Using relation algebra for the analysis of Petri nets in a CASE tool based approach

Alexander Fronk
Software Technology
University of Dortmund
44221 Dortmund, Germany
fronk@LS10.de

Abstract

We provide the calculus of relations as a data type implemented in an object-oriented Java-library, KURE. *We demonstrate how to employ* KURE *in a CASE tool for the relation-algebraic analysis of Petri nets. Relation algebra has already been applied to condition/event nets. We extend this approach to place/transition systems in general and thereby provide a novel relation-algebraic interface to Petri net analysis.* KURE *is usable in any tool modelling with relations. In this paper we address in general how CASE tools are equipped with the calculus of relations and demonstrate in particular how reachability analysis of Petri nets benefits from this formal method.*

1. Introduction

Relations are a well-known data structure in computer science. The formal apparatus of relation algebra is well understood and mathematically well established. For its mechanization, there exists a couple of tools for the computer-aided manipulation of relations and relation-algebraic formulae, as well as for relation-algebraic theorem proving. Having a look at the abundance of literature (see e.g., the references in [15, 5]), however, it seems that in the last decades relation algebra and the respective tools have rarely been used in practical software engineering; apart from some exceptions, their employment was of purely academic nature. For instance, in the domain of programming and software development, scientists investigated semantics, data structures, and algorithm development. Topics which nowadays are of great importance in practical software engineering are, for example, the formal description of graphical design languages, or, for reengineering purposes, the detection of design patterns, the comprehension of the code structure of large object-oriented programs, and software refactoring.

We addressed the relation-algebraic detection of design patterns [11] and also contributed to the description of the syntax and semantics of a three-dimensional graphical design language [10]. But, as far as we know, formal graphical design languages (amongst them Petri nets in the widest sense) are not or scarcely treated with relation algebra.

This calculus has has already been applied in the context of condition/event nets [3]. We investigate place/transition systems in general and provide a novel approach to Petri net analysis using relation algebra. We recently implemented the calculus of relations as a data type provided in an object-oriented Java-library, KURE, usable in any tool modelling with relations. It is accessible for download under `http://cvs.informatik.uni-kiel.de/~KURE/`. KURE extracts the core functionality of the tool RELVIEW [1], a software system for calculating with relations and for relation-algebraic programming developed at the Christian-Albrechts University of Kiel, Germany. We developed KURE in a diploma thesis at the University of Dortmund in cooperation with the RELVIEW-group. The library allows to easily employ the calculus of relations in any CASE tool modelling with relations. With this benefit, these relations need no longer be created manually in the RELVIEW system; nor need they be encoded as an adjacency matrix by a proprietary tool, then imported into RELVIEW in order to calculate with these relations, and results eventually reimported into one's own tool for further use. A developer simply needs to deduce relations from the data structure used in his proprietary CASE tool and can then seamlessly use the calculus of relations therein.

We have recently been developing such a CASE tool, PetRA, based on a framework for the development of graphical editors [14]. PetRA can both model Petri nets

interactively on the screen and analyze them by relational algorithms. Results of running a relational program are drawn into the Petri net displayed on the screen. This allows the user to visually compare results and check their correctness within the modelled net, or simulate the net relation-algebraically.

Our approach is based on a relational transcription of Petri nets. As its main advantage, it deduces relational programs immediately from net theoretic formulae. Since this is done by rigorous transformation rules, these programs are correct by construction. KURE allows to execute them without further implementation-technical considerations and thus makes Petri net analysis less error-prone.

In this paper, we focus on reachability concerns in place/transition systems. In the sense of relational program development, we provide an interface to the relation-algebraic analysis of Petri nets for use in any CASE tool. With the KURE library at hand, relational algebra can fruitfully be integrated into object-oriented engineering-based software development. Thereby, both software engineering methods and tools are enriched by relation algebra and the integration of formal methods and tools applying them is pushed further.

The paper is organized as follows. Section 2 briefly provides basic knowledge about relation algebra, the RELVIEW programming language, and the KURE library. Section 3 equips the reader with both a relation-algebraic formalization and an implementation of reachability in place/transition systems. Section 4 discusses related work. The paper ends with a conclusion in Section 5.

2. Preliminaries

Readers not familiar with relation algebra are referred to [15]. We equip the reader with some relation-algebraic concepts and their notation as far as used throughout this paper (Sect. 2.1). The KURE programming language and the library are briefly described in Sect. 2.2.

2.1. Relation algebra

We assume a heterogeneous abstract relation algebra to be defined as in [15, Def. A.2.1].

A relation R between two sets X and Y is a subset of the cartesian product $X \times Y$. We denote such a (heterogeneous) relation by $R : X \leftrightarrow Y$ and write $[X \leftrightarrow Y]$ for the set of all relations over the cartesian product $X \times Y$. We always consider the sets X and Y to be non-empty and finite. Let $\#X$ and $\#Y$ be the cardinality of X and Y, resp. A relation R is represented as a Boolean matrix with $\#X$ rows and $\#Y$ columns. We write R_{xy} instead of $\langle x, y\rangle \in R$; in this case, the entry (x, y) in the matrix representing R is 1.

Given two relations R and S, $R \cup S$, $R \cap S$, and $R \subseteq S$ denote the *union*, the *intersection*, and the *inclusion* of relations R and S, respectively. We denote the *converse* of R by R^T, its *negation* by $\overline{R}$, and the *relational multiplication* by $R; S$ (or RS). The *empty relation* is denoted by $\mathbb{0}$, the *universal relation* by $\mathbb{L}$, and the *identity relation* by $\mathbb{I}$. Note that these relations are represented by Boolean matrices with 0-entries only, with 1-entries only, and with 1-entries on the diagonal only, resp. These relations are families of relations indexed with sets X and Y which we omit for better readability, since they become clear by the context.

For each relation R, the *domain* of R is defined as $R\mathbb{L}$, the *co-domain* as $R^\mathsf{T}\mathbb{L}$.

A relation R is *univalent*, if $R^\mathsf{T}R \subseteq \mathbb{I}$ holds. It is *total*, if $\mathbb{I} \subseteq RR^\mathsf{T}$ holds, and is called a *mapping*, if it is both univalent and total. For a mapping R with $R_{x,y}$ we write $R(x)$ to refer to y as usual for functions. If R is total, $R\mathbb{L} = \mathbb{L}$ holds. A relation R is injective if $RR^\mathsf{T} \subseteq \mathbb{I}$ holds, and surjective if $\mathbb{I} \subseteq R^\mathsf{T}R$ holds.

Let $\mathbb{1} := \{\diamond\}$ be a singleton set containing exactly one element. A relation $v : X \leftrightarrow \mathbb{1}$ is thus represented as a matrix consisting of exactly one column, i.e. as a *Boolean (column) vector* satisfying $v = v\mathbb{L}$. Such a vector describes the subset $\{x \in X \mid v_x\}$. We omit $\diamond$ in expressions of the form $v_{x\diamond}$ or $v_{\diamond x}$. If the domain of v equals $N \subseteq X$, v_x is equivalent to $(v; \mathbb{L})_{x,N}$ for $\mathbb{L} : \mathbb{1} \leftrightarrow 2^X$. Each relation $R : X \leftrightarrow Y$ can be transformed into a vector of type $[X \times Y \leftrightarrow \mathbb{1}]$ by calculating $(\pi_1 R \cap \pi_2)\mathbb{L}$ where π_1 and π_2 denote the respective natural projections on $X \times Y$ where $\pi_1^\mathsf{T}; \pi_2 = \mathbb{L}$ holds. Natural projections are surjective mappings. Vice versa, each vector can be transformed into a relation by calculating $\pi_1^\mathsf{T}(\pi_2 \cap v\mathbb{L})$. Vectors form a sublattice of relations.

A vector v is a *point* if it is injective and surjective. Points are the atoms in the lattice of vectors and can thus model singleton sets and therefore elements of (ordinary) sets.

A *bipartite graph* on sets X and Y is defined as a tuple $\mathcal{G} = (X, Y, R, S)$ with $X \neq \emptyset \neq Y$, $X \cap Y = \emptyset$, $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow X$. For any relation $Q : X \cup Y \leftrightarrow X \cup Y$ and a point $\nu : X \cup Y \leftrightarrow \mathbb{1}$ modelling a vertex $v \in X \cup Y$, the relations $Q\nu : X \cup Y \leftrightarrow \mathbb{1}$ and $Q^\mathsf{T}\nu : X \cup Y \leftrightarrow \mathbb{1}$ model the set of *predecessors* and *successors* of v, respectively.

There exist exactly two relations of type $[\mathbb{1} \leftrightarrow \mathbb{1}]$, namely $\mathbb{0} : \mathbb{1} \leftrightarrow \mathbb{1}$ and $\mathbb{L} : \mathbb{1} \leftrightarrow \mathbb{1}$, which are used to model the Boolean truth values *false* and *true*, resp.

Membership on sets, i.e. the relation $\in$, is modelled by a relation $\varepsilon : X \leftrightarrow 2^X$ on X and its powerset, such that $\varepsilon_{x,S} :\Longleftrightarrow\ x \in S$ for any set S.

We need residuals which model the greatest solutions of relational inclusions. For the inclusion $RQ \subseteq S$, Q is called the *right residual* of S over R, $R \backslash S$ for short, if it is the greatest relation satisfying the inclusion, and is computed by $\overline{R^\mathsf{T} \overline{S}}$. For the inclusion $QR \subseteq S$, Q is called the *left residual* of S over R, R / S for short, if it is the greatest relation satisfying the inclusion, and is computed by $\overline{\overline{S} R^\mathsf{T}}$. The right residual is left-distributive with intersection, i.e. $R \backslash (P \cap T) = (R \backslash P) \cap (R \backslash T)$. The following equations hold: $\forall x\, R_{x,y} \iff (\mathbb{L} \backslash R)_y \iff (\overline{R} \backslash \mathbb{0})_y$.

The *symmetric quotient* of two relations R and S, $\mathsf{syQ}(R,S)$ for short, is the greatest relation Q satisfying both residual inclusions. It is defined as $(R \backslash S) \cap (S^\mathsf{T} / R^\mathsf{T})$. The following equivalence holds: $\forall x\, R_{x,y} \leftrightarrow S_{x,y'} \iff \mathsf{syQ}(R,S)_{y,y'}$. For each vector $R : X \leftrightarrow \mathbb{1}$, $\mathsf{syQ}(\varepsilon, R)$ denotes the point in the powerset 2^X representing R.

R^{i+1} denotes RR^i for $i \geq 0$ with $R^0 = \mathbb{I}$. The relation $R^+ := \bigcup_{i \geq 1} R^i$ is called the *transitive closure* of R, and $R^* := \bigcup_{i \geq 0} R^i = R^+ \cup \mathbb{I}$ the *reflexive transitive closure* of R. A relation R is *acyclic* if and only if $R^+ \subseteq \overline{\mathbb{I}}$ holds.

2.2. The Kure Programming Language

In KURE, the user may manipulate and analyze relations by pre-defined operations and tests, relational functions and programs. The pre-defined basic operations of relation algebra are denoted by `|` (union), `&` (intersection), `*` (composition), `^` (transposition), and `-` (negation). Tests include, e.g., `incl`, `eq`, and `empty` for testing inclusion, equality, and emptiness of relations, respectively.

A relational function F is a clause of the form `F(`X_1`,...,`X_n`) = t.`, where `F` is the function name, the X_i, $1 \leq i \leq n$, are the formal parameters representing relations, and `t` is a relation-algebraic term over the relations accessible in the system's workspace. `t` contains the formal parameters and is terminated by a dot. A relational program in KURE is a while-program based on the data type of relations. Its structure reads as follows (keywords underlined):

```
<NAME>"("<PARAMETERLIST>")"
    DECL <DECLARATIONS>
    BEG <STATEMENTS>
        RETURN <TERM>
    END.
```

Each program starts with a head line containing the program's name and a list of formal parameters separated by colons. Then the declaration part follows which consists of the optional declaration of local variables, local relational functions, and local relational domain constructions (direct products, sums). The third part of a relational program is its body containing a sequence of statements separated by semicolons and terminated by a mandatory return-clause over a relational term. The set of statements includes a while-loop and an if-clause.

`O(R)` and `L(R)` deliver an empty relation and a universal relation with the same dimensions as R, resp.; `L1n(R)` delivers a row vector with the same column number as R, and `Ln1(R)` delivers a column vector with the same row number as `R` (`O1n(R)` and `On1(R)` analogously); `I(R)` delivers the identity relation with the same dimensions as `R`; `dom(R)` and `ran(R)` calculate the domain and the co-domain of `R` as vectors, respectively. `point(v)` selects a point included in a non-empty vector `v`. `epsi(v)` calculates a membership relation of a set into its powerset; the set is determined by the domain of the vector `v`, and its cardinality is given by the row number of `v`. `PROD(R,S)` defines the product domain over the domains of two homogeneous relations `R` and `S`, `p-1` projects into its first component, `p-2` into its second one. `trans(R)` and `refl(R)` calculate the transitive and reflexive closures of a relation `R`, resp.

We represent natural numbers through a linearly ordered set N together with an injective mapping, i.e. the successor relation `succ` : $N \leftrightarrow N$, and a point `zero` : $N \leftrightarrow \mathbb{1}$ representing the number zero. On N, the following additional axiom needs to hold: $(\mathsf{succ}^\mathsf{T})^*; \mathsf{zero} = \mathbb{L}$, i.e. each $n \in N$ can be reached from zero via the successor relation. The orderings $\leq$ and $\geq$ are constructed using `trans` and `refl` applied to `succ` by `le=refl(trans(succ))` and `ge=refl(trans(succ^))`, resp. A relational program calculating the addition-relation `add` : $N \times N \leftrightarrow N$ can be found at `http://www.informatik.uni-kiel.de/~progsys/relview/Misc` (`sub` works analogously).

KURE is an extraction of the core functionality of RELVIEW mechanizing the calculus of relations as an object-oriented JAVA-library. Therein, relations are treated as objects the entries of which can be set and queried by simple object-oriented methods. Relations are handled within a so called relation manager which reflects the system's workspace, administrates terms and programs, and allows to evaluate them. A CASE tool can simply use KURE by creating the required relations from, for example, a Petri net under consideration. The relational programs for analyzing Petri nets are derived from net theoretical formulae by rigorous transformation rules. They are then given as parameters to the respective evaluation method defined in the

relation manager. A small documented example may illustrate how KURE works:

```
RelManager rM = createRelManager("PNMan");
// initializes a relation manager named "PNMan"

Relation PN = rM.createRelation("PTNet", 2, 3);
// creates a 2×3 relation named "PTNet"

Object p1 = "p1"; // two places
Object p2 = "p2";
Object t1 = "t1"; // three transitions
Object t2 = "t2";
Object t3 = "t3";

PN.setObjects(new Object[]{p1,p2},
              new Object[]{t1,t2,t3});
// defines a flow relation from places to transitions

PN.setBit(p1, t1, true); // sets flow from p1 to t1
PN.setBit(p1, t2, true);
PN.setBit(p2, t1, true);
PN.setBit(p2, t3, true);

rM.createFunction("injective(R)=incl(R*R^,I(R*R^)).");
// defines "injective"

Relation check = rM.evaluateTerm("injective(PTNet)");
// calls "injective" on PN and returns a boolean

if (check.isTrue())
    throw new Exception("Flow relation is  injective .");
// handle result

rM.loadProgram(new URL("SelectPlaces.prog"));
// loads external relational program "SelectPlaces.prog"

Relation places = rM.evaluateTerm("SelectPlaces(PTNet)");
// calls SelectPlaces on relation PN
```

Terms are created via the `createFunction`-method and can then be evaluated after replacing formal through actual parameters. They may also be saved on disk using their name as file name. Like programs, they are then loaded by passing this name to the `loadProgram`-method. A program is used like a term and is thus evaluated by the `evaluateTerm`-method as well.

3. Petri net analysis based on relation algebra

We assume the reader to be familiar with basic Petri net notation and theory as, e.g., found in [16]. Here, we provide their relation algebraic transcription.

3.1. Basic definitions and propositions

Definition 1 *A **net** (or net graph) $\mathcal{N}$ is defined as a bipartite directed graph represented by $\mathcal{N} = (P, T, R, S)$ with $P \cap T = \emptyset$, $R : P \leftrightarrow T$, and $S : T \leftrightarrow P$.*

Let the sets P and T be *places* and *transitions*, resp., as usual. In the usual definition for Petri nets the flow relation F equals $R \cup S$.

$R_{p,t}$ means both p is in the preset $\bullet t$ of t and t is in the postset $p\bullet$ of p, and $S_{t,p}$ means both $t \in \bullet p$ and $p \in t\bullet$. In Petri net notation it is usual to have $\bullet V$ and $V\bullet$ for a subset $V \subseteq P \cup T$ defined as $\bigcup_{v \in V} \bullet v$ and $\bigcup_{v \in V} v\bullet$, resp. In terms of relational algebra, for a vector $v : P \leftrightarrow \mathbb{1}$ modelling $P' \subseteq P$ and a vector $w : T \leftrightarrow \mathbb{1}$ modelling $T' \subseteq T$, we define $\bullet P' := S; v, P'\bullet := R^\mathsf{T}; v, \bullet T' := R; w, T'\bullet := S^\mathsf{T}; w$. The same definitions apply if v and w are points and thus describe a single place and transition, resp.

A *net system*, or *P/T-system*, introduces (anonymous) token into a net graph. These token form a marking of the net.

Definition 2 *A **P/T-system** $\mathcal{N}_M$ is represented as a tuple $\mathcal{N}_M = (P, T, R, S, C, W^{\bullet t}, W^{t\bullet}, M_0)$ with*

1. *(P, T, R, S) is a net graph,*
2. *$C : P \leftrightarrow (\mathbb{N} \cup \{\infty\})$ a mapping from the set of places to the natural numbers or infinity, indicating the* capacity *of places,*
3. *$W^{\bullet t} : P \times T \leftrightarrow \mathbb{N}$ and $W^{t\bullet} : T \times P \leftrightarrow \mathbb{N}$ mappings from the set of edges in R and in S, resp., to the natural numbers, indicating the* weight *of edges,*
4. *$M_0 : P \leftrightarrow \mathbb{N}_0$ a mapping from the set of places to the natural numbers including zero, indicating an* initial marking *of places, such that $\forall p : M_0(p) \leq C(p)$ holds.*

Remark 3 *In addition to this definition, we assume that weights and capacities correlate in the following way to ensure reasonable nets: $\forall p, t : C(p) \geq W(p, t) \wedge C(p) \geq W(t, p)$. That is, the weight of an edge is never greater than the capacities of its incident places.*

3.2. Reachability in place/transition systems

Reachability studies the properties of a reflexive and transitive relation $reach : [P \leftrightarrow N] \leftrightarrow [P \leftrightarrow N]$ over markings where $reach_{MN}$ holds if a marking N is reachable from a marking M. It is for example interesting to know which markings are reachable from a given one, whether a specific marking is reachable at all or whether it can never be reached, or if *reach* contains cycles. We address these questions in the remainder of this section. The set of all markings of a net $\mathcal{N}$ is denoted by $[M_\mathcal{N}>$ as usual.

3.2.1. Concession

Transitions need to be activated under a marking in order to fire. They are then said to have concession.

Definition 4 *A transition t has **concession** under a marking M if the following conditions hold:*

1. $\forall p \in {}^\bullet t : M(p) \geq W^{\bullet t}(p,t)$
2. $\forall p \in t^\bullet : M(p) \leq C(p) - W^{t\bullet}(t,p)$

We construct the set of transitions activated under a given marking as a vector over the set T.

Proposition 5 *The set of all transitions of a net $\mathcal{N}$ with concession under a marking $M \in [M_\mathcal{N}>$ is given by the following vector $CN(\mathcal{N})$ of type $[T \leftrightarrow \mathbb{1}]$:*

$$\mathbb{L}\backslash\,(\,(\overline{R} \cup (M;\geq;\, W^{\bullet t\,\mathsf{T}} \cap \pi^\mathsf{T});\rho) \\ \cap\,(\overline{S^\mathsf{T}} \cup ((M;\leq;\mathsf{sub}^\mathsf{T} \cap C;\sigma^\mathsf{T});\tau;\, W^{t\bullet\,\mathsf{T}} \cap \beta^\mathsf{T});\alpha))$$

Remark 6 *The proof requires the natural projections $\pi : P \times T \leftrightarrow P$, $\rho : P \times T \leftrightarrow T$, $\alpha : T \times P \leftrightarrow T$, $\beta : T \times P \leftrightarrow P$, $\sigma : N \times N \leftrightarrow N$, and $\tau : N \times N \leftrightarrow N$.*

PROOF. The first condition in Def. 4 is equivalent to $\forall p\, R_{p,t} \rightarrow M(p) \geq W^{\bullet t}(p,t)$, the second one is equivalent to $\forall p\, S_{t,p} \rightarrow M(p) \leq C(p) - W^{t\bullet}(t,p)$. Hence:

$$\begin{aligned}
&\forall p\, R_{p,t} \rightarrow M(p) \geq W^{\bullet t}(p,t) \\
&\iff \forall p\, R_{p,t} \rightarrow \geq_{M(p),W^{\bullet t}(p,t)} \\
&\iff \forall p\, R_{p,t} \rightarrow \exists n\, M_{p,n} \wedge \geq_{n,W^{\bullet t}(p,t)} \\
&\iff \forall p\, R_{p,t} \rightarrow (M;\geq)_{p,W^{\bullet t}(p,t)} \\
&\iff \forall p\, R_{p,t} \rightarrow \exists m\, {W^{\bullet t}}_{<p,t>,m} \wedge (M;\geq)_{p,m} \\
&\iff \forall p\, R_{p,t} \rightarrow (M;\geq;\, {W^{\bullet t}}^\mathsf{T})_{p,<p,t>} \\
&\iff \forall p\, R_{p,t} \rightarrow \exists q\, (M;\geq;\, {W^{\bullet t}}^\mathsf{T})_{p,q} \wedge \pi_{q,p} \wedge \rho_{q,t} \\
&\iff \forall p\, R_{p,t} \rightarrow \exists q\, (M;\geq;\, {W^{\bullet t}}^\mathsf{T} \cap \pi^\mathsf{T})_{p,q} \wedge \rho_{p,t} \\
&\iff \forall p\, R_{p,t} \rightarrow ((M;\geq;\, {W^{\bullet t}}^\mathsf{T} \cap \pi^\mathsf{T});\rho)_{p,t} \\
&\iff \forall p\, \overline{R}_{p,t} \vee ((M;\geq;\, {W^{\bullet t}}^\mathsf{T} \cap \pi^\mathsf{T});\rho)_{p,t} \\
&\iff \forall p\, (\overline{R} \cup (M;\geq;\, {W^{\bullet t}}^\mathsf{T} \cap \pi^\mathsf{T});\rho)_{p,t} \\
&\iff (\mathbb{L}\backslash\,(\overline{R} \cup (M;\geq;\, {W^{\bullet t}}^\mathsf{T} \cap \pi^\mathsf{T});\rho))_t
\end{aligned}$$

and

$$\forall p\, S_{t,p} \rightarrow M(p) \leq C(p) - W^{t\bullet}(t,p) \iff \\ (\mathbb{L}\backslash\,(\overline{S^\mathsf{T}} \cup ((M;\leq;\mathsf{sub}^\mathsf{T} \cap C;\sigma^\mathsf{T});\tau;\, W^{t\bullet\,\mathsf{T}} \cap \beta^\mathsf{T});\alpha))_t$$

is proven analogously. Put together, we characterize the set of all transitions with concession under M through

$$(\mathbb{L}\backslash\,(\overline{R} \cup (M;\geq;\, W^{\bullet t\,\mathsf{T}} \cap \pi^\mathsf{T});\rho)) \\ \cap(\mathbb{L}\backslash\,(\overline{S^\mathsf{T}} \cup ((M;\leq;\mathsf{sub}^\mathsf{T} \cap C;\sigma^\mathsf{T});\tau;\, W^{t\bullet\,\mathsf{T}} \cap \beta^\mathsf{T});\alpha))$$

which yields the following component-free specification $CN(\mathcal{N})$ of type $[T \leftrightarrow \mathbb{1}]$:

$$\mathbb{L}\backslash\,(\,(\overline{R} \cup (M;\geq;\, W^{\bullet t\,\mathsf{T}} \cap \pi^\mathsf{T});\rho) \\ \cap\,(\overline{S^\mathsf{T}} \cup ((M;\leq;\mathsf{sub}^\mathsf{T} \cap C;\sigma^\mathsf{T});\tau;\, W^{t\bullet\,\mathsf{T}} \cap \beta^\mathsf{T});\alpha))$$

□

Calculating $CN(\mathcal{N})$ for a given marking is done by the following KURE program `CN` parameterized with the relations `R` and `S`, the capacity relation `C`,the weight relations `W_t` : $P \leftrightarrow T$ and `Wt_` : $T \leftrightarrow P$, and a marking `M`. The projections are introduced in the declaration part by defining the product domains $P \times T$, $T \times P$, and $N \times N$. `CN` returns a vector modelling the desired transitions. It corresponds immediately to Prop. 5. For efficiency reasons, we use $\overline{Q}\backslash\,\mathbb{0}$ instead of $\mathbb{L}\backslash\,Q$ in the return-clause, with Q calculated in `hlp`.

```
CN(R, S, C, W_t, Wt_, M)
    DECL PTdom = PROD(R*S, S*R);
         TPdom = PROD(S*R, R*S);
         NNdom = PROD(M^*M, M^*M);
         pi, rho, alpha, beta, sigma, tau, hlp
    BEG
         pi = p-1(PTdom);
         rho = p-2(PTdom);
         alpha = p-1(TPdom);
         beta = p-2(TPdom);
         sigma = p-1(NNdom);
         tau = p-2(NNdom);
         hlp = (-R | (M*ge*W_t^ & pi^)*rho)
                 &
               (-S^ | ((M*le*sub^ & C*sigma^)
                         *tau*Wt_^ & beta^) *alpha)
         RETURN -hlp \ On1(R)
    END.
```

3.2.2. Immediately reachable markings

For technical reasons, we define a domain restriction.

Definition 7 *Let $R : X \leftrightarrow Y$ be a relation and $S : X \leftrightarrow \mathbb{1}$ a vector modelling a subset of X. The **domain restriction of** R **by** S, $R|_S : X \leftrightarrow Y$ for short, is defined as $S;\mathbb{L} \cap R$.*

Remark 8 *1. $R|_S$ contains the pair (x,y) if and only if (x,y) is in R and x is in S:*

$$\begin{aligned}(S;\mathbb{L} \cap R)_{x,y} &\iff (S;\mathbb{L})_{x,y} \wedge R_{x,y} \\ &\iff S_x \wedge R_{x,y}\end{aligned}$$

2. The domain restriction is realized as a relational function `DOMRES(R, S) = S*L1n(R) & R`.

Firing a transition t with concession under a marking M results in a new marking M', $M[t > M'$ for short, which is called *immediately reachable from M*. The usual firing rule for P/T-systems reads as follows:

Definition 9 *Let t be a transition with concession under a marking M. Firing t results in an* ***immediately reachable marking*** *M' which is defined as follows:*

$$M'(p) := \begin{cases} M(p) - W^{\bullet t}(p,t), & \text{if } p \in {\bullet t} \setminus t{\bullet} \\ M(p) + W^{t\bullet}(t,p), & \text{if } p \in t{\bullet} \setminus {\bullet t} \\ M(p) - W^{\bullet t}(p,t) + W^{t\bullet}(t,p), & \text{if } p \in {\bullet t} \cap t{\bullet} \\ M(p), & \text{if } p \notin {\bullet t} \cup t{\bullet} \end{cases}$$

Immediately reachable markings form a relation $iReach : [P \leftrightarrow N] \leftrightarrow [P \leftrightarrow N]$ such that $iReach^t_{MM'}$ holds for each transition t with concession under M if $M'(p)$ evolves from $M(p)$ by the firing rule. This requires to describe the "update" of a pair (p,n) to (p,n') with $n' = M'(p)$ as in Def. 9. Therefore, we follow a constructive approach and define $iReach^t_{M,M'}$ to hold if and only if M' is derived from M in the following way:

Algorithm 10 *For each transition t modelled as a point $t : T \leftrightarrow \mathbb{1}$ with concession under M, i.e. for each $t \in CN(\mathcal{N})$ (see Prop. 5) do the following:*

1. *Build a relation $\mathsf{update}^t : P \leftrightarrow N$ such that for each pair $(p,n) \in \mathsf{update}^t$ one of the following conditions holds:*
   (a) *$p \in {\bullet t} \setminus t{\bullet}$ and $n = M(p) - W^{\bullet t}(p,t)$,*
   (b) *$p \in t{\bullet} \setminus {\bullet t}$ and $n = M(p) + W^{t\bullet}(t,p)$,*
   (c) *$p \in {\bullet t} \cap t{\bullet}$ and $n = M(p) - W^{\bullet t}(p,t) + W^{t\bullet}(t,p)$.*

   update^t models the first three cases of the firing rule.
2. *Build a relation $N \subset M$ such that N only contains pairs $(p,n) \in M$ with $p \notin {\bullet t} \cup t{\bullet}$, i.e. N models the fourth case of the firing rule.*
3. *Let M' be defined as $\mathsf{update}^t \cup N, t \in CN(\mathcal{N})$. Then, M' is the marking immediately reachable from M by firing t.*

The algorithm translates the firing rule. Step 1 performs the update of $M(p)$ taking the weights of arcs incident to p into account. Step 2 extracts all places the markings of which do not change by firing t, and step 3 conjoins the first two steps. Next, we formulate these steps relation-algebraically.

Remark 11 *Since we treat transitions as points, it is technically mandatory to rewrite the relation $W^{\bullet t} : P \times T \leftrightarrow N$ as $w^{\bullet t} : P \leftrightarrow N$ ($W^{t\bullet} : T \times P \leftrightarrow N$ analogously). It is easy to see that these relations are defined through $(R; \rho^\mathsf{T} \cap \pi^\mathsf{T}); W^{\bullet t}$ and $(S^\mathsf{T}; \alpha^\mathsf{T} \cap \beta^\mathsf{T}); W^{t\bullet}$, resp., with π, ρ, α, and β as in Rem. 6.*

Proposition 12 1. *Case* (1a) *corresponds to the domain restriction of the relation $(M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T}); \mathsf{sub}$ by $R; t \cap \overline{S^\mathsf{T}; t}$.*
2. *Case* (1b) *corresponds to the domain restriction of the relation $(M; \sigma^\mathsf{T} \cap w^{t\bullet}; \tau^\mathsf{T}); \mathsf{add}$ by $S^\mathsf{T}; t \cap \overline{R; t}$.*
3. *Case* (1c) *corresponds to the domain restriction of the relation $((M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T}); \mathsf{sub}; \sigma^\mathsf{T} \cap w^{t\bullet}; \tau^\mathsf{T}); \mathsf{add}$ by $R; t \cap S^\mathsf{T}; t$.*
4. *Case* (2) *corresponds to the domain restriction of M by $\overline{R; t \cup S^\mathsf{T}; t}$ implemented as a function* `UNCHANGED(R, S, t, M) = DOMRES(M, -(R*t | S^*t))`.
5. *With $M' := \mathsf{update}^\mathsf{t} \cup M|_{\overline{R;t \cup S^\mathsf{T};t}}$, $iReach^t$ is univalent and $iReach^t(M)$ defines the immediately reachable marking from M by firing a transition t with concession under M, i.e. $iReach^t(M)$ models $M[t> M'$.*

PROOF. For (1) we get:

$$\begin{aligned}
& p \in {\bullet t} \setminus t{\bullet} \text{ and } n = M(p) - w^{\bullet t}(p) \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge \mathsf{sub}_{<M(p), w^{\bullet t}(p)>, n} \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge \exists q\, \mathsf{sub}_{q,n} \wedge \sigma_{q,M(p)} \wedge \tau_{q,w^{\bullet t}(p)} \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge \exists q\, \mathsf{sub}_{q,n} \wedge \exists m\, M_{p,m} \wedge \sigma_{q,m} \\
& \qquad \wedge \exists n\, {w^{\bullet t}}_{p,n} \wedge \tau_{q,n} \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge \exists q\, \mathsf{sub}_{q,n} \wedge (M; \sigma^\mathsf{T})_{p,q} \\
& \qquad \wedge (w^{\bullet t}; \tau^\mathsf{T})_{p,q} \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge \exists q\, \mathsf{sub}_{q,n} \wedge (M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T})_{p,q} \\
\iff & (R; t \cap \overline{S^\mathsf{T}; t})_p \wedge ((M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T}); \mathsf{sub})_{p,n} \\
\iff & ((R; t \cap \overline{S^\mathsf{T}; t}); \mathbb{L})_{p,n} \wedge ((M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T}); \mathsf{sub})_{p,n} \\
\iff & ((R; t \cap \overline{S^\mathsf{T}; t}); \mathbb{L} \cap ((M; \sigma^\mathsf{T} \cap w^{\bullet t}; \tau^\mathsf{T}); \mathsf{sub}))_{p,n}
\end{aligned}$$

(2) and (3) are shown analogously. The correspondence in (4) is easy to follow since $p \notin {\bullet t} \cup t{\bullet}$ equals $\overline{R; t \cup S^\mathsf{T}; t}$. For (5) it remains to show that M' is a marking, i.e. a total and univalent relation of type $[P \leftrightarrow N]$ with $M'(p) \leq C(p)$ for all $p \in P$, which is trivially given by construction. □

The marking M' is obtained by the following relational program `IRM` which is a syntactical translation of the results given in Prop. 12 into a KURE program, and which conjoins the steps of Alg. 10 in the return-clause. It uses the functions `w_t`, `wt_` , and `UNCHANGED` as defined above.

```
UNCHANGED(R, S, t, M) = DOMRES(M, -(R*t | S^*t)).

IRM(R, S, W_t, Wt_, M, t)
DECL NNdom = PROD(M^*M, M^*M);
     PTdom = PROD(R*S, S*R);
     TPdom = PROD(S*R, R*S);
     w_t, wt_,
```

```
     pi, rho, alpha, beta, sigma , tau,
     updt_, upd_t, upd_t_, hlp
BEG
     sigma = p-1(NNdom);
     tau = p-2(NNdom);
     pi = p-1(PTdom);
     rho = p-2(PTdom);
     alpha = p-1(TPdom);
     beta = p-2(TPdom);
     w_t = (R*rho^ & pi^)*W_t;
     wt_ = (S^*alpha^ & beta^)*Wt_;
     upd_t =
        DOMRES((M*sigma^ & w_t*tau^)*sub, R*t &-(S^*t));
     updt_ =
        DOMRES((M*sigma^ & wt_*tau^)*add, S^*t & -(R*t));
     hlp =
        ((M*sigma^ & w_t*tau^)*sub*sigma^ & wt_*tau^)*add;
     upd_t_ = DOMRES(hlp, R*t & S^*t)
     RETURN updt_ | upd_t | upd_t_
          | UNCHANGED(R, S, t, M)
END.
```

Remark 13 *The programs* `CN` *and* `IRM` *are used to simulate a net system:* `CN` *delivers the set of all activated transitions* t *such that* t *passed to* `IRM` *delivers* $M[t>M'$.

For each net $\mathcal{N}$ and each marking $M \in [M_{\mathcal{N}}>$, the relation of immediately reachable markings, $iReach : [P \leftrightarrow N] \leftrightarrow [P \leftrightarrow N]$, is now refinable as follows:

$$iReach_{M,M'} :\Longleftrightarrow \exists t \in CN(\mathcal{N}) : M' = iReach^t(M)$$

The following implementation of *iReach* constructs the set of all pairs of immediately reachable markings depth-first (see `IREACH` and `IRMARKS` below).

We create an empty relation `allMarks` : $2^{P\times N} \leftrightarrow 2^{P\times N}$ (line 6). Therefore, let $\tilde{m} : P \times N \leftrightarrow \mathbb{1}$ with $\tilde{m} := (\pi_1 M \cap \pi_2)\mathbb{1}$ denote the vector representing M. Together with $\varepsilon : P \times N \leftrightarrow 2^{P\times N}$, the membership relation of markings in their powerset, the relation $\mathsf{syQ}(\varepsilon, \tilde{m}) : 2^{P\times N} \leftrightarrow \mathbb{1}$ models the point in the powerset of markings which represents M (lines 4 and 5). The relation $iReach^{\mathsf{T}}; \mathsf{syQ}(\varepsilon, \tilde{m})$ then models all markings immediately reachable from M (cf. [3]). `IREACH` calls `IRMARKS` (line 7) parameterized with the marking `M`, its corresponding point in $2^{P\times N}$ `mpoint` for efficiency reasons, and the empty relation `allMarks`. This program recursively calculates the relation of pairwise reachable markings as follows. Each time it is called, the marking passed is marked as reached and thus recorded in `allMarks` as an entry in its diagonal (line 13) (saying that each marking is immediately reachable from itself). We calculate the set of activated transitions, `cnc` (line 14), and stepwise determine for each point `t` in `cnc` (lines 15, 16, and 24) the marking `irm` immediately reachable from `M` under `t` (line 17). A corresponding entry is set in `allMarks` to express that `irm` is immediately reachable from `M` (lines 18 to 20). If `irm` has not yet been visited, i.e. if its corresponding point is not included in the diagonal of `allMarks` (line 21), we recursively call `IRMARKS` for the marking `irm` (line 22).

```
1 IREACH(R, S, C, W_t, Wt_, M)
2 DECL mvec, mpoint, allMarks
3 BEG
4      mvec = RelToVec(M);
5      mpoint = syq(epsi(mvec), mvec);
6      allMarks = 0(mpoint * mpoint^)
7      RETURN IRMARKS(M, mpoint, allMarks)
8 END.
9
10 IRMARKS(M, mpoint, AM)
11 DECL allMarks, cnc, t, irm, irmvec, irmpoint
12 BEG
13     allMarks = AM | mpoint * mpoint^;
14     cnc = CN(R, S, C, W_t, Wt_, M);
15     WHILE -empty(cnc) DO
16        t = point(cnc);
17        irm = IRM(R, S, W_t, Wt_, M, t);
18        irmvec = RelToVec(irm);
19        irmpoint = syq(epsi(irmvec), irmvec);
20        allMarks = allMarks | mpoint * irmpoint^;
21        IF -incl(irmpoint * irmpoint^, allMarks) THEN
22           allMarks = IRMARKS(irm, irmpoint, allMarks)
23        FI;
24        cnc = cnc & -t
25     OD
26     RETURN allMarks
27 END.
```

iReach is reflexive by construction (line 13). Markings not reachable in $\mathcal{N}$ do not occur in *iReach*. The domain of *iReach* thus provides the set of all markings reachable from M. To calculate the entire reachability relation, *reach*, it is thus sufficient to determine the transitive closure of *iReach*.

3.2.3. Reachable Markings

The transitive closure of *iReach*,

$$reach(\mathcal{N}) := iReach(\mathcal{N})^+ = \bigcup_{i\geq 1} iReach(\mathcal{N})^i\,,$$

is calculated using fixedpoint theory as usual. The set of all relations over markings, $[2^{P\times N} \leftrightarrow 2^{P\times N}]$, together with meet and join forms a complete lattice by definition of a relation algebra. The transitive closure of *iReach* is then the least fixedpoint μ of the monotonous upward continuous function $f(X) := iReach; X$ which can iteratively be computed by $a_{i+1} := f(a_i), i \geq 0$ with $a_0 := iReach$ such that a_∞ is a lower bound for μ_f (cf. [15, App. A.3]). This construction results in the following algorithm for *reach* regarding f as defined above:

```
result ← 0
iterator ← iReach
while result ≠ iterator do
     result ← iterator
     iterator ← iReach; iterator
return result
```

KURE uses this approach to calculate R^+ and predefines a function `trans` such that `trans(IREACH(R, S,`

`C, W_t, Wt_, M))` delivers the reachability relation of a net system $\mathcal{N}_M$ with initial marking `M`.

3.3. Benefits for Reachability Analysis

Before we discuss a small example applying these algorithms in the next section, we sum up the most significant aspects of our approach. The reachability problem is generally NP-complete, and polynomial for some specific net classes [7]. Our approach works for arbitrary predicate/transition nets. It is exponential, since in `IRMARKS` each marking needs to be checked for inclusion in `allMarks`. As soon as the reachability graph is constructed, however, each test can be deduced from a net theoretic formula and implemented in KURE without further considerations. With this benefit, problems which are of particular interest in reachability analysis, e.g. finding a pair of markings such that two places are marked simultaneously, or finding home states, i.e. markings that are reachable form every reachable marking, can be treated in analogy to the following reachability tests (cf. [3]).

The set of all markings reachable from M, $reachable(\mathcal{N}, M)$ for short, is the set of successors of M modelled as a vector of type $[2^{P\times N} \leftrightarrow \mathbb{1}]$ with respect to the reachability relation *reach*:

$$reachable(\mathcal{N}, \tilde{m}) := reach(\mathcal{N})^{\mathsf{T}}; \mathsf{syQ}(\varepsilon, \tilde{m})\,.$$

The program `REACHABLE` returns this set as a vector over $[2^{P\times N} \leftrightarrow \mathbb{1}]$:

```
REACHABLE(R, S, C, W_t, Wt_, M)
DECL mvec
BEG
     mvec = RelToVec(M)
     RETURN trans(IREACH(R, S, C, W_t, Wt_, M))^
            * syq(epsi(mvec), mvec)
END.
```

To test whether a marking N is reachable from M, we check whether the pair (M, N) is contained in $reach(\mathcal{N})$. Let $\tilde{n}$ and $\tilde{m}$ be their vector representations, resp. Then,

$$\mathsf{syQ}(\varepsilon, \tilde{m}); \mathsf{syQ}(\varepsilon, \tilde{n})^{\mathsf{T}} \subseteq reach(\mathcal{N})$$

trivially delivers the desired test and is implemented as the following relational program:

```
isREACHABLE(R, S, C, Wt_, W_t, M, N)
DECL mvec, nvec
BEG
     mvec = RelToVec(M);
     nvec = RelToVec(N)
     RETURN incl(syq(epsi(mvec), mvec)
                 * syq(epsi(nvec), nvec)^,
                 trans(IREACH(R, S, C, W_t, Wt, M)))
END.
```

Testing whether *reach* does not contain cycles, i.e. no marking can be reproduced, is straightforward by the usual relation algebraic formula $R^+ \subseteq \overline{\mathbb{I}}$ the implementation of which reads as `incl(trans(IREACH(R, S, C, W_t, Wt_, M)), -I)`. The markings lying on a cycle are given by $(R^+ \cap \mathbb{I}); \mathbb{L}$. Relational tests thus allow to check a property and may simultaneously deliver the candidates that fail to pass the test.

3.4. A small example

We model a crossroad with traffic-light regulation as a Petri net by means of mutual exclusion and analyze its behavior by reachable markings. The crossing consists of four lanes, one from north to south (NS), one from south to north (SN), one from east to west (EW), and one from west to east (WE), and may only be passed by cars on opposite lanes simultaneously. For simplicity, we neglect left- and right-turn vehicles. Each lane is modelled as a simple process which uses tokens as cars and requests the crossing center as resource (place sem). We model this situation as usual with the Petri net shown in Fig. 1. To ensure that only opposite cars may pass the crossing, sem is incident to two outgoing arcs with weight 2 (all other flows have weight 1). Of course, this crossroad is not fair but this is not our concern here. We extended the usual semaphore model by a second pair of concurrent processes to increase the number of reachable markings combinatorially and thus to aggravate the calculation of *reach*.

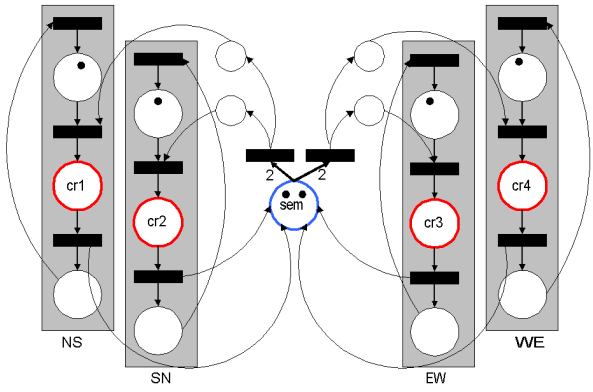

Figure 1. A Petri net modelling a crossroad

We modified the program `IREACH` in such a way we could check for each marking M calculated during the execution of `IREACH` whether the predicate $E := (P \; xor \; Q \vee \neg(P \vee Q))$ holds with $P \iff (M(cr1) + M(cr2) \geq 1)$ and $Q \iff (M(cr3) + M(cr4) \geq 1)$.

That is, either NS or SN are in their respective critical section and neither EW nor WE are (and vice versa), or no process is in its critical section. To perform this test, we simply need to develop its relation-algebraic correspondence and use it in a KURE program without further implementation-technical considerations. That is, no other data structures than relations are need to analyze the crossroad. This makes the development of programs for Petri net analysis less error-prone and provides a development process based on sheer mathematical considerations.

The relation-algebraic correspondence of P is $(\sigma; M^\mathsf{T}; NS \cap \tau; M^\mathsf{T}; NS \cap add; \geq; succ^\mathsf{T}; zero)^\mathsf{T}; \mathbb{L}$ where the relation $NS : S \leftrightarrow \mathbb{1}$ models the set $\{cr_1, cr_2\}$ (Q analogously for $\{cr_3, cr_4\}$). The expression E delivers a Boolean value. We did not find a violation of the above predicate. Hence, the crossroad is safe. It is also 2-bounded, i.e. there is no marking $M' \in [M_\mathcal{N} >$ with $M'(s) > 2, s \in S$. Again, we simply transform this net theoretic formula into a relation-algebraic term and use it in a KURE program. The respective relation-algebraic test reads as $M; \leq; succ^\mathsf{T}; succ^\mathsf{T}; zero = \mathbb{L}$.

Using a maximum capacity of 10 token for each place, the relation `allMarks` as established in `IREACH` is of size $1.96E56 \times 1.96E56$, since for 17 places and a capacity of 10 token, the powerset $2^{P\times(N+1)}$ contains 2^{187} elements. The above net consists of 184 different markings. `allMarks` contains 704 pairs of immediately reachable markings, and *reach* contains 33856 pairs. On a Pentium 4 with 2.66 GHz and 512 MB RAM, it took about 1.5 minutes to calculate this relation by a tuned version of `IREACH`. It took 0.06 seconds for only two lanes (transforming a marking into a point is costly), only 0.0009 seconds to check the above safety and boundedness properties for a given marking, and 5.0 and 1.7 seconds to check them for all 184 markings, resp.; testing whether a marking is reachable and determining the set of all markings reachable from a given one is both done in approximately 0.002 seconds. As soon as the entire reachability graph is calculated, checking its properties performs well.

4. Related work

Reachability analysis is usually based on linear programming and considers the incidence matrix C of a Petri net. A marking N is reachable from M if the marking equation $C * x = N - M$ has a solution for x. This equation is not sufficient to characterize the set of all reachable markings such that other techniques to construct the reachability graph need to be considered. Our approach both constructs the entire reachability graph and allows to analyze it by simple and efficient relational constructions. No other data structures than relations are needed. With the embedding into an imperative programming language, developing algorithms for Petri net analysis is both easy and close to net theory since net theoretic formulae can be transformed into relation-algebraic expressions by rigorous transformation rules. Our programs are thus correct by construction.

In [8], a graph theoretic algorithm is provided for testing reachability of partial markings in 1-safe nets. It is based on an unfolding technique and corresponds to the clique problem. The algorithm is based on imperative concepts and does not respect the advantages of relational programming. The clique problem is efficiently solved with relation algebra [2]. It might be interesting to capture the unfolding technique relation-algebraically, which is furthermore shown to work for LTL model checking [6]. Relation algebra is based on set theory and boolean logic; model checking based on temporal logics is feasible with relation algebra (see e.g. [13]) but is not a concern in our approach.

In [3], condition/event nets are investigated with relation algebra. On the static side, relational programs are developed that test whether a net is a free choice net, a synchronization graph, or a state machine. It is shown how deadlocks and traps can be located. On the dynamic side, reachability, liveness, concurrency, conflicts, and contact are discussed. These results, however, require further considerations to be carried over to general place/transition systems.

In [12], relation-algebraic graph traversal algorithms are developed. They are considered for homogeneous graphs but not for bipartite ones and hence exclude Petri nets.

The tool GROK [9] has been tailored for the use of relation algebra in software architecture and maintenance and uses a very simple data structure for the representation of relations. If non-relational constructs like loops are frequently needed within a GROK-program, manipulations are executed slowly in the case of very large relations which easily occur in the Petri net context. In order to make KURE work on large data – a significant feature for Petri net analysis – it uses an efficient implementation of relations based on reduced ordered binary decision diagrams (ROBDDs) [4].

The tool CROCOPAT was recently equipped with a programming language based on ROBDDs. Although the idea seems to be taken from RELVIEW, it is not clear to us whether these programs can be developed by rigorous transformation rules, and whether they are thus correct by construction. As far as we know, there does not exist a library usable within other CASE tools.

5. Conclusion

We proposed a novel relation-algebraic interface to reachability analysis of Petri nets. Based on an object-oriented JAVA-library, KURE, we realized a mechanization for the calculus of relations such that it can be used in any CASE tool modelling with relations. We demonstrated both the benefit of relation algebra in Petri net analysis and the employment of KURE. We use it in our own CASE tool, PetRA. KURE allows to combine relation algebra with arbitrary visualizations and thereby offers a flexible way to graphically create relations and illustrate the results a relational program delivers.

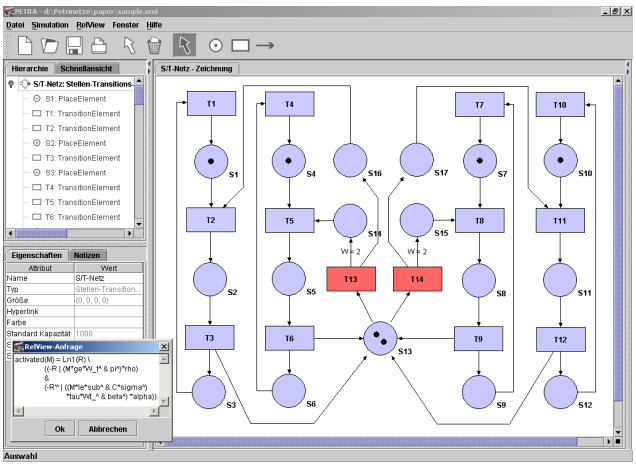

Figure 2. The Petri net CASE tool PetRA

We are planning to re-implementing the RELVIEW system as a plug-in for the development environment Eclipse using the KURE library. Thereby, we offer professional support in relational program development. Furthermore, other tools used in practice profit from integrating KURE into the Eclipse environment: With KURE, the calculus of relations and tools for practical purposes can easily be joined such that both software engineering methods and tools are enriched by relation algebra. This contributes to the integration of formal methods and the tools applying them.

Acknowledgements. Thanks to O. Szymanski, J. Pleumann, E.-E. Doberkat, R. Berghammer, and two anonymous referees for providing very detailed and useful information.

References

[1] R. Behnke, R. Berghammer, E. Meyer, and P. Schneider. RELVIEW – A system for calculating with relations and relational programming. In *Proc. 1st Intl. Conf. on Fundamental Approaches to Software Engineering*, volume 1382 of *LNCS*, pages 318 – 321. Springer, 1998.

[2] R. Berghammer. The clique problem. Private communication; paper in progress, 2004.

[3] R. Berghammer, B. Karger, and C. Ulke. Relational-algebraic analysis of Petri nets with RELVIEW. In *Proc. 2nd Workshop on Tools and Applications for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 49–69. Springer, 1996.

[4] R. Berghammer, B. Leoniuk, and U.Milanese. Implementation of relational algebra using binary decision diagrams. In *Proc. 6th Intl. Workshop on Relational Methods in Computer Science*, volume 2561 of *LNCS*, pages 241–257. Springer, 2002.

[5] C. Brink, W. Kahl, and G. Schmidt, editors. *Relational methods in computer science.* Advances in Computing Science. Springer, 1997.

[6] J. Esparza and K. Heljanko. A new unfolding approach to LTL model checking. In *Proc. 27th Intl. Coll. on Automata, Languages, and Programming*, volume 1853 of *LNCS*, pages 475–486. Springer-Verlag, 2000.

[7] J. Esparza and M. Nielsen. Decibility issues for Petri nets - a survey. *Journal of Information Processing and Cybernetics*, 30(3):143–160, 1994.

[8] J. Esparza and C. Schröter. Unfolding based algorithms for the reachability problem. *Fundamenta Informaticae*, 47(3–4):231–245, 2001.

[9] H. M. Fahmy, R. C. Holt, and J. R. Cordy. Wins and losses of algebraic transformations of software architecture. In *IEEE 16th Intl. Conf. on Automated Software Engineering*, 2001.

[10] A. Fronk and R. Berghammer. Applying relational algebra in 3D graphical software design. In *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *LNCS*, pages 62–73. Springer, 2003.

[11] A. Fronk and R. Berghammer. Solving Design Problems in OO-Software Engineering with Relations and RelView. *Journal on Relational Methods in Computer Science*, 1, 2004. To appear.

[12] T. Hoffmann. *Fallstudien relationaler Programmentwicklung am Beispiel ausgewählter Graphdurchlaufstrategien.* PhD thesis, Universität Kiel, 2002.

[13] B. Konikowska and E. Orlowska. A relational formalisation of a generic many-valued modal logic. In *Relational methods for computer science applications.* Physica-Verlag, 2001.

[14] J. Pleumann, J. Schröder, and K. Alfert. Software engineering education needs adequate modeling tools. In *Proc. 17th Conf. on Software Engineering Education & Training*, pages 72–77. IEEE Computer Society, 2004.

[15] G. Schmidt and T. Ströhlein. *Relations and graphs.* EATCS Monographs on Theoretical Computer Science. Spinger, 1993.

[16] P. H. Starke. *Analyse von Petri-Netz-Modellen.* Teubner, 1990.