mathsf{syq}([\varepsilon, \varepsilon]\mathfrak{P}, \varepsilon)$
Join of sets:	$\mathbb{J} : 2^X \times 2^X \leftrightarrow 2^X$	$\mathbb{J} = (\mathbb{C}\,\mathfrak{P}\|\|\mathfrak{P}\,\mathbb{C})\mathbb{M}\mathbb{C}$
Inclusion of sets:	$\sqsubseteq : 2^X \leftrightarrow 2^X$	$\sqsubseteq = \varepsilon \setminus \varepsilon$

In the description of the binary operations meet and join we use a direct product $\mathfrak{P} = (\pi, \rho)$, consisting of the two projections from $2^X \times 2^X$ to the first and the second component, respectively.

Relation spaces and function spaces: We are also interested in describing the set of all relations between two sets and some certain subsets by relation-algebraic means. The set of all relations between X and Y is easy to handle, since it is a powerset relation $\varepsilon_R : X \times Y \leftrightarrow 2^{X \times Y}$. Hence, $\mathfrak{R} = (\pi, \rho, \varepsilon_R)$ is called a *relation space* if and only if

$$\begin{array}{ll} (R_1) & (\pi, \rho) \text{ is a direct product} \\ (R_2) & \pi^{\mathrm{T}}\varepsilon_R \text{ and } \rho^{\mathrm{T}}\varepsilon_R \text{ exist} \\ (R_3) & \varepsilon_R \text{ is a powerset relation.} \end{array}$$

The transformation of a relation $R : X \leftrightarrow Y$ into a point of the relation space is described by $\mathsf{vp}(\mathsf{rv}(R))$, where $\mathsf{rv}(R) := (\pi R \cap \rho)\mathsf{L} : X \times Y \leftrightarrow \mathsf{1}$ is the vector corresponding to R; the opposite direction uses vp^{-1} and the left-inverse of rv which is $\mathsf{rv}^{-1}(v) = \pi^{\mathrm{T}}(\rho \cap v\mathsf{L}) : X \leftrightarrow Y$.

The set Y^X of all functions from X to Y may be characterized in two different ways. On one hand, it can be described by a vector $f : 2^{X \times Y} \leftrightarrow \mathsf{1}$ such that

$$\begin{aligned} f_r &\iff \bigwedge_x \bigwedge_y \bigwedge_{y'} (((x,y) \in r \wedge (x,y') \in r) \to y = y') \wedge \bigwedge_x \bigvee_y (x,y) \in r \\ &\iff \neg(\bigvee_t t \in r \wedge \bigvee_{t'} (t' \in r \wedge \pi(t') = \pi(t) \wedge \rho(t') \neq \rho(t))) \wedge \bigwedge_x \bigvee_t (t \in r \wedge \pi(t) = x) \end{aligned}$$

giving the intricate expression

$$f = \overline{[\varepsilon_R^{\mathrm{T}} \cap \varepsilon_R^{\mathrm{T}}(\pi\pi^{\mathrm{T}} \cap \overline{\rho\rho^{\mathrm{T}}})]\mathsf{L}} \cap (\varepsilon_R^{\mathrm{T}}\pi \,/\, \mathsf{L}).$$

On the other hand, we can define an "is-element-of" relation $\varepsilon_F : X \times Y \leftrightarrow Y^X$ as follows: $\mathfrak{F} = (\pi, \rho, \varepsilon_F)$ is called a *function space* if and only if (R_1) and (R_2) (numbered by (F_1) and (F_2), respectively) hold with ε_R replaced by ε_F and, furthermore,

$$(F_3) \quad \mathsf{syq}(\varepsilon_F, \varepsilon_F) \subset \mathrm{I}$$
$$(F_4) \quad \bigwedge_R \mathsf{L}\,\mathsf{syq}(\varepsilon_F, R) = \mathsf{L} \leftrightarrow (RR^{\mathrm{T}} \subset \overline{\pi\pi^{\mathrm{T}}} \cup \rho\rho^{\mathrm{T}} \wedge R^{\mathrm{T}}\pi = \mathsf{L}).$$

Now, for R being a vector $v : X \times Y \leftrightarrow \mathsf{1}$ the second condition (F_4) means that the point $\mathsf{syq}(\varepsilon_F, v) : 2^{X \times Y} \leftrightarrow \mathsf{1}$ is in the function domain Y^X if and only if v represents a function, i.e., the relation $\mathsf{rv}^{-1}(v) : X \leftrightarrow Y$ is a function. For details, see [Abold-Thalmann et al. 89].

Injections: In addition to vectors, we have injective functions as second concept for representing subsets. Given an injective function $\iota : X' \to X$, we call X' a *subset of X given by ι*. Clearly, if X' is a subset of X given by ι, then the vector $\iota^{\mathrm{T}}\mathsf{L} : X \leftrightarrow \mathsf{1}$ describes X' in the above sense. Since we deal only with concrete relations, the transition in the other direction, i.e., the construction of an injective function $\iota : X' \to X$ from a given vector $v : X \leftrightarrow \mathsf{1}$, is also possible. Generally, we have: Let the vector $v : X \leftrightarrow \mathsf{1}$ describe the subset X' of the set X. Then, $\imath(v) : X' \leftrightarrow X$ is called an *injection of v* or an *injection of X' into X* if and only if

$$(I_1) \quad \imath(v) \text{ is an injective function} \qquad (I_2) \quad v = \imath(v)^{\mathrm{T}}\mathsf{L}.$$

Clearly, it follows that X' is a subset of X given by $\imath(v)$. Again, it can be shown that by (I_1) and (I_2) injections are determined uniquely up to isomorphism. Namely, if $\Psi : X \leftrightarrow Y$ is a bijective function and $v' : Y \leftrightarrow \mathsf{1}$ is a vector such that it describes a subset Y' of Y and $v = \Psi v'$ is satisfied (i.e., $v' = \Psi^{\mathrm{T}} v$), then $\mathfrak{I} = (\Phi, \Psi)$ is an isomorphism between $\imath(v)$ and $\imath(v')$, where the bijective function $\Phi : X' \leftrightarrow Y'$ is given by $\Phi = \imath(v)\Psi\imath(v')^{\mathrm{T}}$.

In most cases, injections are used within our applications in the context of higher-order objects like sets of sets. Namely, if the vector $v : 2^X \leftrightarrow \mathsf{1}$ describes a subset S of sets, then it is straightforward to compute an injection $\imath(v)$ of S into 2^X. From $\imath(v)$ we obtain the elements of S (represented as vectors) as the columns of the relation $\varepsilon\imath(v)^{\mathrm{T}} : X \leftrightarrow S$, which leads to an economic representation of the set S of sets by a Boolean matrix.

3 Relational Problem Specification and Prototyping

In this section, we show how the computation of certain mathematical objects (like sets, correspondences, mappings) can be described in terms of relations. The general method is as follows. First, we specify the problem with the help of a first-order formula φ which characterizes the objects to be computed. Using the correspondences between certain kinds of first-order formulae and certain relation-algebraic constructions (resp. operations), we then transform φ into a component-free relational expression R such that φ is valid if and only if its free variables are related by R. Hence, this expression R can be seen as a relational problem specification (cf. [Berghammer Schmidt 93]) which is executable as it stands, i.e., as an algorithm for computing the set of specified objects.

How to perform an executable relational specification within a manipulation system for (concrete) relations will be shown in the next section by means of the RELVIEW system.

Of course, algorithms produced in the way just described frequently may be fairly inefficient compared to hand-made ones and, thus, in many cases only applicable to small or medium-sized

examples. However, they are built up very quickly, which is an important factor of economy. Furthermore, they ensure correctness and their proofs of correctness are very simple. Moreover, an executable relational specification can be the starting point for the derivation of an efficient algorithm using some development method. Hence, we have the typical situation of the rapid prototyping approach (see [Budde et al. 84]) for a validation and analysis of specifications.

Most of the following examples of rapid prototyping using a relation-algebraic description of the given problem are borrowed from graph theory. They deal with the computation (strictly speaking: the enumeration) of higher-order objects like sets of sets (represented as vectors $v : 2^X \leftrightarrow \mathsf{1}$) or sets of functions (represented as vectors $v : Y^X \leftrightarrow \mathsf{1}$). Here our approach leads to an extensive use of the relational characterization of sets and functions as presented in Section 2. Therefore, in the following, we have to distinguish between the two meta-level symbols $\in$ and $\subseteq$ and the "is-element-of" relation and the set inclusion relation on the object-level. As in Section 2, we use in the sequel on the object level the two relations $\varepsilon : X \leftrightarrow 2^X$ and $\sqsubseteq : 2^X \leftrightarrow 2^X$. Especially, we have $x \in s$ (resp. $s \subseteq t$) if and only if the relation ε_{xs} (resp. $\sqsubseteq_{st}$) holds.

3.1 Point Sets of a Directed Graph

In this subsection, we consider some well-known examples on a directed graph. To this end, let $\mathcal{G} = (V, B)$ be a *directed graph*, i.e., V a non-empty set of points (also: vertices, nodes) and $B : V \leftrightarrow V$ a relation between points. Furthermore, assume $\varepsilon : V \leftrightarrow 2^V$ to be the "is-element-of" relation between V and its powerset 2^V and $\sqsubseteq : 2^V \leftrightarrow 2^V$ to be the inclusion relation between point sets.

Kernels: A set $a \in 2^V$ of points is said to be *absorbant* in $\mathcal{G}$ if from every point outside of a there is at least one arc leading into a, i.e., if the first-order formula

$$\bigwedge_x x \notin a \to \bigvee_y (y \in a \wedge B_{xy})$$

holds. Furthermore, a set $s \in 2^V$ of points is called *stable* in $\mathcal{G}$ if no two points of s are related via the relation B. This situation is characterized by the first-order formula

$$\bigwedge_x x \in s \to \bigwedge_y (y \in s \to \overline{B}_{xy}).$$

Finally, a *kernel* $k \in 2^V$ of $\mathcal{G}$ is a set which is at the same time absorbant and stable. The concept of a kernel plays an import rôle in combinatorial games, for an overview see e.g., [Schmidt Ströhlein 85] and [Schmidt Ströhlein 89], Sections 8.2 and 8.3.

Expressing the above two formulae in terms of the operations on relations introduced in Section 2, we get that the first one is equivalent to $(\mathsf{L} \setminus (\varepsilon \cup B\varepsilon))^{\mathrm{T}}_a$ and the second one is equivalent to $((\varepsilon \cap B\varepsilon) \setminus \mathsf{O})_s$. In the second case, for instance, we express $\bigwedge_y y \in s \to \overline{B}_{xy}$ as $(\overline{B\varepsilon})_{xs}$ and obtain, thus, the original formula in the form $\bigwedge_x (\overline{\varepsilon \cap B\varepsilon})_{xs}$. Now, the universal quantification can be removed using a right residual construction, cf. Section 2.

Summing up, in a component-free notation, we have

$$\mathit{absorb}(B) = (\mathsf{L} \setminus (\varepsilon \cup B\varepsilon))^{\mathrm{T}}$$

as the vector $\mathit{absorb}(B) : 2^V \leftrightarrow \mathsf{1}$ describing the absorbant sets of $\mathcal{G}$ (where $\mathsf{L} : V \leftrightarrow \mathsf{1}$) and

$$\mathit{stable}(B) = (\varepsilon \cap B\varepsilon) \setminus \mathsf{O}$$

as the vector $stable(B) : 2^V \leftrightarrow \mathsf{1}$ describing the stable sets of $\mathcal{G}$ (where $\mathsf{O} : V \leftrightarrow \mathsf{1}$). Now, the vector $kernel(B) : 2^V \leftrightarrow \mathsf{1}$ describing the elements of 2^V which are kernels of $\mathcal{G}$ is given by

$$kernel(B) = absorb(B) \cap stable(B).$$

Components: As a second example, we consider the problem of computing strongly connected components. Obviously, the procedure we will present in the following easily generalizes to a method for computing equivalence classes.

A set $c \in 2^V$ is called a *strongly connected component* of $\mathcal{G}$ if it contains as many points as possible that can all be reached from one another. This is equivalent to the fact that there is a point $x \in V$ such that for each point $y \in V$ the relation $y \in s$ holds if and only if B^*_{yx} as well as B^*_{xy} hold. Using a symmetric quotient construction, we can express the corresponding first-order formula

$$\bigvee_x \bigwedge_y (y \in s \leftrightarrow (B^*_{yx} \wedge B^*_{xy}))$$

as $\bigvee_x \mathsf{syq}(\varepsilon, B^* \cap (B^{\mathrm{T}})^*)_{sx}$, i.e., as $(\mathsf{syq}(\varepsilon, B^* \cap (B^{\mathrm{T}})^*)\mathsf{L})_s$, and this, finally, yields

$$scc(B) = \mathsf{syq}(\varepsilon, B^* \cap (B^{\mathrm{T}})^*)\mathsf{L}$$

(where $\mathsf{L} : V \leftrightarrow \mathsf{1}$) as the vector $scc(B) : 2^V \leftrightarrow \mathsf{1}$ describing the strongly connected components of $\mathcal{G}$.

Now, let $\mathcal{C} \subseteq 2^V$ denote the set of the strongly connect components and consider the homomorphism $\Phi_{red} : V \leftrightarrow \mathcal{C}$ defined by $\Phi_{red} = \varepsilon\imath(scc(B))^{\mathrm{T}}$, where $\imath(scc(B)) : \mathcal{C} \leftrightarrow 2^V$ is an injection of $\mathcal{C}$ into 2^V in the sense of Section 2. Its columns are exactly the strongly connected components of $\mathcal{G}$ and it maps every point to the component it belongs to. Hence, we get the reduced graph of $\mathcal{G}$ as $\mathcal{G}_{red} = (\mathcal{C}, \Phi^{\mathrm{T}}_{red} B \Phi_{red})$.

Point bases: A subset b of V is said to be a *point base* of $\mathcal{G}$ if it is a minimal (wrt. set inclusion) set of points from which every point can be reached via a path. The fact that every point can be reached from b is expressed by the first-order formula

$$\bigwedge_x \bigvee_y (y \in b \wedge B^*_{yx}).$$

Translating this formula into a relational one, we arrive at $((\varepsilon^{\mathrm{T}} B^*)/\mathsf{L}^{\mathrm{T}})_b$. Hence, by a minimalization of the vector $(\varepsilon^{\mathrm{T}} B^*)/\mathsf{L}^{\mathrm{T}} : 2^V \leftrightarrow \mathsf{1}$ wrt. the inclusion relation $\sqsubseteq : 2^V \leftrightarrow 2^V$ via the functional min, we obtain (a first characterization of) the vector $base(B) : 2^V \leftrightarrow \mathsf{1}$ of the point bases of $\mathcal{G}$ by

$$base(B) = \mathsf{min}(\sqsubseteq, (\varepsilon^{\mathrm{T}} B^*)/\mathsf{L}^{\mathrm{T}}),$$

where $\mathsf{L} : V \leftrightarrow \mathsf{1}$.

It can easily be shown that $b \in 2^V$ is a point base of $\mathcal{G}$ if and only if every point in the complement $\overline{b}$ of b can be reached from a point in b and that no point from b can be reached from some other point in b. In other words, a set b is a point base of $\mathcal{G}$ if and only if it is a kernel of $\mathcal{G}' = (V, (B^{\mathrm{T}})^+ \cap \overline{\mathsf{I}})$. This leads to the following second characterization of the vector $base(B) : 2^V \leftrightarrow \mathsf{1}$ of point bases of $\mathcal{G}$ as

$$base(B) = kernel((B^{\mathrm{T}})^+ \cap \overline{\mathsf{I}}).$$

Hammocks: A set $h \in 2^V$ is a *hammock* of $\mathcal{G}$ if it is connected with the remaining part of the graph only through two different nodes $x, y \notin h$, one for outgoing edges (z, x) and one for ingoing edges (y, z) . If the given graph $\mathcal{G}$ is the control flow graph of a program P, i.e., contains two specific points p and q such that p has no predecessors, q has no successors, and each point lies on at least one path from p to q, then a partition of $\mathcal{G}$ into hammocks is necessary for an application of optimizing algorithms during a compilation of P, see [Ershov 70].

To define a vector $hammock(B) : 2^V \leftrightarrow \mathsf{1}$ which produces all hammocks of $\mathcal{G}$, first, we consider the formula

$$x \notin h \wedge \bigvee_y (y \in h \wedge B_{yx})$$

which says that the point x is a proper successor of the point set h. Using the operations on relations, it can be expressed as $(\overline{\varepsilon} \cap B^{\mathrm{T}} \varepsilon)_{xh}$. Hence, in a component-free notation we get

$$succ(B) = (\overline{\varepsilon} \cap B^{\mathrm{T}} \varepsilon)^{\mathrm{T}}$$

as the relation $succ(B) : 2^V \leftrightarrow V$ which associates h to x if and only if x is a proper successor of h. Now, we apply to this relation the so-called univalent part (see [Schmidt Ströhlein 89]), i.e., define a new relation $singlesucc(B) : 2^V \leftrightarrow V$ by

$$singlesucc(B) = \mathsf{syq}(succ(B)^{\mathrm{T}}, \mathrm{I}) = \mathsf{syq}(\overline{\varepsilon} \cap B^{\mathrm{T}} \varepsilon, \mathrm{I}).$$

The univalent part $\mathsf{up}(R) := \mathsf{syq}(R^{\mathrm{T}}, \mathrm{I})$ of a relation R is characterized as follows: A point x is related to a point y by $\mathsf{up}(R)$ if and only if R_{xy} holds, but there is no z different from y such that also R_{xz} is valid. Hence $singlesucc(B)$ associates h to x if and only if x is the only proper successor of h. Obviously, y is the only proper predecessor of h if and only if $singlesucc(B^{\mathrm{T}})_{hy}$ holds. Hence, with the help of the auxiliary functional *singlesucc* we obtain that $h \in 2^V$ is a hammock if and only if

$$\bigvee_x \bigvee_y (singlesucc(B)_{hx} \wedge singlesucc(B^{\mathrm{T}})_{hy} \wedge y \neq x)$$

is fulfilled and the equivalent version

$$\bigvee_x singlesucc(B)_{hx} \wedge \bigvee_y (singlesucc(B^{\mathrm{T}})_{hy} \wedge \overline{\mathrm{I}}_{yx}),$$

finally, leads to the following definition of the vector $hammock(B)$ of all hammocks:

$$hammock(B) = (singlesucc(B) \cap singlesucc(B^{\mathrm{T}})\overline{\mathrm{I}})\mathsf{L},$$

where $\mathrm{I} : V \leftrightarrow V$ and $\mathsf{L} : V \leftrightarrow \mathsf{1}$.

3.2 Point Sets of a Partially Ordered Set

Now, we deal with two order-theoretical problems. Therefore, let $\mathcal{O} = (M, E)$ be a partially ordered set, i.e., M be a non-empty set of points and $E : M \leftrightarrow M$ be an ordering relation on points. Furthermore, assume again $\varepsilon : M \leftrightarrow 2^M$ to be the "is-element-of" relation between M and its powerset 2^M and $\sqsubseteq : 2^M \leftrightarrow 2^M$ to be the inclusion relation on point sets.

Dedekind cuts: For an element $s \in 2^M$, let $Ma(s)$ denote its upper bounds wrt. E and $Mi(s)$ denote its lower bounds wrt. E. Then $c \in 2^M$ is called a *Dedekind cut* of $\mathcal{O}$ if the equation $c = Mi(Ma(c))$ is valid, i.e, if the first-order formula

$$\bigwedge_x x \in c \leftrightarrow x \in Mi(Ma(c))$$

holds which in turn is equivalent to

$$\bigvee_s \bigwedge_x (x \in c \leftrightarrow x \in Mi(Ma(s))) \wedge c = s.$$

Obviously, for each element $x \in M$ the set $(x) = \{y \in M : E_{yx}\}$ is a cut, called the *principal cut* generated by x. The fact that a set $p \in 2^M$ is a principal cut, hence, is described by the first-order formula

$$\bigvee_x x \in p \wedge \bigwedge_y (y \in p \leftrightarrow E_{yx}).$$

Now, let $\mathcal{C} \subseteq 2^M$ denote the set of cuts of $\mathcal{O}$ and $\mathcal{P} \subseteq 2^M$ denote the set of principal cuts of $\mathcal{O}$. Furthermore, let $\sqsubseteq_{\mathcal{C}} : \mathcal{C} \leftrightarrow \mathcal{C}$ and $\sqsubseteq_{\mathcal{P}} : \mathcal{P} \leftrightarrow \mathcal{P}$ denote the restrictions of set inclusion to the cuts and principal cuts, respectively. Then $\mathcal{O}' = (\mathcal{C}, \sqsubseteq_{\mathcal{C}})$ is a complete lattice, called the *cut completion* of $\mathcal{O}$, and the function $emb(E) : M \to \mathcal{C}$ mapping x to the principal cut (x) is an injective order homomorphism. Using abstract relation algebra and the above formulae (for the characterization of cuts the second version is more suited since it immediately leads to a symmetric quotient construction $\mathsf{syq}(\varepsilon, \ldots)_{cs}$) in combination with the functionals mi, ma, and syq, by

$$cut(E) = (\mathsf{syq}(\varepsilon, \mathsf{mi}(E, \mathsf{ma}(E, \varepsilon))) \cap \mathrm{I})\mathsf{L}$$

(where $\mathsf{L} : 2^M \leftrightarrow \mathbf{1}$ and $\mathrm{I} : 2^M \leftrightarrow 2^M$) we obtain the vector $cut(E) : 2^M \leftrightarrow \mathbf{1}$ describing the elements of 2^M which are Dedekind cuts, and by

$$pricut(E) = (\varepsilon^{\mathrm{T}} \cap \mathsf{syq}(\varepsilon, E))\mathsf{L}$$

(where $\mathsf{L} : M \leftrightarrow \mathbf{1}$) we get the vector $pricut(E) : 2^M \leftrightarrow \mathbf{1}$ describing the elements of 2^M which are principal cuts. Since the cuts are ordered by set inclusion, we consider the relation $\sqsubseteq : 2^M \leftrightarrow 2^M$ and the injection $\imath(cut(E)) : \mathcal{C} \leftrightarrow 2^M$ in the sense of Section 2 and receive for $\sqsubseteq_{\mathcal{C}}$ the representation $\sqsubseteq_{\mathcal{C}} \; = \imath(cut(E)) \sqsubseteq \imath(cut(E))^{\mathrm{T}}$.

Also the function $emb(E)$ can be computed with the help of $\imath(cut(E))$. From the componentwise definition of $emb(E)_{xc}$ by the first-order formula

$$\bigwedge_y E_{yx} \leftrightarrow \bigvee_s (y \in s \wedge \imath(cut(E))_{cs})$$

we obtain $\bigwedge_y E_{yx} \leftrightarrow (\varepsilon\imath(cut(E))^{\mathrm{T}})_{yc}$ and, thus, an application of the functional syq yields the component-free relation-algebraic representation

$$emb(E) = \mathsf{syq}(E, \varepsilon\imath(cut(E))^{\mathrm{T}}).$$

Chains: A nonempty set $c \in 2^M$ is called a *chain* in $\mathcal{O}$ if E_{xy} or E_{yx} (or both) holds for every two points $x, y \in M$. Said another way, c is a chain if the first-order formula

$$\bigvee_x x \in c \wedge \bigwedge_x x \in c \to \bigwedge_y (y \in c \to (E_{xy} \vee E_{yx}))$$

is valid. Translating this into a component-free relational notation, we get (for the second part of the outermost meet cf. the formula yielding the vector $stable(B)$ and replace there s by c and B by $\overline{E \cup E^{\mathrm{T}}}$)

$$chain(E) = \varepsilon^{\mathrm{T}} \mathsf{L} \cap ((\varepsilon \cap \overline{E \cup E^{\mathrm{T}}}\varepsilon) \setminus \mathsf{O}) = \overline{\mathbb{E}} \cap ((\varepsilon \cap \overline{E \cup E^{\mathrm{T}}}\varepsilon) \setminus \mathsf{O})$$

(where $\mathsf{O} : M \leftrightarrow \mathsf{1}$ and $\mathsf{L} : M \leftrightarrow \mathsf{1}$) as the vector $chain(E) : 2^M \leftrightarrow \mathsf{1}$ describing the chains of $\mathcal{O}$. Thus, a maximalization wrt. set inclusion via the functional max, finally, yields

$$maxchain(E) = \mathsf{max}(\sqsubseteq, chain(E))$$

as the vector $maxchain(E) : 2^M \leftrightarrow \mathsf{1}$ of the inclusion-maximal chains.

3.3 Point Sets of a Simple Graph

The problems of this subsection deal with a *simple graph* $\mathcal{G} = (V, \Gamma)$ which means that V is a non-empty set of points (vertices, nodes) and $\Gamma : V \leftrightarrow V$ is an irreflexive and symmetric relation on points. Let $\varepsilon : V \leftrightarrow 2^V$ be the "is-element-of" relation between V and its powerset 2^V and let $\sqsubseteq : 2^V \leftrightarrow 2^V$ be the inclusion relation on point sets.

Point coverings: A set $c \in 2^V$ of points is called a *(point) covering* of $\mathcal{G}$ if every edge is incident with at least one point from c. Hence, c is a covering if the first-order formula

$$\bigwedge_x \bigwedge_y \Gamma_{xy} \rightarrow (x \in c \vee y \in c)$$

is valid. This formula is equivalent to

$$\bigwedge_x x \notin c \rightarrow \bigwedge_y (y \notin c \rightarrow \overline{\Gamma}_{xy})$$

and, thus, using a component-free relational notation, we arrive (see again the formula corresponding to the vector $stable(B)$ and write there $\notin$ for $\in$, c for s, and Γ for B) at

$$cover(\Gamma) = (\overline{\varepsilon} \cap \Gamma\overline{\varepsilon}) \setminus \mathsf{O},$$

(where $\mathsf{O} : V \leftrightarrow \mathsf{1}$) as the vector $cover(B) : 2^V \leftrightarrow \mathsf{1}$ describing the set of coverings of $\mathcal{G}$.

The coverings which lose the covering property when some point is removed are called inclusion-minimal. Again, the vector $mincover(\Gamma) : 2^V \leftrightarrow \mathsf{1}$ of the inclusion-minimal coverings is given by an application of the functional min, where the first argument is set inclusion and the second argument is the vector of coverings. I.e., we have

$$mincover(\Gamma) = \mathsf{min}(\sqsubseteq, cover(\Gamma)).$$

Cliques: A subset $c \in 2^V$ is said to be a *clique* of $\mathcal{G}$ if it generates an inclusion-maximal complete subgraph. The fact that c generates a complete subgraph is expressed by the first-order formula

$$\bigwedge_x x \in c \rightarrow \bigwedge_y (y \in c \rightarrow (x = y \vee \Gamma_{xy})).$$

Translating this formula into a component-free relational expression followed by a maximalization wrt. the inclusion relation $\sqsubseteq : 2^V \leftrightarrow 2^V$ via the functional max, we receive, finally, the vector $clique(\Gamma) : 2^V \leftrightarrow \mathsf{1}$ describing the cliques of $\mathcal{G}$ by

$$clique(\Gamma) = \mathsf{max}(\sqsubseteq, (\varepsilon \cap \overline{\mathrm{I} \cup \Gamma}\varepsilon) \setminus \mathsf{O}),$$

where $\mathrm{I} : V \leftrightarrow V$ and $\mathsf{O} : V \leftrightarrow \mathsf{1}$. For the right residual, i.e., the second argument of the functional max, see again the component-wise definition of a set to be stable.

3.4 Point and Edge Sets of a Hypergraph

Next, we consider two problems on a hypergraph. A *hypergraph* (see [Berge 73]) is a relational structure $\mathcal{H} = (K, V, M)$ which consists of a non-empty set K of edges, a non-empty set V of points (vertices, nodes), and a total relation $M : K \leftrightarrow V$ between edges and points.

Point coverings: Assume $\varepsilon : V \leftrightarrow 2^V$ to be the "is-element-of" relation between V and its powerset 2^V and $\sqsubseteq : 2^V \leftrightarrow 2^V$ to be the inclusion relation on point sets. As in the case of simple graphs, a set $c \in 2^V$ of points is called a *(point) covering* of $\mathcal{H}$ if every edge is incident with at least one point from c. Expressed as a first-order formula, a set $c \in 2^V$ is a covering of $\mathcal{H}$ if and only if

$$\bigwedge_k \bigvee_x (x \in c \wedge M_{kx})$$

holds. Coverings of hypergraphs are also said to be *set coverings* since every edge k from K can be regarded as an element from 2^V, viz. the set $\{x \in V : M_{kx}\}$ of vertices incident with it.

For expressing the above formula into a component-free relation-algebraic term, we look at the component-wise description of point bases (since there we have the same pattern) and get the vector $setcover(M) : 2^V \leftrightarrow \mathsf{1}$ describing the set coverings of $\mathcal{H}$ by

$$setcover(M) = (M\varepsilon)^{\mathrm{T}} / \mathsf{L}^{\mathrm{T}},$$

where $\mathsf{L} : K \leftrightarrow \mathsf{1}$. From this, the vector $minsetcover(M) : 2^V \leftrightarrow \mathsf{1}$ of the inclusion-minimal set coverings immediately is obtained by a minimalization using the functional min, i.e., as

$$minsetcover(M) = \mathsf{min}(\sqsubseteq, setcover(M)).$$

Edge coverings: A set $c \in 2^K$ of edges is called an *edge covering* of $\mathcal{H}$ if every point is incident with at least one edge from c, i.e., if

$$\bigwedge_x \bigvee_k (k \in c \wedge M_{kx})$$

is fulfilled. Based on the description of $setcover(M)$ and $minsetcover(M)$, an abstract relational characterization of the (minimal) edge coverings is obvious using the "is-element-of" relation between K and its powerset and the inclusion relation on edge sets. For $edgecover(M) : 2^K \leftrightarrow \mathsf{1}$ describing the edge coverings of $\mathcal{H}$ we have

$$edgecover(M) = (\varepsilon^{\mathrm{T}} M) / \mathsf{L}^{\mathrm{T}},$$

where $\mathsf{L} : L \leftrightarrow \mathsf{1}$, and for the vector $minedgecover(M) : 2^K \leftrightarrow \mathsf{1}$ of the inclusion-minimal edge coverings of $\mathcal{H}$ we obtain the representation

$$minedgecover(M) = \mathsf{min}(\sqsubseteq, edgecover(M)).$$

3.5 Sets of Places of a Petri Net

A *Petri net* is a bipartite directed graph $\mathcal{N} = (X, Y, R, S)$. The point set of $\mathcal{N}$ is decomposed into the sets X of *places* and Y of *transitions* and the associated relations are $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow X$. Petri nets have been widely used to design and model concurrent systems, see [Reisig 85] for example. Many of the static properties of a Petri net (e.g., to be free-choice, to

be conflict-free, or to contain specific sets of places/transitions) can be tested using our relational approach. In the following we show, how to compute specific sets of places. In doing so, $\varepsilon_X : X \leftrightarrow 2^X$ denotes the "is-element-of" relation between the places and the sets of places.

Deadlocks and traps: A set $d \in 2^X$ of places is called a *deadlock* if each of its predecessors is also a successor. A somewhat dual notion is that of a trap: $t \in 2^X$ is said to be a *trap* if its successor set is a subset of its predecessor set. Both, deadlocks and traps are of interest if one is concerned with liveness properties. E.g., a transition never can been enabled if its predecessor set contains an unmarked place of a deadlock.

Expressed by first-order formulae, we have that a set d of places is a deadlock if and only if

$$\bigwedge_y (\bigvee_x x \in d \wedge S_{yx}) \rightarrow (\bigvee_x x \in d \wedge R_{xy})$$

is valid and that a set t of places is a trap if and only if

$$\bigwedge_y (\bigvee_x x \in t \wedge R_{xy}) \rightarrow (\bigvee_x x \in t \wedge S_{yx})$$

holds. Using the operations on relations, the first formula becomes $\bigwedge_y (\varepsilon_X^{\mathrm{T}} S^{\mathrm{T}})_{dy} \rightarrow (\varepsilon_X^{\mathrm{T}} R)_{dy}$ which in turn is equivalent to $\bigwedge_y (\overline{\varepsilon_X^{\mathrm{T}} S^{\mathrm{T}}} \cup \varepsilon_X^{\mathrm{T}} R)_{dy}$. Hence, using a left residual construction and the universal relation $\mathsf{L} : Y \leftrightarrow \mathbf{1}$ we get in a component-free notation

$$\mathit{deadlock}(R,S) = (\overline{\varepsilon_X^{\mathrm{T}} S^{\mathrm{T}}} \cup \varepsilon_X^{\mathrm{T}} R) / \mathsf{L}^{\mathrm{T}}$$

as the vector $\mathit{deadlock}(R,S) : X \leftrightarrow \mathbf{1}$ enumerating all deadlocks of the net $\mathcal{N}$. In the same way one obtains the vector $\mathit{trap}(R,S) : X \leftrightarrow \mathbf{1}$ describing all traps of $\mathcal{N}$ by exchanging the rôle of R and S^{T}, i.e., by

$$\mathit{trap}(R,S) = (\overline{\varepsilon_X^{\mathrm{T}} R} \cup \varepsilon_X^{\mathrm{T}} S^{\mathrm{T}}) / \mathsf{L}^{\mathrm{T}}.$$

3.6 Correspondences on a Directed Graph

In the preceeding examples we have dealt with sets of points and edges, i.e., elements of the powerset of a given "base set". The example of this subsection shows how to describe the computation of pairs of elements of the powerset of a given "base set" with relation-algebraic means, i.e., the computation of a relation (correspondence) on a powerset. We consider again a directed graph $\mathcal{G} = (V, B)$ and suppose the "is-element-of" relation $\varepsilon : V \leftrightarrow 2^V$ and the inclusion relation $\sqsubseteq : 2^V \leftrightarrow 2^V$ on point sets.

Dicliques: A pair $(d, c) \in 2^V \times 2^V$ is called a *block* of $\mathcal{G}$ with domain d and co-domain c if the product $d \times c$ is contained in B. The (nontrivial) *dicliques* of $\mathcal{G}$ (this term is introduced in [Haralick 74]) are the inclusion-maximal blocks with non-empty domains and co-domains. In other words, (d, c) is a diclique if and only if it generates an inclusion-maximal complete bipartite subgraph of $\mathcal{G}$. A decomposition of a graph into its dicliques can be very useful e.g., for storing it in a computer or for determining the essential subsystems of the system it describes. See again [Haralick 74].

The description of a block $(d, c) \neq (\emptyset, \emptyset)$ by a first-order formula is

$$\bigvee_x x \in d \wedge \bigvee_x x \in c \wedge \bigwedge_x x \in d \rightarrow \bigwedge_y (y \in c \rightarrow B_{xy}).$$

Obviously, $\bigwedge_y y \in c \to B_{xy}$ is equivalent to $(\overline{\overline{B}\varepsilon})_{xc}$, i.e., to $(B \setminus \varepsilon^{\mathrm{T}})_{xc}$, which implies the equivalence of $\bigwedge_x x \in d \to \bigwedge_y (y \in c \to B_{xy})$ and $(\varepsilon \setminus (B/\varepsilon^{\mathrm{T}}))_{dc}$. This immediately leads to

$$block(B) = \varepsilon^{\mathrm{T}} \mathsf{L} \cap \mathsf{L}^{\mathrm{T}} \varepsilon \cap (\varepsilon \setminus (B/\varepsilon^{\mathrm{T}})) = \overline{\mathbb{E}} \cap \overline{\mathbb{E}}^{\mathrm{T}} \cap \overline{\varepsilon^{\mathrm{T}} \overline{B} \varepsilon}$$

(where $\mathsf{L} : V \leftrightarrow 2^V$) as the relation $block(B) : 2^V \leftrightarrow 2^V$ of the blocks with non-empty domain and co-domain. I.e., $block(B)_{dc}$ holds if and only if $(d, c) \neq (\emptyset, \emptyset)$ is a block. To describe dicliques, we use that for a relation $R : 2^V \leftrightarrow 2^V$ and a pair $(s, t) \in 2^V \times 2^V$ we have $\mathsf{max}(\sqsubseteq, R)_{st}$ if and only if s is inclusion-maximal in the set $\{s' \in 2^V : R_{s't}\}$. Hence, a two-fold maximalization via the functional max yields

$$diclique(B) = \mathsf{max}(\sqsubseteq, \mathsf{max}(\sqsubseteq, block(B))^{\mathrm{T}})^{\mathrm{T}}$$

as the relation $diclique(B) : 2^V \leftrightarrow 2^V$ of the dicliques of the directed graph $\mathcal{G}$.

Now, let $D \subseteq 2^V$ (resp. $C \subseteq 2^V$) be the set of domains (resp. co-domains) of the dicliques of $\mathcal{G}$. Then we have the two injections $\imath(diclique(B)\mathsf{L}) : D \leftrightarrow 2^V$ and $\imath(diclique(B)^{\mathrm{T}}\mathsf{L}) : C \leftrightarrow 2^V$ for embedding D and C, respectively, into 2^V, Based on these injections, finally, we are able to define a relation $Diclique(B) : D \leftrightarrow C$ describing the correspondence between the domain and the co-domain of a diclique by

$$Diclique(B) = \imath(diclique(B)\mathsf{L})diclique(B)\imath(diclique(B)^{\mathrm{T}}\mathsf{L})^{\mathrm{T}}.$$

This means: A pair $(d, c) \in D \times C$ is a diclique of $\mathcal{G}$ if and only if the relation $Diclique(B)_{cd}$ holds.

3.7 Sets of Mappings

The examples of this subsection show how to describe the computation of sets of functions from a set V to a set W with relation-algebraic means. As already mentioned, we consider the set W^V of functions from V to W as the set of the functional and total relations between V and W, i.e., as a subset of the powerset $2^{V \times W}$. Therefore, suppose $\varepsilon_F : V \times W \leftrightarrow W^V$ to be the "is-element-of" relation between $V \times W$ and the set W^V.

Homomorphisms: Let $\mathcal{G} = (V, B)$ and $\mathcal{H} = (W, C)$ be directed graphs and $\Phi : V \leftrightarrow W$ be a homomorphism from B to C. Furthermore, assume $\pi : V \times W \leftrightarrow V$ and $\rho : V \times W \leftrightarrow W$ to be the two projections from the direct product $V \times W$ to the first and the second component, respectively. If we use the common function notation for π and ρ, the fact that Φ is a homomorphism means exactly that the first-order formula

$$\bigwedge_u u \in \Phi \to \bigwedge_v (v \in \Phi \to (B_{\pi(u)\pi(v)} \to C_{\rho(u)\rho(v)}))$$

is valid; see Section 2. Now, we use the relational notation also for the two projections π and ρ, i.e., write $(\pi B \pi^{\mathrm{T}})_{uv}$ (resp. $(\rho C \rho^{\mathrm{T}})_{uv}$) instead of $B_{\pi(u)\pi(v)}$ (resp. $C_{\rho(u)\rho(v)}$). This leads to the equivalent version

$$\bigwedge_u u \in \Phi \to \bigwedge_v (v \in \Phi \to ((\pi B \pi^{\mathrm{T}})_{uv} \to (\rho C \rho^{\mathrm{T}})_{uv}))$$

which is in turn equivalent to

$$\bigwedge_u u \in \Phi \to \bigwedge_v (v \in \Phi \to (\overline{\pi B \pi^{\mathrm{T}}} \cup \rho C \rho^{\mathrm{T}})_{uv}).$$

Hence, we have again the pattern of the formula defining a set to be stable and, therefore, we get the vector $hom(B,C) : W^V \leftrightarrow \mathsf{1}$ describing the homomorphisms from B to C as

$$hom(B,C) = (\varepsilon_F \cap auxhom(B,C)\varepsilon_F) \setminus \mathsf{0},$$

where $\mathsf{0} : V \times W \leftrightarrow \mathsf{1}$ and the auxiliary operation *auxhom* is defined by

$$auxhom(B,C) = \overline{\overline{\pi B \pi^{\mathrm{T}}} \cup \rho C \rho^{\mathrm{T}}} = \pi B \pi^{\mathrm{T}} \cap \overline{\rho C \rho^{\mathrm{T}}}$$

and associates a pair u to a pair v if and only if the implication $B_{\pi(u)\pi(v)} \rightarrow C_{\rho(u)\rho(v)}$ does not hold.

Now, assume additionally $V = W$ and let $\Phi : V \leftrightarrow V$ be even an isomorphism from B to C. Then this means that Φ is bijective and, furthermore, the first-order formula

$$\bigwedge_u u \in \Phi \rightarrow \bigwedge_v (v \in \Phi \rightarrow (B_{\pi(u)\pi(v)} \leftrightarrow C_{\rho(u)\rho(v)}))$$

holds. See again Section 2. Thus, if we assume a vector $bij : V^V \leftrightarrow \mathsf{1}$ which produces the bijective functions on the set V, then this yields the vector $iso(B,C) : W^V \leftrightarrow \mathsf{1}$ of the isomorphisms between the relations B and C as

$$iso(B,C) = bij \cap ((\varepsilon_F \cap auxiso(B,C)\varepsilon_F) \setminus \mathsf{0}),$$

where $\mathsf{0} : V \times W \leftrightarrow \mathsf{1}$ again and the auxiliary operation *auxiso* is defined by

$$auxiso(B,C) = \overline{(\overline{\pi B \pi^{\mathrm{T}}} \cup \rho C \rho^{\mathrm{T}}) \cap (\overline{\rho C \rho^{\mathrm{T}}} \cup \pi B \pi^{\mathrm{T}})} = (\pi B \pi^{\mathrm{T}} \cap \overline{\rho C \rho^{\mathrm{T}}}) \cup (\rho C \rho^{\mathrm{T}} \cap \overline{\pi B \pi^{\mathrm{T}}})$$

and associates the pair u to the pair v if and only if the equivalence $B_{\pi(u)\pi(v)} \leftrightarrow C_{\rho(u)\rho(v)}$ does not hold.

Bijective functions: We look for a component-free relation-algebraic description of the set of bijective functions. This can be achieved analogously to the vector f of all functions contained in a relation space (cf. Section 2). Since a function R is bijective if and only if R^{T} is a function, too, we expect a result similar to f, where π and ρ are exchanged and ε_R is replaced by ε_F. We now describe how this can be confirmed by applying our method.

To this end, let $R : V \leftrightarrow V$ be a function and let $\pi : V \times V \leftrightarrow V$ and $\rho : V \times V \leftrightarrow V$ be the two projections from the direct product $V \times V$ to the first and the second component, respectively. If we use again the common function notation for π and ρ, then R is injective if and only if the first-order formula

$$\bigwedge_u u \in R \rightarrow \bigwedge_v (v \in R \rightarrow (\rho(u) = \rho(v) \rightarrow \pi(u) = \pi(v)))$$

holds, while R is surjective if and only if the first-order formula

$$\bigwedge_u \bigvee_v (v \in R \wedge \rho(v) = u)$$

holds. Looking at the definition of homomorphisms in the first case and of point bases in the second one (here we have respectively the same pattern of the corresponding first-order description), we obtain

$$inj = (\varepsilon_F \cap \overline{\overline{\rho\rho^{\mathrm{T}}} \cup \pi\pi^{\mathrm{T}}}\varepsilon_F) \setminus \mathsf{0} = (\varepsilon_F \cap (\rho\rho^{\mathrm{T}} \cap \overline{\pi\pi^{\mathrm{T}}})\varepsilon_F) \setminus \mathsf{0},$$

(where $\mathsf{O} : V^V \leftrightarrow \mathsf{1}$) as the vector $inj : V^V \leftrightarrow \mathsf{1}$ of the injective functions on V and

$$surj = (\varepsilon_F^{\mathrm{T}} \rho) / \mathsf{L}^{\mathrm{T}}$$

(where $\mathsf{L} : V \leftrightarrow \mathsf{1}$) as the vector $surj : V^V \leftrightarrow \mathsf{1}$ of the surjective functions on V. Altogether, we arrive, finally, at

$$bij = inj \cap surj$$

as the vector $bij : V^V \leftrightarrow \mathsf{1}$ of the bijective functions (or *permutations*) on V. It should yet be mentioned that another possibility to describe the set of bijective functions in a relation-algebraic way as a vector $bij : V^V \leftrightarrow \mathsf{1}$ is to use the first-order formula

$$\bigwedge_u u \in R \to \bigwedge_v (v \in S \to (\rho(u) = \pi(v) \to \pi(u) = \rho(v)))$$

which expresses the fact that the function $S : V \leftrightarrow V$ is a left-inverse of the function $R : V \leftrightarrow V$. Here we have the same pattern as in the case of blocks. Hence, we obtain

$$linv = \varepsilon_F \setminus ((\overline{\rho \pi^{\mathrm{T}}} \cup \pi \rho^{\mathrm{T}}) / \varepsilon_F^{\mathrm{T}}) = \overline{\varepsilon_F^{\mathrm{T}} (\rho \pi^{\mathrm{T}} \cap \overline{\pi \rho^{\mathrm{T}}}) \varepsilon_F}$$

as the relation $linv : V^V \leftrightarrow V^V$ which associates R to S if and only if $SR = I$ which in turn leads to

$$bij = linv \mathsf{L} \cap linv^{\mathrm{T}} \mathsf{L}$$

(where $\mathsf{L} : V^V \leftrightarrow \mathsf{1}$) as a second characterization of the vector bij.

4 Using the RELVIEW System For Rapid Prototyping

The RELVIEW system ([Berghammer Schmidt 91]) is a computer system for the manipulation of relations. In this section, first, a short description of the system is given. Using RELVIEW we then show, how to perform executable relational specifications as those developed in Section 3 from first-order descriptions. By means of examples we demonstrate also the development of a more efficient algorithm from the original inefficient specification within abstract relation algebra. Finally, we sketch some further experiences with the RELVIEW system.

4.1 A Short Look at RELVIEW

People working with relations (e.g., in the theory of partial orderings, lattice theory, or graph theory) very often a use greater or smaller example and manipulate it with pencil and paper in order to prove or disprove some property. For supporting such a task by machine (and also since manipulation by hand is no more feasible with bigger examples), the RELVIEW system has been constructed at the Bundeswehr-University at Munich. The system is written in C and is currently available for Sun workstations with American National Standard C and Sunview 4.0.

RELVIEW is a totally interactive and completely video-oriented computer system for the manipulation of concrete relations which are considered as Boolean matrices. Its screen is divided into two parts. The left part is the drawing-window; here matrices can be drawn and manipulated using a mouse. The right part contains the command buttons and the scrollbars. The scrollbars can be used for showing a part of a relation the size of which exceeds the maximal

window size. Also textual input (e.g., dimensions or names of relations) and output (e.g., results of tests, error messages) is requested and shown, respectively, in this part.

One relation, the so-called working copy, is displayed on the screen for editing. A whole collection of relations can be kept in the working memory during a working session. Such a collection may also be saved on permanent memory, e.g., on a hard disk. If a stored relation from the memory is displayed into the drawing-window for editing, a duplicate working copy is created. Editing with the mouse does only affect the working copy and thus does not change the original. To overwrite the original by the working copy, a specific RELVIEW command has to be used.

Execution of system commands is possible by clicking on command buttons. If a command requires arguments, then execution does not start before the last argument is given. Thus, if the user has inadvertently chosen a wrong button, undo consists in choosing the correct button – provided that the argument input has not been finished. Besides some management commands, first, the system provides commands implementing the basic operations on relations described in Subsection 2.1. The textual representation of the operations are:

Join:	`R|S`	*Negation:*	`-R`
Meet:	`R&S`	*Transposition:*	`R^`
Composition:	`R*S`		

Furthermore, we have commands for residuals, quotients, and closures (cf. Subsection 2.3), commands for certain tests on relations (cf. Subsection 2.2), and commands which implement the operations important in relation-algebraic domain description (cf. Subsection 2.4). And, finally, RELVIEW allows the user to define and apply its own functionals on relations, where in the case of a unary functional with identical domain and range also repeated application is possible by an iteration command. A useful fact in applications is that the latter command can be used to compute fixpoints of monotone functionals. For instance, if the homogeneous relation `R` is contained in the working memory and one declares a RELVIEW functional

$$\texttt{initial} \qquad -(\texttt{R} * -\%),$$

where % stands for the variable, then a repeated application of this functional to the empty vector yields the vector of the points from which only paths of finite length emerge. (Compare the definition of the initial part of a graph in [Schmidt Ströhlein 89], Section 6.3.)

A detailed description of how to draw on the drawing-window, how to use the scrollbars, and how to execute a command (inclusive parameter passing and result delivery) is given in [Abold-Thalmann et al. 89] and [Berghammer 92]. The first report also presents some implementation details, e.g., the internal representation of relations, and outlines fast algorithms for computing products, symmetric quotients, and residuals of relations. In the second report, also an example for prototyping using RELVIEW is presented, viz. the computation of the cut completion of a partially ordered set.

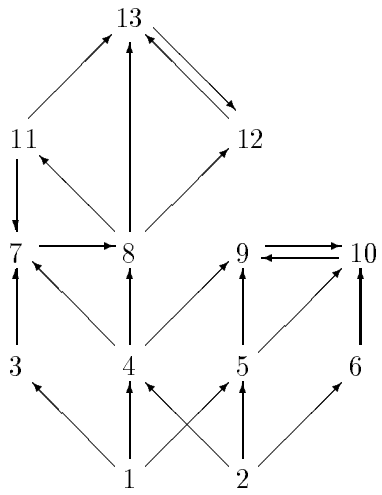
4.2 Examples

In this subsection, we present three small examples for prototyping relational specifications using RELVIEW.

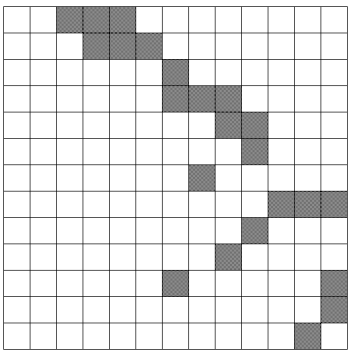
Kernels: The first example deals with kernels. We want to demonstrate how the RELVIEW system can be used to produce the set of all kernels of a direct graph $\mathcal{G} = (V, B)$. At this place

it should be mentioned that the problem of finding a kernel in a graph has been shown to be NP-complete, even if it is restricted to planar graphs; compare [Fraenkel 81].

We assume the point set V of our example graph $\mathcal{G}$ to consist of the natural numbers from 1 through 13. The drawing of $\mathcal{G}$ is given below. This directed graph contains a circuit $(7, 8, 11, 7)$ of odd lenght and, therefore, none of the well-known existence theorems for kernels (a lot of which can, for instance, be found in [Schmidt Ströhlein 85]) applies.



To produce the kernels of $\mathcal{G}$ using the RELVIEW system, we have to perform three steps. First, there are to create some Boolean matrices and to put into the system's working memory. We need a 13×13 matrix `B` for the relation B. Looking at Subsection 3.1, we also have to create the 13×1 universal vector, the 13×1 empty vector, and the 13×2^{13} "is-element-of" relation. We save these Boolean matrices as `L`, `O`, and `epsi`, respectively. The following picture shows `B` as presented by RELVIEW on the drawing-window.

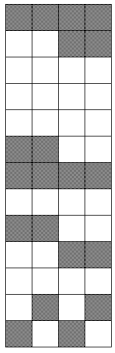


Looking again at Subsection 3.1, in the second step of our procedure we have to define the three functionals for computing the vectors of the absorbant and stable sets, and the kernels, respectively. In the RELVIEW system, the definition of these functionals is possible by a built-in command and reads as follows:

`absorb`	$(\texttt{L} \setminus (\texttt{epsi} \mid \% * \texttt{epsi}))\hat{}$
`stable`	$(\texttt{epsi} \,\&\, \% * \texttt{epsi}) \setminus \texttt{O}$
`kernel`	$\texttt{absorb}(\%) \,\&\, \texttt{stable}(\%)$

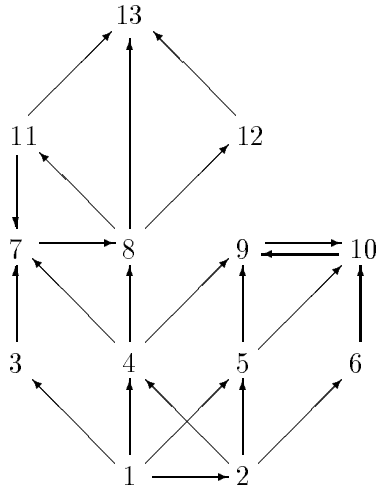
In the third step, now, we apply the functional `kernel` to the Boolean matrix `B`. The result is a $2^{13} \times 1$ vector describing the subsets of the point set which are kernels. In order to represent

the kernels as the columns of a Boolean matrix (compare Subsection 2.4), we use a built-in command and compute from the vector of kernels the injection of the set of kernels into the powerset of the point set. Multiplying `epsi` and the transpose of this injection, finally, we arrive at the following column-wise representation of the set of kernels:

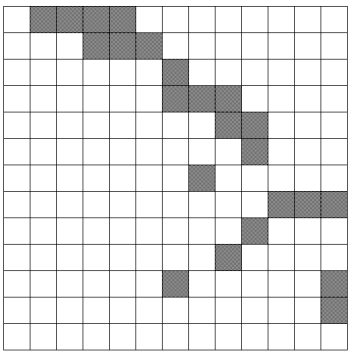


I.e., the given graph has four kernels, viz. the sets $\{1,6,7,9,13\}$, $\{1,6,7,9,12\}$, $\{1,2,7,10,13\}$, and $\{1,2,7,10,12\}$.

Hammocks: Now, we consider a second example. We delete from the graph $\mathcal{G}$ the arc $(13,12)$ and insert the arc $(1,2)$. Graphically, the result is:



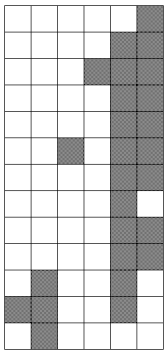
This new graph – we denote it by $\mathcal{H} = (V, C)$ – is a control flow graph since the point 1 has no predecessors, the point 13 has no successors, and each point lies on at least one path from 1 to 13. (It should be mentioned that such a test easily can be performed using RELVIEW, too.) The relation C of $\mathcal{H}$ as presented by RELVIEW looks as follows:



We want to compute the set of all hammocks of this graph. As can been seen again from Subsection 3.1, besides the 13×13 Boolean matrix `C` for the relation C of $\mathcal{H}$ we have to create three further matrices, viz. a 13×13 matrix `I` for the identity relation I on V, the 13×1 universal vector `L`, and the 13×2^{13} "is-element-of" relation `epsi` between points and pointsets. Furthermore, we have to define the following two RELVIEW functionals:

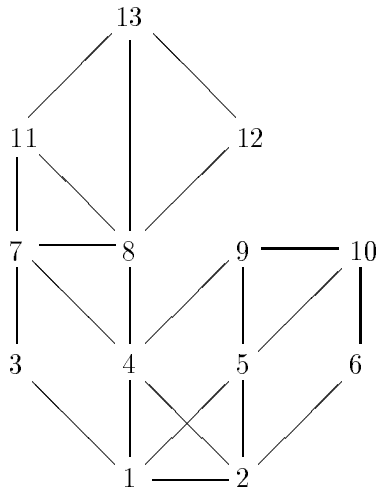
`singlesucc`	`syq(-epsi & % * epsi , I)`
`hammock`	`(singlesucc(%) & singlesucc(% ^) * -I) * L`

The rest is as in the case of kernels. First, we apply the functional `hammock` to the Boolean matrix `C` and obtain a $2^{13} \times 1$ vector describing the hammocks. Then we compute from this vector the injection into the powerset of the point set. And, finally, we multiply the matrix `epsi` by the transpose of the injection. The result is the following column-wise representation of the set of hammocks:



Besides the three singleton hammocks $\{12\}$, $\{6\}$, and $\{3\}$, we have the hammocks $\{11, 12, 13\}$ (with entry point 8 and leaving point 7), $\{2, \ldots, 12\}$ (with entry point 1 and leaving point 13), and $V \setminus \{8, 11, 12, 13\}$ (with entry point 11 and leaving point 8).

Cliques: In our last example, we consider the symmetric version $\mathcal{H}_s = (V, C \cup C^{\mathrm{T}})$ of $\mathcal{H}$, i.e., the following simple graph:

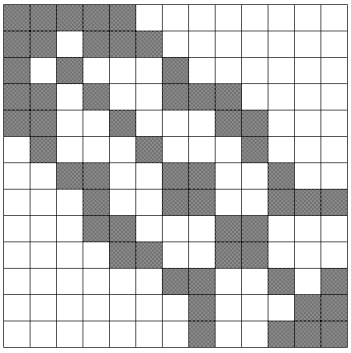


Using the RELVIEW system, we want to compute the set of all cliques of this graph. To this end, first, we have to create the 13×1 empty vector `0`, the 13×2^{13} "is-element-of" relation

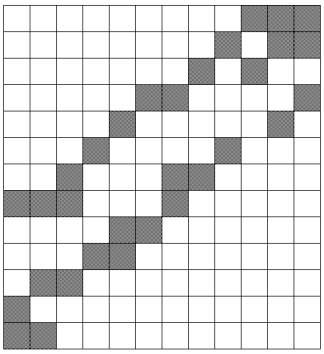
`epsi`, the $2^{13} \times 2^{13}$ inclusion relation `incl`, and the $2^{13} \times 2^{13}$ identity relation `I`. Furthermore, we have to define the following two functionals:

`max`	$\%1 \,\&\, -((\%0 \,\&\, -\,\mathtt{I}) * \%1)$
`clique`	$\mathtt{max}(\mathtt{incl}, (\mathtt{epsi} \,\&\, -\,\% * \mathtt{epsi}) \setminus \mathtt{0})$

(In the case of many-place user-declared functionals, the present version of RELVIEW uses `%0`, `%1`, etc. as variables.) Next, we apply the RELVIEW functional `clique` to the Boolean matrix



representing the reflexive closure of the relation $C \cup C^{\mathrm{T}}$. The result is a $2^{13} \times 1$ vector describing the set of all cliques of $\mathcal{H}_s$. And, finally, we multiply the matrix `epsi` by the transpose of the injection of this vector into the powerset, yielding the following column-wise representation of the cliques:



I.e., the above graph contains exactly 12 cliques. Five of them have cardinality 2; the remaining cliques have cardinality 3.

4.3 Development of Efficient Algorithms Using Abstract Relation Algebra

The execution of a specification produced in the way described in Section 3, frequently may be fairly inefficient. In this subsection we demonstrate how to develop more efficient algorithms from the original inefficient specifications using the relational calculus.

We consider again the problem of computing the kernels of a graph $\mathcal{G} = (V, B)$. In contrast with Subsection 3.1, however, in the following we do not consider sets as points $p : 2^V \leftrightarrow \mathbf{1}$ but as vectors $v : V \leftrightarrow \mathbf{1}$. In doing so, $x \in p$ will be replaced by v_x. So, the first-order formulae of Subsection 3.1 defining absorbant and stable sets become

$$\bigwedge_x \overline{a}_x \to \bigvee_y (a_y \wedge B_{xy}) \qquad\qquad \bigwedge_x s_x \to \bigwedge_y (s_y \to \overline{B}_{xy}).$$

Translating these formulae into a notation without components, we get the inclusions $\overline{a} \subset Ba$ and $s \subset \overline{Bs}$. As a consequence, a vector $k : V \leftrightarrow \mathsf{1}$ is a kernel of $\mathcal{G}$ if and only if $\overline{k} = Bk$, i.e., if and only if it is a fixpoint of the functional $\tau(v) = \overline{Bv}$.

Progressively finite graphs: The just defined functional τ is antitone, so the fixpoint theorem for monotone functions on complete lattices cannot be applied. We therefore study the fixpoints of $\tau^2(v) := \tau(\tau(v)) = \overline{B\overline{Bv}}$ which is monotone. Suppose m_{τ^2} and M_{τ^2} to denote the least resp. greatest fixpoint of τ^2. Then we have for each kernel k that

$$\mathsf{O} \subset \tau^2(\mathsf{O}) \subset \tau^4(\mathsf{O}) \subset \ldots \subset m_{\tau^2} \subset k \subset M_{\tau^2} \subset \ldots \tau^4(\mathsf{L}) \subset \tau^2(\mathsf{L}) \subset \mathsf{L}.$$

Also the two equations

$$\text{(i)} \quad \tau(m_{\tau^2}) = M_{\tau^2} \qquad\qquad \text{(ii)} \quad \tau(M_{\tau^2}) = m_{\tau^2}$$

easily can be shown. Hence, if the gap between the lower bound m_{τ^2} and the upper bound M_{τ^2} of the set of all kernel closes, then the uniquely determined fixpoint is a kernel of $\mathcal{G}$:

Theorem. If the functional τ^2 has exactly one fixpoint (which is equivalent to $M_{\tau^2} \subset \tau(M_{\tau^2})$ or to $\tau(m_{\tau^2}) \subset m_{\tau^2}$), then $\mathcal{G}$ has precisely one kernel. □

Using this theorem, for instance, it can be shown that a progressively finite graph $\mathcal{G} = (V, B)$ (i.e., a graph in which all paths have finite lengths) has exactly one kernel. When specifying progressive finiteness relationally, we obtain

$$(*) \quad \bigwedge_v (v = v\mathsf{L} \wedge v \subset Bv \rightarrow v = \mathsf{O}).$$

Now, we use the Schröder equivalences and get $B^{\mathrm{T}} M_{\tau^2} \subset B M_{\tau^2}$ from $M_{\tau^2} \subset \tau^2(M_{\tau^2})$. Next, the Dedekind rule yields

$$BM_{\tau^2} \cap M_{\tau^2} \subset (B \cap M_{\tau^2} M_{\tau^2}^{\mathrm{T}})(M_{\tau^2} \cap B^{\mathrm{T}} M_{\tau^2}) \subset B(M_{\tau^2} \cap B^{\mathrm{T}} M_{\tau^2}) \subset B(M_{\tau^2} \cap BM_{\tau^2})$$

and, finally, in combination with $(*)$ we obtain $BM_{\tau^2} \cap M_{\tau^2} = \mathsf{O}$ i.e., $M_{\tau^2} \subset \tau(M_{\tau^2})$. Hence, the functional τ has precisely one fixpoint.

For the point set being finite we have that a directed graph is progressively finite if and only if it is circuit-free. Using the RELVIEW system, therefore, we can compute the only kernel of a circuit-free graph $\mathcal{G} = (V, B)$ as follows: Let n denote the cardinality of the point set V. First, we create a $n \times n$ Boolean matrix `B` for the relation B and the $n \times 1$ empty vector `O`. Then, we define the functional

```
tau2    -(B * -(B * %0)).
```

And, finally, we employ the iteration `O, tau2(tau2(O)), ...` that approximates the kernel using the built-in iteration command.

At this place it should be mentioned that in the case of a transitive relation B the iteration stops after one step. This is due to the fact that in the case $B = B^+$ and $B = B^* \overline{B\mathsf{L}}$ (this latter condition means that from each point there is a path to a terminal one) the only kernel of $\mathcal{G}$ is the least absorbant set which equals the set $\overline{B\mathsf{L}} : V \leftrightarrow \mathsf{1}$ of all terminal points:

$$\tau^2(\mathsf{O}) = \overline{B\overline{B\mathsf{O}}} = \overline{B\mathsf{L}} = \overline{B\overline{B\mathsf{L}}} = \tau^2(\mathsf{L})$$

follows from $B\overline{B\mathsf{L}} \subset B\mathsf{L}$ and $\mathsf{L} = B^*\overline{B\mathsf{L}} = \overline{B\mathsf{L}} \cup B^+\overline{B\mathsf{L}} = \overline{B\mathsf{L}} \cup B\overline{B\mathsf{L}}$ which in turn is equivalent to $B\mathsf{L} \subset B\overline{B\mathsf{L}}$.

Bipartite graphs: Now, assume $\mathcal{G} = (X, Y, R, S)$ to be a bipartite graph in the sense of Subsection 3.5, i.e., we have $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow X$. Furthermore let $\imath : X \leftrightarrow X + Y$ and $\kappa : Y \leftrightarrow X + Y$ be the natural injections into the direct sum $X + Y$. Then

$$\mathcal{H} = (V, B), \text{ where } V := X + Y \text{ and } B := \imath^{\mathrm{T}} R \kappa \cup \kappa^{\mathrm{T}} S \imath,$$

is the "ordinary" directed graph corresponding to $\mathcal{G}$. Generalizing the above technique of the composition of the antitone functional τ with itself to pairs of antitone functionals and using the laws of the direct sum, in the following we will show that $\mathcal{H}$ has at least two kernels. To this end, we consider the functionals $\alpha(v) = \overline{Rv}$ and $\beta(w) = \overline{Sw}$ and obtain for the least and greatest fixpoints of the compositions

$$\begin{array}{llll} \text{(i)} & \alpha(M_{\beta\circ\alpha}) = m_{\alpha\circ\beta} & \text{(ii)} & \alpha(m_{\beta\circ\alpha}) = M_{\alpha\circ\beta} \\ \text{(iii)} & \beta(M_{\alpha\circ\beta}) = m_{\beta\circ\alpha} & \text{(iv)} & \beta(m_{\alpha\circ\beta}) = M_{\beta\circ\alpha}. \end{array}$$

Define $k_1 := \imath^{\mathrm{T}} M_{\alpha\circ\beta} \cup \kappa^{\mathrm{T}} m_{\beta\circ\alpha} : X + Y \leftrightarrow \mathsf{1}$ and $k_2 := \imath^{\mathrm{T}} m_{\alpha\circ\beta} \cup \kappa^{\mathrm{T}} M_{\beta\circ\alpha} : X + Y \leftrightarrow \mathsf{1}$. Then, we obtain the equations $Bk_1 = \overline{k_1}$ and $Bk_2 = \overline{k_2}$. E.g., a proof of $Bk_1 = \overline{k_1}$ proceeds as follows: We use an immediate consequence of (S_1) through (S_3), viz. $\imath^{\mathrm{T}}\mathsf{L} = \overline{\kappa^{\mathrm{T}}\mathsf{L}}$, and obtain

$$\kappa^{\mathrm{T}}\mathsf{L} \subset \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \qquad \imath^{\mathrm{T}}\mathsf{L} \subset \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}}.$$

From the axioms of the direct sum we get also that both $\imath^{\mathrm{T}}$ and κ^{T} are partial functions. Therefore, due to Proposition 4.2.2.v of [Schmidt Ströhlein 89], we have

$$\imath^{\mathrm{T}}\overline{M_{\alpha\circ\beta}} = \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cap \imath^{\mathrm{T}}\mathsf{L} \qquad \kappa^{\mathrm{T}}\overline{m_{\beta\circ\alpha}} = \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}} \cap \kappa^{\mathrm{T}}\mathsf{L}.$$

Now, we combine these properties and arrive at

$$\begin{aligned} \imath^{\mathrm{T}}\overline{M_{\alpha\circ\beta}} \cup \kappa^{\mathrm{T}}\overline{m_{\beta\circ\alpha}} &= (\overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cap \imath^{\mathrm{T}}\mathsf{L}) \cup (\overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}} \cap \kappa^{\mathrm{T}}\mathsf{L}) \\ &= (\overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cup \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}}) \cap (\overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cup \kappa^{\mathrm{T}}\mathsf{L}) \cap (\imath^{\mathrm{T}}\mathsf{L} \cup \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}}) \cap (\imath^{\mathrm{T}}\mathsf{L} \cup \kappa^{\mathrm{T}}\mathsf{L}) \\ &= (\overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cup \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}}) \cap \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cap \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}} \\ &= \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta}} \cap \overline{\kappa^{\mathrm{T}} m_{\beta\circ\alpha}}, \end{aligned}$$

which in turn implies the desired result as follows:

$$\begin{aligned} Bk_1 &= (\imath^{\mathrm{T}} R \kappa \cup \kappa^{\mathrm{T}} S \imath)(\imath^{\mathrm{T}} M_{\alpha\circ\beta} \cup \kappa^{\mathrm{T}} m_{\beta\circ\alpha}) \\ &= \imath^{\mathrm{T}} R m_{\beta\circ\alpha} \cup \kappa^{\mathrm{T}} S M_{\alpha\circ\beta} && \text{axioms of the direct sum} \\ &= \imath^{\mathrm{T}}\overline{\alpha(m_{\beta\circ\alpha})} \cup \kappa^{\mathrm{T}}\overline{\beta(M_{\alpha\circ\beta})} \\ &= \imath^{\mathrm{T}}\overline{M_{\alpha\circ\beta}} \cup \kappa^{\mathrm{T}}\overline{m_{\beta\circ\alpha}} && \text{due to (ii) and (iii)} \\ &= \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta} \cap \kappa^{\mathrm{T}} m_{\beta\circ\alpha}} \\ &= \overline{\imath^{\mathrm{T}} M_{\alpha\circ\beta} \cup \kappa^{\mathrm{T}} m_{\beta\circ\alpha}} \\ &= \overline{k_1}. \end{aligned}$$

Altogether, we have shown the following generalization of Richardson's theorem an early version of which goes back to [Ströhlein 70].

Theorem. If $\mathcal{G}$ is a bipartite graph, then the corresponding directed graph $\mathcal{H}$ has two distinguished (but not necessarily distinct) kernels. □

In RELVIEW, there exist built-in commands for producing direct sums and their injections. Thus, given R and S as Boolean matrices of dimension $m \times n$ resp. $n \times m$, it is obvious how to compute the $(m+n) \times 1$ Boolean vectors `k1` and `k2` corresponding to the two kernels k_1 and k_2. First, we introduce a direct sum, compute the corresponding natural injections, and store them as `iota` and `kappa`. Then, we define five functionals, viz.:

`alpha`	$-(\mathtt{R} * \%)$
`beta`	$-(\mathtt{S} * \%)$
`al_be`	$\mathtt{alpha}(\mathtt{beta}(\%))$
`be_al`	$\mathtt{beta}(\mathtt{alpha}(\%))$
`vecsum`	$\mathtt{jota}^\frown * \%\mathtt{0} \mid \mathtt{kappa}^\frown * \%\mathtt{1}$

In order to compute, for instance, the vector `k1`, finally, we employ the two iterations

$$\mathtt{L}, \mathtt{al_be}(\mathtt{L}), \mathtt{al_be}(\mathtt{al_be}(\mathtt{L})), \ldots \qquad \mathtt{0}, \mathtt{be_al}(\mathtt{0}), \mathtt{be_al}(\mathtt{be_al}(\mathtt{0})), \ldots$$

(where `L` is the $m \times 1$ universal vector and `0` is the $n \times 1$ empty vector) using the iteration command, store the results as `M_al_be` resp. `m_be_al`, and apply the functional `vecsum` to the stored results.

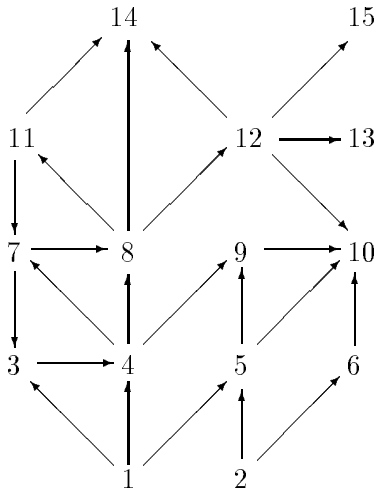
Games: The previous results on kernels will now be generalized to games. We deal with two-player games where both players make their moves alternately, and where the outcome is rated a "win", a "loss", or a "draw by infinite repetition". We do not consider games involving integer-valued or real-valued scores. Furthermore, we shall only discuss games with "complete information", where questions regarding probability play no role, in contrast to games of cards or of dice.

Let two players $\mathcal{L}, \mathcal{R}$ with two sets called their *positions* X, Y and a bipartitioned graph be given, i.e., relations $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow X$, on which they play a game as follows. At any given stage of the play one of the positions is the *current* one. In the positions on the left-hand side, say X, it is always player $\mathcal{L}$'s turn to make the next move and on the right side Y player $\mathcal{R}$ has to move. So, each position determines the player who is to make a move which in turn means that he chooses one of the arcs emanating from the current positions. The new position after making the move is then the one described by the head of the arc chosen. Because of the bipartition of the graph, the right of making a move is alternating.

The rule of loss says: "If some player is to make a move but can no longer make any (terminal positions), then he loses". One player *winning* is equivalent to the opponent *losing*.

For simplicity, in the following we consider only games with symmetric rules in which each player can follow the same instructions. I.e., we will assume $X = Y$ and $R = S$. Then the bipartitioned graph can be replaced by the "ordinary" directed graph $\mathcal{G} = (X, R)$. An impression of such a symmetric game is furnished by the following (very simple) game graph

which, obviously, has no kernel:



Now, the question arises as to whether such games can be theoretically solved using relation algebra, or – even better – be solved computationally by a system like RELVIEW. At first sight this seems not feasible, as playing the game means following a path in the graph which is determined alternately by the two players. To keep track of the set of all such paths we had to spend much space and time. The situation with games is better, though. Only three sets have to be computed in order to give the answer whether a play of the game starting from a given situation will end with loss, or win, or draw. To enable a player to achieve win thus predicted, or to avoid loss if a draw is predicted, requires in addition *sequences of sets*.

First, we try to describe the sets predicting the result. Given a position x of the game graph, and given that it is player p's turn to move from x where $p \in \{\mathcal{L}, \mathcal{R}\}$, we call x a *winning position* for p if there is a strategy for p that, against every strategy of its opponent results in a play starting in x and ending in a position where its opponent has no more move available. Furthermore, x is called a *losing position* for p if there is a strategy for its opponent that, against every strategy of p, results in a play starting in x and ending in a position in which p is to move but cannot. Finally, x is termed a *draw position* if x is neither a winning nor a losing position.

We give the following three obvious facts, keeping in mind that positions are being considered from the viewpoint of the player next to move. We omit the proof which relies on the notion of strategy and is not completely trivial.

- From a winning position there is at least one possible move into a losing position.
- From a losing position there is no move into a position of draw or a losing position.
- From a position of draw there is at least one move into some other position of draw, but no move into a losing position.

Other arcs are admitted if not explicitely excluded. We compare this to the relationship between a kernel and its complement, which is obviously the special case with no positions of draw:

- From outside the kernel there is at least one arc into the kernel.
- From a point inside the kernel there is no arc to a point inside the kernel.

This suggests modifying the kernel formulae of Subsection 3.1. Describing again sets by vectors instead of points, we arrive at

$$(1) \quad \bigwedge_x (loss \cup draw)_x \to \bigwedge_y (loss_y \to \overline{B}_{xy})$$
$$(2) \quad \bigwedge_x \overline{(loss \cup draw)}_x \to \bigvee_y (loss_y \wedge B_{xy})$$
$$(3) \quad \bigwedge_x \overline{(loss \cup win)}_x \to \bigvee_y (draw_y \wedge B_{xy})$$
$$(4) \quad \bigwedge_x loss_x \to \bigwedge_y (draw_y \to \overline{B}_{xy}).$$

We first recognize that $loss : X \leftrightarrow \mathsf{1}$ is a stable set and among those the greatest. Furthermore, the complement of $win : X \leftrightarrow \mathsf{1}$ (i.e., $loss \cup draw$) is an absorbant set. So, we iterate

$$\mathsf{O} \subset \tau^2(\mathsf{O}) \subset \tau^4(\mathsf{O}) \subset \ldots$$

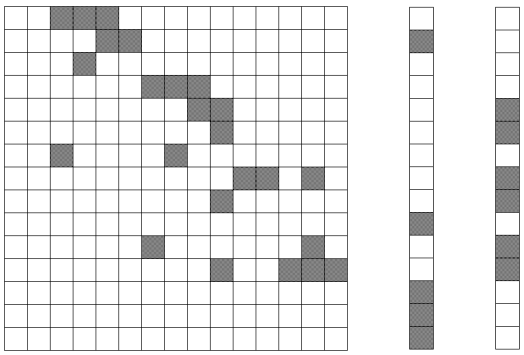
in order to obtain the greatest stable set m_{τ^2} and

$$\mathsf{L} \supset \tau^2(\mathsf{L}) \supset \tau^4(\mathsf{L}) \supset \ldots$$

in order to obtain the smallest absorbant set M_{τ^2}, where the functional τ is as above. Then we form the three vectors

$$loss := m_{\tau^2} \qquad draw := \overline{m_{\tau^2}} \cap M_{\tau^2} \qquad win := \overline{M_{\tau^2}}.$$

Since in RELVIEW there is a command for the repeated application of a functional, the system can be used to compute these vectors. In the following, we show the relation of the above game graph as well as the vectors *loss* and *win* as presented by RELVIEW:



Hence, the situations with loss for the player about to move are $\{2, 10, 13, 14, 15\}$ and the situations with win for the same player are $\{5, 6, 8, 9, 11, 12\}$. Since we also can compute the "layers" $a_0 := \mathsf{O}$, $a_1 := \tau^2(\mathsf{O})$, ... of *loss*, in a winning position there is a strategy which secures win. This strategy says: "Move into a loosing position belonging to the smallest a_i that can be reached".

5 Concluding Remarks

In this paper we have described a rapid prototyping approach for the enumeration of certain mathematical objects in terms of relations. This has lead to an extensive use of a relational

characterization of higher-order objects like sets, sets of sets, or functions. We have also shown how to perform the resulting executable relational specifications using the computer system RELVIEW and how to develop more efficient algorithms from inefficient specifications within the abstract relational framework.

Besides the experiments described in the last section, a lot of other studies have been performed with the RELVIEW system including further graph- and order-theoretic questions resp. algorithms, DAG-languages, domain constructions, relational specifications, and relational semantics. Of course, computation with RELVIEW is limited in space and time. The limit, however, depends heavily on the type of problem handled. As an example, we mention the Hasse diagram of a (finite or, more general, discrete) partially ordered set $\mathcal{O} = (M, E)$ to be computed by the functional $hasse(E) : M \leftrightarrow M$, given by

$$hasse(E) = (E \cap \overline{I}) \cap \overline{(E \cap \overline{I})(E \cap \overline{I})}.$$

On our installation (SUN SPARCstation 10), we have treated, e.g., ordered sets with up to 5 000 elements. In general, the handling of relations $R : V \leftrightarrow V$, vectors $v : V \times V \leftrightarrow 1$, or vectors $v : V \leftrightarrow 1$ is of such a complexity that admits a wide range of RELVIEW applications. However, it is much more difficult to treat the powerset of a set or function spaces. This means the computation of vectors $v : 2^V \leftrightarrow 1$ or $v : W^V \leftrightarrow 1$ and consumes a lot of space and time, since e.g., in the first case $|V| = 25$ causes $2^{25} = 33\,554\,432$ rows.

An example for a vector with domain $V \times V$ is the computation of all bridges whereas, e.g., a computation of all cut-points or all initial/terminal points leads to a vector with domain V. As we have shown in Subsection 4.3, also the computation of the only kernel of a finite circuit-free graph deals with vectors of the second kind.

Using the only two relations $\mathsf{O} : 1 \leftrightarrow 1$ and $\mathsf{L} : 1 \leftrightarrow 1$ on the singletion set 1 as the Booleans, it is even possible to test properties of relations. As an example, we consider relational inclusion $R \subset S$ which is, for $R : V \leftrightarrow W$ and $S : V \leftrightarrow W$, defined by the first-order formula

$$\bigwedge_x \bigwedge_y R_{xy} \rightarrow S_{xy}.$$

By the rules of universal quantification (see Subsection 2.3) and the propositional fact that $p \rightarrow q$ is equivalent to $\overline{p} \vee q$ we get

$$R \subset S \iff \overline{(\overline{R} \cup S) \,/\, \mathsf{L}^{\mathrm{T}}} \setminus \mathsf{O} = \mathsf{L} \iff ((R \cap \overline{S})\mathsf{L}) \setminus \mathsf{O} = \mathsf{L},$$

where $\mathsf{O} : V \leftrightarrow 1$ and $\mathsf{L} : W \leftrightarrow 1$. Therefore, we can integrate the test on inclusion by the functional $incl(R, S) : 1 \leftrightarrow 1$, which is defined as

$$incl(R, S) = ((R \cap \overline{S})\mathsf{L}) \setminus \mathsf{O}.$$

Further relational properties consisting of inclusions, such as functionality, totality, surjectivity, injectivity, transitivity, reflexivity, antisymmetry, and many others, can then easily be reformulated in terms of *incl* since, clearly, the logical connectives $\vee$, $\wedge$, and $\overline{}$ directly correspond to their relational variants. Note that this result is similar to a logical result of Schröder on simple homogeneous relation algebras, which reads as follows (see [Schröder 1895]): For every relation-algebraic formula φ, there is a relation-algebraic term φ^* such that φ is logically equivalent to the equation $\varphi^* = \mathsf{L}$.

Let us close with a few remarks on further developments on RELVIEW. It turns out that the system is a good tool for the interactive manipulation of relations. However, experience has shown that for some tasks certain additional features will be very helpful.

A main improvement is possible in the layout. The present Boolean matrix visualization of relations is well-suited for many tasks, in particular, if the intention is to get insight into an "abstract" relational problem. However, if the system is used to solve concrete problems on graphs or related structures by relational methods (as we did in this paper), then it seems better to visualize homogeneous relations as directed graphs. Therefore, for the future we plan the incorporation of commands realizing a transition between Boolean matrices and graphs. Especially, it should be possible to edit a relation as a graph. For a visualization of results, furthermore, the user should be given the option to display a relation on the screen as a directed graph and to emphasize a specific subset of the nodes described by a vector.

Besides this main extension, we plan also some minor extensions of RELVIEW. E.g., we are concerned with interfaces to other systems. The ability for producing scientific papers on relations which mix text and drawings of Boolean matrices and graphs, respectively, can be obtained by interfacing the RELVIEW system with some typesetting systems. Furthermore, an interface to the relational formula manipulation system and proof checker RALF (also developed at Bundeswehr-University Munich [Brethauer 91]) is planned.

References

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

[Backhouse et al. 92]
Backhouse R.C., Hoogendijk P., Voermans E., van der Woude J.C.S.P: A relational theory of datatypes. Eindhoven University of Technology, Dept. of Mathematics and Computer Science (1992)

[Berge 73]
Berge C.: Graphs and hypergraphs. North-Holland (1973)

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

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

[Berghammer Schmidt 93]
Berghammer R., Schmidt G.: Relational specifications. In: Rauszer C. (ed.): Algebraic Logic. Banach Center Publications, Vol. 28, Polish Academy of Sciences, to appear

[Berghammer Zierer 86]
Berghammer R., Zierer H.: Relational algebraic semantics of deterministic and nondeterministic programs. Theoret. Comput. Sci. 43, 123–147 (1986)

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

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

[Budde et al. 84]
Budde R., Kuhlenkamp K., Matthiassen H., Züllinghoven H. (eds.): Approaches to prototyping. Springer (1984)

[Chin Tarski 51]
Chin L.H., Tarski A.: Distributive and modular laws in the arithmetic of relation algebras. University of California Publications in Mathematics (new series) 1, 341–384 (1951)

[de Bakker de Roever 73]
De Bakker J.W., de Roever W.P.: A calculus for recursive program schemes. In: Nivat M. (ed.): Proc. ICALP 73, North-Holland, 167–196 (1973)

[de Roever 74]
De Roever W.P.: Recursion and parameter mechanisms: An axiomatic approach. In: Loeckx J. (ed.): Proc. ICALP 74, LNCS 14, Springer, 34-65 (1974)

[Desharnais et al. 93]
Desharnais J., Jaoua A., Mili F., Boudriga N., Mili A.: Conjungate kernels: An operator for program construction. Theoret. Comput. Sci., to appear

[Ershov 70]
Ershov A.: A multilanguage programming system oriented to languages description and universal optimization algorithms. Proc. IFIP Working Conf. on ALGOL 68 implementation, North-Holland (1970)

[Fraenkel 81]
Fraenkel A.S.: Planar kernel and Grundy with $d \leq 3, d_{out} \leq 2, d_{in} \leq 2$ are NP-complete. Disc. Appl. Math. 3, 257–262 (1981)

[Haralick 74]
Haralick R.M.: The diclique representation and decomposition of binary relations. J. ACM 21, 3, 356–366 (1974)

[Hoare He 86]
Hoare, C.A.R., He Jifeng: The weakest prespecification, Parts I&II, Fundamenta Informaticae IX, 51–84 & 217–252 (1986)

[Jónsson Tarski 52]
Jónsson B., Tarski A.: Boolean algebras with operators, Part II. Amer. J. Math. 74, 127–167 (1952)

[Mili 83]
Mili A.: A relational approach to the design of deterministic programs. Acta Informatica 20, 315–328 (1983)

[Mili et al. 87]
Mili A., Desharnais J., Mili F.: Relational heuristics for the design of deterministic programs. Acta Informatica 24, 239–276 (1987)

[Möller 91]
Möller B.: Relations as a program development language. In: Möller B. (ed.): Proc. IFIP TC2/WG2.1 Working Conference on Constructing Programs from Specifications, North-Holland, 373–397 (1991)

[Reisig 85]
Reisig W.: Petri nets – An introduction. EATCS Monographs on Theoret. Comput. Sci., Springer (1985)

[Schmidt 81]
Schmidt G.: Programs as partial graphs I: Flow equivalence and correctness. Theoret. Comput. Sci. 15, 1–25 (1981)

[Schmidt Ströhlein 85]
Schmidt G., Ströhlein T.: On kernels of graphs and solutions of games: A synopsis based on relations and fixed points. SIAM J. Alg. Disc. Meth. 6,1, 54–65 (1985)

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

[Spivey 89]
Spivey J.M.: The Z notation: A reference manual. Prentice Hall (1989)

[Schröder 1895]
Schröder E.: Algebra der Logik, 3. Band, Teubner (1895)

[Ströhlein 70]
Ströhlein T.: Untersuchungen über kombinatorische Spiele. Doctoral Thesis, Technische Universität München (1970)

[Tarski 41]
Tarski A.: On the calculus of relations. Journal of Symbolic Logic 6, 73–89 (1941)

[Veloso Haeberer 91]
Veloso P., Haeberer A.: A finitary relational algebra for classical first-order logic. Bull. of the Section on Logic of the Polish Academy of Sciences 20, 52–62 (1991)

[Veloso et al. 93]
Veloso P, Haeberer A., Baum G.: Formal program construction within an extended calculus of binary relations. J. Symbolic Comp., to appear

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