Relation-Algebraic Analysis of Petri Nets with RELVIEW

Rudolf Berghammer [*]
Burghard von Karger [*]
Christiane Ulke [*]

ABSTRACT We present a method for specifying and implementing algorithms for the analysis of Petri nets. It is formally grounded in relational algebra. Specifications are written in ordinary predicate logic and then transformed systematically into relational programs which can be executed directly in RELVIEW, a graphical computer system for calculating with relations. Our method yields programs that are correct by construction. Its simplicity and efficiency is illustrated in many examples.

1 Introduction

Petri nets [9, 10] are widely used for designing and modeling concurrent and interacting processes. The success of Petri nets derives from their intuitive graphical representation which has great appeal even for people who are not familiar with the underlying theory. Furthermore, they have a well-defined semantics which unambiguously defines the behaviour of a net and allows formal analysis. And, finally, since they may contain cycles, a large class of processes can be represented by finite nets of manageable sizes.

In recent years, Tarski's relational algebra [13] has been used successfully for formal problem specification, prototyping, and algorithm development. Relations are well suited for reasoning about discrete structures in general and graphs in particular [12, 2, 3]. Since the static part of a Petri net is a directed graph, relational algebra is very promising for computer-aided investigations of their structure. Many interesting properties of Petri nets can be expressed in relational algebra. This is easiest for static properties such as causality and free choice but possible also for dynamic qualities like reachability and liveness.

The design of a relational algorithm starts from a logical problem specification that describes the desired result of a computation. With the aid of simple but rigorous transformation rules the specification is translated

[*]Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Preusserstraße 1–9, D–24105 Kiel, Germany

stepwise into a relational term. The goal of this transformation is the elimination of all quantifiers. In case of success, the resulting relational expression can be executed directly and efficiently in RELVIEW [4]. In this way, a program is built up very quickly and its correctness is guaranteed by a completely formal development.

Since RELVIEW can manipulate relations very efficiently, the performance of our programs is often good enough. However, in some cases optimizations are possible. Then again the formal framework of relational algebra can be very helpful because we can use its highly developed apparatus for transforming a given relational expression into a more efficient one.

The relational approach to specification, prototyping and design applies, at least in principle, to all discrete structures that can be represented naturally by binary relations. For the purpose of presentation we restrict ourselves here to a certain class of Petri nets, known as condition/event nets. A quite different set of graph-theoretic algorithms has been handled in the same style in [2, 3].

2 Relation-Algebraic Preliminaries

A *typed relation* $R : X \leftrightarrow Y$ consists of a domain X, a range Y and a set $R \subseteq X \times Y$. The set of all (typed) relations with domain X and range Y is denoted by $[X \leftrightarrow Y]$. When the type is clear, we abbreviate $R : X \leftrightarrow Y$ to R.

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. Since this matrix interpretation is well suited for a graphical representation, we use Boolean 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), and $R;S$ (composition). The empty relation is denoted by O, the universal relation by L, and the identity relation by I. The latter relation is the relation-level description of the meta-level symbol "=". For relation inclusion we write $R \subseteq S$. In a component-free manner, now we introduce some further relational notions which are needed in this article. Further details can be found in the textbook [12].

2.1 *Closures*

A relation $R : X \leftrightarrow X$ is reflexive if $\mathsf{I} \subseteq R$ and transitive if $R;R \subseteq R$. The least transitive relation containing R is called the transitive closure of R and denoted by $R^+ = \bigcup_{i \geq 1} R^i$, while the least reflexive and transitive relation containing R is called the reflexive-transitive closure of R and denoted by

$R^* = \bigcup_{i \geq 0} R^i$. Obviously, we have the equations $R^+ = R; R^* = R^*; R$ and $R^* = \mathrm{I} \cup R^+$.

2.2 Mappings

Let $R : X \leftrightarrow Y$ be a relation. Then R is said to be functional if $R^\mathsf{T}; R \subseteq \mathrm{I}$, and total if $R; \mathsf{L} = \mathsf{L}$. As usual, a functional and total relation is called a mapping. A relation R is injective if R^T is functional and surjective if R^T is total.

2.3 Description of Sets

Relational algebra offers two different ways of describing the subsets of a given set.

The first representation uses vectors, i.e., relations $v : X \leftrightarrow Y$ with $v = v; \mathsf{L}$. This condition means: Whatever set Z and universal relation $\mathsf{L} : Y \leftrightarrow Z$ we choose, an element x from X is either in relation $v; \mathsf{L}$ to none of the elements of Z or to all elements of Z. As for a vector $v : X \leftrightarrow Y$ the set Y is irrelevant, we consider in the following only vectors $v : X \leftrightarrow 1$ with a specific singleton set 1 as range and omit the second subscript. 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.

A vector $v : X \leftrightarrow 1$ is said to be a point if it is injective and surjective. These properties mean that it describes a singleton set. In the matrix model, hence a point is a Boolean column vector in which exactly one component is true.

Instead of vectors, we can use injective mappings for representing subsets. Given an injective mapping $\imath : Y \leftrightarrow X$, we call Y a subset of X given by $\imath$. If Y is a subset of X given by $\imath$, then the vector $\imath^\mathsf{T}; \mathsf{L} : X \leftrightarrow 1$, where $\mathsf{L} : Y \leftrightarrow 1$, describes Y in the above sense. Clearly, the transition in the other direction, i.e., the construction of an injective mapping $\imath(v) : Y \leftrightarrow X$ from a given vector $v : X \leftrightarrow 1$ describing Y, is also possible. In combination with the set-theoretic membership relation

$$\varepsilon : X \leftrightarrow 2^X \qquad \varepsilon_{xs} :\Longleftrightarrow x \in s\,, \tag{1}$$

the relation-level equivalent of the meta-level symbol "$\in$", injective mappings can be used to enumerate sets of sets. More specifically, if the vector $v : 2^X \leftrightarrow 1$ describes a subset $\mathcal{S}$ of the powerset 2^X, then it is straightforward to compute an injection $\imath(v) : \mathcal{S} \leftrightarrow 2^X$, from which we obtain the elements of $\mathcal{S}$ as the columns of the relation $\varepsilon; \imath(v)^\mathsf{T} : X \leftrightarrow \mathcal{S}$. If X is finite, this leads to an economic representation of $\mathcal{S}$ by a Boolean matrix with $|X|$ rows and $|\mathcal{S}|$ columns.

2.4 Residuals and Symmetric Quotients

Residuals are the greatest solutions of certain inclusions. The left residual of S over R (in symbols $S\,/\,R$) is the greatest relation X such that $X;R \subseteq S$ and the right residual of S over R (in symbols $R\backslash S$) is the greatest relation X such that $R;X \subseteq S$. We will also 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 X such that $R;X \subseteq S$ and $X;S^{\mathsf{T}} \subseteq R^{\mathsf{T}}$. In terms of the basic operations we have

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

The left residual is only defined if both relations have the same range and the right residual and the symmetric quotient are only defined if both relations have the same domain. Translating the first two equations into component-wise predicate logic notation yields

$$(S\,/\,R)_{yx} \Longleftrightarrow \forall\, z\ R_{xz} \rightarrow S_{yz} \qquad (R\setminus S)_{xy} \Longleftrightarrow \forall\, z\ R_{zx} \rightarrow S_{zy}\,. \tag{2}$$

In particular, for $S : Y \leftrightarrow Z$ and $R : Z \leftrightarrow X$ we obtain the two correspondences

$$(S\,/\,\mathsf{L})_y \Longleftrightarrow \forall\, z\ S_{yz} \qquad (\overline{R}\setminus \mathsf{O})_x \Longleftrightarrow \forall\, z\ R_{zx} \tag{3}$$

for single first-order universal quantification using a relation $\mathsf{L} : 1 \leftrightarrow Z$ and a vector $\mathsf{O} : Z \leftrightarrow 1$. And, finally, in component-wise notation the symmetric quotient satisfies the equivalence

$$\mathsf{syq}(R,S)_{xy} \Longleftrightarrow \forall\, z\ R_{zx} \leftrightarrow S_{zy}\,. \tag{4}$$

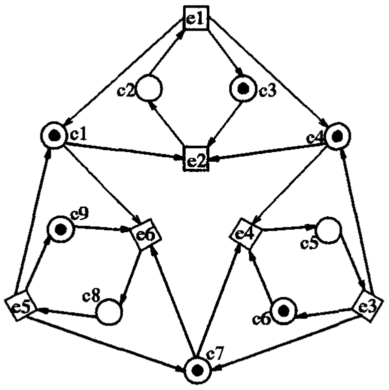
Let us consider (4) for the special case where R is a membership relation $\varepsilon : X \leftrightarrow 2^X$ and S is a vector $v : X \leftrightarrow 1$. Then the type of $\mathsf{syq}(\varepsilon, v)$ is $[2^X \leftrightarrow 1]$ and for each set Y from 2^X we have $\mathsf{syq}(\varepsilon, v)_Y$ if and only if $\forall\, z\ z \in Y \leftrightarrow v_z$. As a consequence, $\mathsf{syq}(\varepsilon, v) : 2^X \leftrightarrow 1$ is exactly the point in the powerset corresponding to the vector v.

3 Nets and their Relational Representation

In this section we recall the basics of condition/event nets and explain their representation in the RELVIEW tool.

3.1 Nets

A *(condition/event) net* $\mathcal{N}$ is a bipartite directed graph, which we represent as a a quadruple $\mathcal{N} = (C, E, R, S)$ with relations $R : C \leftrightarrow E$ and $S : E \leftrightarrow C$. The elements of C and E are called *conditions* and *events*, respectively, and we require that $C \cap E = \emptyset$. In the graphical representation, conditions (also known as *places*) are drawn as circles whereas events appear as squares. A *marking* is simply a subset M of C. A marked net $(\mathcal{N}, M)$ is visualized by decorating each condition $c \in M$ with a bullet, called a *token*. For example,



depicts a marked net with nine conditions $C = \{c_1, \ldots, c_9\}$, six events $E = \{e_1, \ldots, e_6\}$ and marking $M = \{c_1, c_3, c_4, c_6, c_7, c_9\}$. The relation R (resp. S) is coded by the set of arrows leading into (resp. out of) squares.

A net is a statical structure whereas markings are subject to change. The dynamic evolution of a marked net is described by a simple token game which specifies the effect of events on the current marking. An event e is currently *enabled* if all its predecessors but none of its successors carry a mark. In this case the *execution* (or *firing*) of e results in a new marking N which is obtained from the previous marking M by removing all predecessors of e and then adding all successors of e, i.e., by $N = (M \setminus \mathsf{pred}(e)) \cup \mathsf{succ}(e)$. In this way, by

$$M \xrightarrow{e} N \;:\Longleftrightarrow\; \begin{array}{l} e \text{ is enabled by } M \text{ and its exe-} \\ \text{cution transforms } M \text{ into } N \end{array}$$

every (unmarked) net induces a labeled transition relation on markings.

The above net is a somewhat simplified description of E.W. Dijkstra's dining philosophers [5]. Three philosophers are sitting round a table with a large bowl of tangled spaghetti in the middle. A hungry philosopher needs two forks to eat but there are only three forks on the entire table, one between each pair of neighbours. Each philosopher is thinking most of the time but can decide to start eating at any time provided both his forks are free. After eating his fill, he is supposed to return the forks to their places and go back into thinking mode. The initial marking indicates that all philosophers are busy thinking (c_3, c_6, and c_9) and all three forks are available (c_1, c_4, and c_7). The eating states c_2, c_5, and c_8 are unmarked.

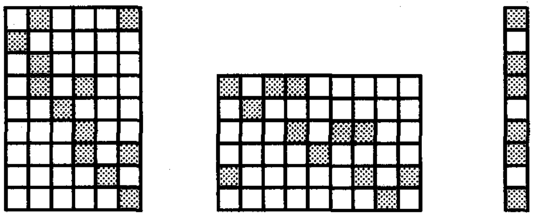
The transitions e_2, e_4, and e_6 from thinking to eating are enabled (although the philosophers may only eat one at a time) whereas the transitions from eating to thinking are disabled.

3.2 The RELVIEW System

RELVIEW [4] is a relation-based computer system for visualization, analysis and manipulation of discrete structures. Written in the C programming language, it runs under X windows and makes full use of the graphical user interface. Currently RELVIEW is used in about 30 installations all over the world.

All data are represented as binary relations, which RELVIEW visualizes in different ways. RELVIEW offers several different algorithm for pretty-printing a relation as a directed graph, including an algorithm for drawing bipartite graphs such as Petri nets. Alternatively, a relation may be displayed as a Boolean matrix which is very useful for visual editing and also for discovering various structural properties that are not evident from a graphical presentation.

For example, in RELVIEW the marked dining philosophers net is represented by the following relations (matrices) R and S and the (column) vector *init*:



RELVIEW can manage as many relations simultaneously as memory allows and the user may manipulate and analyse them by combining them with the operators of relational algebra. The elementary operations can be accessed through simple mouse-click and combined into *relational programs* which can then be stored and applied to many sets of input data. Because RELVIEW often is used on large input data, we have incorporated some very efficient routines for computing relational products, residuals and transitive closures.

Relational programs are extremely compact: Every program considered in this paper easily fits on a single line. To the uninitiated they seem arcane. However, that does not mean relational programming is difficult. On the contrary, each program is constructed from its obvious logical specification in a short series of refinement steps each of which is formally based on one of a very small set of transformation rules. As a result, every relational program we present in this paper is *correct by construction*.

4 Reachability and Liveness

Given a net and two markings M and N, we say that N is reachable from M iff there is a sequence of transitions $M \xrightarrow{e_1} \dots \xrightarrow{e_n} N$ that transforms M into N. Many safety properties of nets depend on the (un-)reachability of certain markings. Unlike the properties we considered in the previous section, reachability is a dynamic quality in the sense that its definition involves a potentially large number of transition steps. As a consequence, the costs of computing reachability are inherently exponential. Nevertheless the relational program for testing reachability which we derive in Sec. 4.1 is very useful for experimenting with small to medium-sized nets. It can be used as a building block for analysing more specific properties such as liveness which we investigate in Sec. 4.2.

4.1 Reachability

Reachability is defined in terms of sequences of transitions. Therefore, in the first step of our development we consider a single transition from a marking M to a marking N which is caused by the execution of an event e. We have to transscribe the definition of the transition relation of a net into a logical predicate. The first condition in that definition requires that M enables e which yields

$$(\forall c\ R_{ce} \rightarrow c \in M) \wedge (\forall c\ S_{ec} \rightarrow c \notin M)\,.$$

Now we represent events by points from $[E \leftrightarrow 1]$. Then $R;e : C \leftrightarrow 1$ is the vector of the set of predecessors and $S^{\mathsf{T}};e : C \leftrightarrow 1$ the vector of set of the successors of the point $e : E \leftrightarrow 1$. Furthermore, a condition c is a predecessor of e if and only if $(R;e)_c$ and a successor of e if and only if $(S^{\mathsf{T}};e)_c$. Hence, the above formula becomes

$$(\forall c\ (R;e)_c \rightarrow c \in M) \wedge (\forall c\ (S^{\mathsf{T}};e)_c \rightarrow c \notin M)\,.$$

Using the correspondences between certain kinds of logical and relation-algebraic constructions, our next aim is to replace the set-theoretic and logical symbols of this formula with relational operations and "outermost" subscripts M, N following the general method outlined in the introduction. The desired form is derived by

$$\begin{array}{lll}
& (\forall c\ (R;e)_c \rightarrow c \in M) \wedge (\forall c\ (S^{\mathsf{T}};e)_c \rightarrow c \notin M) & \\
\iff & (\forall c\ (R;e)_c \rightarrow \varepsilon_{cM}) \wedge (\forall c\ (S^{\mathsf{T}};e)_c \rightarrow \overline{\varepsilon}_{cM}) & (1),\ \varepsilon : C \leftrightarrow 2^C \\
\iff & (\forall c\ (R;e;\mathsf{L})_{cN} \rightarrow \varepsilon_{cM}) \wedge (\forall c\ (S^{\mathsf{T}};e;\mathsf{L})_{cN} \rightarrow \overline{\varepsilon}_{cM}) & \mathsf{L} : 1 \leftrightarrow 2^C \\
\iff & (R;e;\mathsf{L} \setminus \varepsilon)_{NM} \wedge (S^{\mathsf{T}};e;\mathsf{L} \setminus \overline{\varepsilon})_{NM} & (2) \\
\iff & ((R;e;\mathsf{L} \setminus \varepsilon)^{\mathsf{T}} \cap (S^{\mathsf{T}};e;\mathsf{L} \setminus \overline{\varepsilon})^{\mathsf{T}})_{MN}\,, &
\end{array}$$

where composition binds more than the residuals. If e is executed, the new marking N results from the old marking M by replacing the predecessors of e with its successors. On account of our point representation $e : E \leftrightarrow 1$ of events and since thus $\overline{R;e} : C \leftrightarrow 1$ is the complement of set of the predecessors of e, this is specified by the formula

$$\forall c\ (c \in M \wedge \overline{R;e}_c) \vee (S^{\mathsf{T}};e)_c \leftrightarrow c \in N\,.$$

Again, we are able to replace all the set-theoretic and predicate logic symbols with relational operations and subscripts M and N; a possible derivation is

$$\begin{array}{lll} & \forall c\ (c \in M \wedge \overline{R;e}_c) \vee (S^{\mathsf{T}};e)_c \leftrightarrow c \in N & \\ \Longleftrightarrow & \forall c\ (\varepsilon_{cM} \wedge \overline{R;e}_c) \vee (S^{\mathsf{T}};e)_c \leftrightarrow \varepsilon_{cN} & (1),\ \varepsilon : C \leftrightarrow 2^C \\ \Longleftrightarrow & \forall c\ (\varepsilon_{cM} \wedge (\overline{R;e};\mathsf{L})_{cM}) \vee (S^{\mathsf{T}};e;\mathsf{L})_{cM} \leftrightarrow \varepsilon_{cN} & \mathsf{L} : 1 \leftrightarrow 2^C \\ \Longleftrightarrow & \forall c\ ((\varepsilon \cap \overline{R;e};\mathsf{L}) \cup S^{\mathsf{T}};e;\mathsf{L})_{cM} \leftrightarrow \varepsilon_{cN} & \\ \Longleftrightarrow & \mathsf{syq}((\varepsilon \cap \overline{R;e};\mathsf{L}) \cup S^{\mathsf{T}};e;\mathsf{L}, \varepsilon)_{MN} & (4). \end{array}$$

Now we can remove the subscripts M and N in the results of the last two derivations. Putting together the remainig relation-algebraic expressions, we arrive at the component-free specification

$$\begin{array}{rl} \mathit{trans}(R,S,e) := & (R;e;\mathsf{L} \setminus \varepsilon)^{\mathsf{T}} \\ & \cap\ (S^{\mathsf{T}};e;\mathsf{L} \setminus \overline{\varepsilon})^{\mathsf{T}} \\ & \cap\ \mathsf{syq}((\varepsilon \cap \overline{R;e};\mathsf{L}) \cup S^{\mathsf{T}};e;\mathsf{L}, \varepsilon) \end{array}$$

of a relation $\mathit{trans}(R,S,e) : 2^C \leftrightarrow 2^C$ that describes all possible single transitions between markings of $\mathcal{N}$ which are caused by an execution of the event (point) $e : E \leftrightarrow 1$.

Having derived a relational specification of the transition relation, we have solved the most difficult part of the reachability problem. By definition, the reachability relation on markings we have searched for is precisely the reflexive-transitive closure of the union of all transition relations. Hence, we define a relation

$$\mathit{reach}(R,S) := (\bigcup_{e \in \mathsf{P}(E)} \mathit{trans}(R,S,e))^*$$

the type of which is also $[2^C \leftrightarrow 2^C]$, where $\mathsf{P}(E)$ denotes the set of all points from $[E \leftrightarrow 1]$.

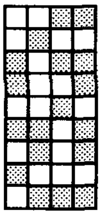
Also testing whether one marking can be reached from another is now trivial. If they are given as vectors $m : C \leftrightarrow 1$ and $n : C \leftrightarrow 1$, we then produce the corresponding points $\mathsf{syq}(\varepsilon, m) : 2^C \leftrightarrow 1$ and $\mathsf{syq}(\varepsilon, n) : 2^C \leftrightarrow 1$ in the powerset as described in Sec. 2.4 and have that n is reachable from a marking vector m if and only if $\mathsf{syq}(\varepsilon, m);\mathsf{syq}(\varepsilon, n)^{\mathsf{T}} \subseteq \mathit{reach}(R,S)$.

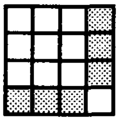
To obtain the set of all markings reachable from $m : C \leftrightarrow 1$, we start by computing the vector $\mathit{Reach}(R,S,m) : 2^C \leftrightarrow 1$ of the relation-theoretic successors (wrt. the reachability relation) of the point corresponding to m using

$$\mathit{Reach}(R,S,m) := \mathit{reach}(R,S)^{\mathsf{T}} ; \mathsf{syq}(\varepsilon, m)\,.$$

Then, we represent the elements contained in the subset of 2^C described by the vector $\mathit{Reach}(R,S,m)$ as the columns of a Boolean matrix as described in Sec. 2.3.

We have formulated the above specifications in the RELVIEW system and applied to the relational representation of the philosophers net given in Sec. 3.2. The left-hand of the following two RELVIEW pictures shows the column-wise representation of the four markings reachable from the initial one; on the right-hand we have the "transition matrix" describing the possible transitions between these markings. This latter matrix is obtained as value of the relational expression $\imath(r); (\bigcup_e \mathit{trans}(R,S,e)); \imath(r)^{\mathsf{T}}$, where e ranges over the points $\mathsf{P}(E)$ and $r := \mathit{Reach}(R,S,\mathit{init})$.





The last column of the first matrix describes the initial marking *init* (all philosophers are thinking). Three different markings are reachable (exactly one philosopher eats) and each of them corresponds to one of the first three columns. The 4×4 transition matrix shows that each of the three eating states can evolve into the thinking state and vice versa, but that no other transitions are possible. Thus, every sequence of markings/events of the token game of the philosophers net which starts with the initial marking corresponds to a run of a philosopher's dinner and vice versa.

4.2 Liveness

In the literature one finds several notions of liveness. Five different formal definitions of a marking to be live are given, investigated, and compared in [8]. All of them can easily be specified in our relational framework. In the following, we concentrate on the version which is preferred by [8]: Given a net $\mathcal{N} = (C,E,R,S)$, an event e is said to be *dead* under a marking M if there is no reachable marking N which enables e, and a marking M is called *live* if for all markings N reachable from M and all events e we have that e is not dead under N.

We start our development of an executable relational specification of liveness by reconsidering the predicate logic formula

$$(\forall c\; R_{ce} \to c \in M) \wedge (\forall c\; S_{ec} \to c \notin M)$$

which specifies that the marking M enables the event e. In contrast with Sec. 4.1, however, we do not represent events by points in the relational sense. This allows the following derivation which replaces the set-theoretic and predicate logic symbols with relational operations and the subscripts M and e:

$$\begin{aligned}
& (\forall c\; R_{ce} \to c \in M) \wedge (\forall c\; S_{ec} \to c \notin M) & \\
\Longleftrightarrow\; & (\forall c\; R^{\mathsf{T}}_{ec} \to \varepsilon^{\mathsf{T}}_{Mc}) \wedge (\forall c\; S_{ec} \to \overline{\varepsilon^{\mathsf{T}}}_{Mc}) & (1),\, \varepsilon : C \leftrightarrow 2^C \\
\Longleftrightarrow\; & (\varepsilon^{\mathsf{T}} / R^{\mathsf{T}})_{Me} \wedge (\overline{\varepsilon^{\mathsf{T}}} / S)_{Me} & (2) \\
\Longleftrightarrow\; & ((\varepsilon^{\mathsf{T}} / R^{\mathsf{T}}) \cap (\overline{\varepsilon^{\mathsf{T}}} / S))_{Me} \,.
\end{aligned}$$

Now the subscripts M and e can be removed, yielding

$$\mathit{enable}(R,S) := (\varepsilon^{\mathsf{T}} / R^{\mathsf{T}}) \cap (\overline{\varepsilon^{\mathsf{T}}} / S)$$

as a component-free specification of the enabling relation of type $[2^C \leftrightarrow E]$. In the Boolean matrix model this means that the entry in the M-row and e-column of $\mathit{enable}(R,S)$ is true if and only if e is enabled by M.

Combining the above relation $\mathit{enable}(R,S)$ with the reachability relation $\mathit{reach}(R,S)$ derived in Sec. 4.1, we have that e is dead under M if and only if

$$\neg\exists N\; \mathit{reach}(R,S)_{MN} \wedge \mathit{enable}(R,S)_{Ne} \,.$$

So the set of all such pairs M, e, the "is-dead-under" relation, is given by

$$\mathit{dead}(R,S) := \overline{\mathit{reach}(R,S); \mathit{enable}(R,S)}$$

which is a relational specification of type $[2^C \leftrightarrow E]$.

To specify liveness in predicate logic, finally, we use the reachability relation $\mathit{reach}(R,S)$ again, but now in combination with $\mathit{dead}(R,S)$. We get that a marking M is live if and only if the formula

$$\forall N\; \forall e\; \mathit{reach}(R,S)_{MN} \to \neg \mathit{dead}(R,S)_{Ne}$$

holds. In this case, the replacement of the set-theoretic and predicate logic symbols with relational operations and the subscript M proceeds as follows:

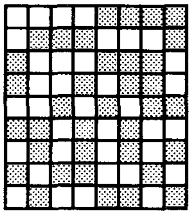
$$\begin{aligned}
& \forall N\; \forall e\; \mathit{reach}(R,S)_{MN} \to \neg \mathit{dead}(R,S)_{Ne} & \\
\Longleftrightarrow\; & \forall N\; \mathit{reach}(R,S)_{MN} \to \neg\exists e\; \mathit{dead}(R,S)_{Ne} & \\
\Longleftrightarrow\; & \neg\exists N\; \mathit{reach}(R,S)_{MN} \wedge \exists e\; \mathit{dead}(R,S)_{Ne} & \\
\Longleftrightarrow\; & \neg\exists N\; \mathit{reach}(R,S)_{MN} \wedge (\mathit{dead}(R,S); \mathsf{L})_N & \mathsf{L} : E \leftrightarrow 1 \\
\Longleftrightarrow\; & \overline{\mathit{reach}(R,S); \mathit{dead}(R,S); \mathsf{L}}_M \,.
\end{aligned}$$

Finally, a removal of the subscript M yields

$$live(R,S) := \overline{reach(R,S); dead(R,S); \mathsf{L}}$$

as the desired vector of type $[2^C \leftrightarrow 1]$ which describes all markings of $\mathcal{N}$ which are live. Considered as Boolean column vector this means that for a subset M of 2^C the M-component of $live(R,S)$ is true if and only if M is a live marking.

The following picture shows the column-wise representation of the eight live markings of the philosophers net as computed by RELVIEW:



From the columns 1, 2, 5, and 6 of this matrix we see that every marking reachable from *init* is live. This means that the marked philosophers net is live, a result which can also be verified with RELVIEW using the test $Reach(R, S, init) \subseteq live(R, S)$. There are four more live markings, but none of them corresponds to a "real" state in a philosopher's dinner. For example, the marking depicted in the third column describes the impossible situation where each philosopher is eating

5 Protoptyping Some Further Dynamic Properties

Now we consider further examples for prototyping relational specifications of dynamic net properties. First, we consider concurrency and conflicts. Then we are then concerned with deadlocks and traps. Finally, we treat the notion of contact-freeness.

5.1 Concurrency and Conflicts

When we use nets to model concurrent and interacting processes, we cannot state whether and when an event will happen, but can only specify conditions that enable it. In this connection, however, it is very interesting to know which events can take place concurrently at a given state. This leads to the following notion. Two events e and f are *concurrently enabled* by a marking M if both of them are enabled by M in the sense of Sec. 3.1 and they have neither a predecessor nor a successor in common.

Assume $\mathcal{N} = (C, E, R, S)$ is a net. Expressed as logical formulae, the first part of the definition of the two events e and f being concurrently enabled by the marking M reads as

$$\begin{aligned} & (\forall c\ R_{ce} \rightarrow c \in M) \wedge (\forall c\ S_{ec} \rightarrow c \notin M) \\ \wedge\ & (\forall c\ R_{cf} \rightarrow c \in M) \wedge (\forall c\ S_{fc} \rightarrow c \notin M) \end{aligned}$$

and the second part as

$$\neg(\exists c\ R_{ce} \wedge R_{cf}) \wedge \neg(\exists c\ S_{ec} \wedge S_{fc}) .$$

Compared with Sec. 4.1 and Sec. 4.2, in the sequel we choose a third variant for a transformation of the formulae describing that a marking enables an event. Assume the marking to be represented by a vector $m : C \leftrightarrow 1$. Then $c \in M$ resp. $c \notin M$ will be replaced by m_c resp. $\overline{m}_c$ and we get for the first part the derivation

$$\begin{aligned} & (\forall c\ R_{ce} \rightarrow m_c) \wedge (\forall c\ S_{ec} \rightarrow \overline{m}_c) \\ & \wedge (\forall c\ R_{cf} \rightarrow m_c) \wedge (\forall c\ S_{fc} \rightarrow \overline{m}_c) \\ \Longleftrightarrow\ & (R \backslash m)_e \wedge (S^{\mathsf{T}} \backslash \overline{m})_e \wedge (R \backslash m)_f \wedge (S^{\mathsf{T}} \backslash \overline{m})_f \qquad (2) \\ \Longleftrightarrow\ & ((R \backslash m) \cap (S^{\mathsf{T}} \backslash \overline{m}))_e \wedge ((R \backslash m) \cap (S^{\mathsf{T}} \backslash \overline{m}))_f \\ \Longleftrightarrow\ & (((R \backslash m) \cap (S^{\mathsf{T}} \backslash \overline{m})); ((R \backslash m) \cap (S^{\mathsf{T}} \backslash \overline{m}))^{\mathsf{T}})_{ef} . \end{aligned}$$

A transformation of the second part is even simpler. We obtain

$$\begin{aligned} & \neg(\exists c\ R_{ce} \wedge R_{cf}) \wedge \neg(\exists c\ S_{ec} \wedge S_{fc}) \\ \Longleftrightarrow\ & \overline{R^{\mathsf{T}}; R}_{ef} \wedge \overline{S; S^{\mathsf{T}}}_{ef} \\ \Longleftrightarrow\ & \overline{R^{\mathsf{T}}; R \cup S; S^{\mathsf{T}}}_{ef} . \end{aligned}$$

Combining the results of these two derivations and dropping the subscripts e and f, we arrive at the relational specification

$$concur(R, S, m) := aux(R, S, m); aux(R, S, m)^{\mathsf{T}} \cap \overline{R^{\mathsf{T}}; R \cup S; S^{\mathsf{T}}}$$

of type $[E \leftrightarrow E]$, where the auxiliary function aux is defined by

$$aux(R, S, m) := (R \backslash m) \cap (S^{\mathsf{T}} \backslash \overline{m}) .$$

It computes for the net $\mathcal{N}$ and a given marking vector $m : C \leftrightarrow 1$ all pairs of events which are concurrently enabled by m. If we evaluate $concur(R, S, m)$ using Boolean matrices and vectors and their standard procedures for implementing relational algebra, then the run time is determined by the times needed for the compositions $R^{\mathsf{T}}; R$ and $S; S^{\mathsf{T}}$.

The dual notion to concurrency is that of a conflict. Given a marking M, two events e and f are in *conflict* under M if they are both enabled by M in the sense of Sec. 3.1, have a common predecessor or a common successor, and are different. In a predicate logic form we have, hence, the conjunction of the above formula describing that e and f are enabled by the marking M with

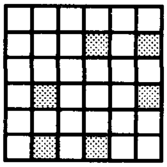
$$(\exists c\ R_{ce} \wedge R_{cf}) \vee (\exists c\ S_{ec} \wedge S_{fc}),$$

saying that $\mathsf{pred}(e) \cap \mathsf{pred}(f) \neq \emptyset$ or $\mathsf{succ}(e) \cap \mathsf{succ}(f) \neq \emptyset$, and the inequality $e \neq f$. For a net $\mathcal{N} = (C, E, R, S)$ and a vector $m : C \leftrightarrow 1$ representing the marking, this immediately leads (using the function *aux* again) to

$$conf(R, S, m) := aux(R, S, m); aux(R, S, m)^{\mathsf{T}} \cap (R^{\mathsf{T}}; R \cup S; S^{\mathsf{T}}) \cap \bar{\mathrm{I}}$$

as a relational specification of type $[E \leftrightarrow E]$ for computing all pairs of events which are in conflict under m. It has the same time complexity as $concur(R, S, m)$.

Let us apply the results to our running example. Since two or more philosophers cannot eat at the same time, one would expect that no two events are concurrently enabled by a marking reachable from *init*. The RELVIEW system confirms this conjecture. For every column m_i of the column-wise representation of $Reach(R, S, init)$ given in Sec. 4.1, the evaluation of the relational specification $concur(R, S, m_i)$ yields the empty Boolean 6×6 matrix. On the other hand, conflicts do exist. For the columns m_1, m_2, m_3 the conflict matrix is empty, but for the last column – the initial marking vector – RELVIEW yields the following Boolean 6×6 matrix:



Hence, in the initial state we have conflicts between each pair of events representing transitions from eating to thinking. This result agrees with the transition matrix shown in Sec. 4.1.

5.2 Deadlocks and Traps

Let $\mathcal{N} = (C, E, R, S)$ be a net. A set D of conditions is called a *deadlock* if each of its predecessors is also a successor. The dual notion is that of a trap: A subset T of C is said to be a *trap* if its successor set is a subset of its predecessor set. Both deadlocks and traps are useful for reasoning about liveness properties; details about the deadlock approach to the liveness problem can be found in [7].

If we represent sets of conditions by vectors of type $[C \leftrightarrow 1]$, then $S; v$ describes the set of predecessors and $R^{\mathsf{T}}; v$ the set of successors of $v : C \leftrightarrow 1$.

Hence, it is very easy to test a single set to be a deadlock or a trap. We have that $d : C \leftrightarrow 1$ is a deadlock if and only if $S;d \subseteq R^{\mathsf{T}};d$ and, by exchanging the rôle of the relations S and R^{T}, also that $t : C \leftrightarrow 1$ is a trap if and only if $R^{\mathsf{T}};t \subseteq S;t$. These inclusions can be tested efficiently if we implement R, S by Boolean matrices and d, t by Boolean vectors and use again the standard procedures for the relational operations.

In the following we concentrate on algorithms describing all deadlocks and traps. For a net without restrictions, the number of deadlocks and traps can grow exponentially with its size. As in the case of the algorithms of Sec. 4, therefore, our approach will lead to a complexity which is exponential in time and space.

Expressed in predicate logic we have that a set D of conditions is a deadlock if and only if the formula

$$\forall e\ (\exists c\ c \in D \wedge S_{ec}) \rightarrow (\exists c\ c \in D \wedge R_{ce})$$

is valid and that a set T of conditions is a trap if and only if

$$\forall e\ (\exists c\ c \in T \wedge R_{ce}) \rightarrow (\exists c\ c \in T \wedge S_{ec})$$

holds. The first formula can be transformed as follows, in doing so replacing the set-theoretic and logical symbols with relational operations and the subscript D:

$$\begin{aligned}
& \forall e\ (\exists c\ c \in D \wedge S_{ec}) \rightarrow (\exists c\ c \in D \wedge R_{ce}) & \\
\Longleftrightarrow\ & \forall e\ (\varepsilon^{\mathsf{T}};S^{\mathsf{T}})_{De} \rightarrow (\varepsilon^{\mathsf{T}};R)_{De} & (1),\ \varepsilon : C \leftrightarrow 2^C \\
\Longleftrightarrow\ & \forall e\ (\overline{\varepsilon^{\mathsf{T}};S^{\mathsf{T}}} \cup \varepsilon^{\mathsf{T}};R)_{De} & \\
\Longleftrightarrow\ & ((\overline{S;\varepsilon}^{\mathsf{T}} \cup \varepsilon^{\mathsf{T}};R) \,/\, \mathsf{L})_D & (3),\ \mathsf{L} : 1 \leftrightarrow E.
\end{aligned}$$

Therefore, changing to a component-free relation-algebraic notation, we get

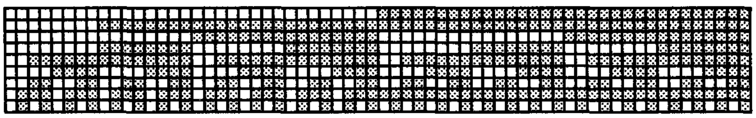
$$\mathit{deadlock}(R, S) := (\overline{S;\varepsilon}^{\mathsf{T}} \cup \varepsilon^{\mathsf{T}};R) \,/\, \mathsf{L}$$

as specification of the vector of type $[2^C \leftrightarrow 1]$ describing all deadlocks of the net $\mathcal{N}$. In the same way one obtains from the above predicate logic description of traps by exchanging the rôle of the relations R and S^{T} that

$$\mathit{trap}(R, S) := (\overline{\varepsilon^{\mathsf{T}};R} \cup (S;\varepsilon)^{\mathsf{T}}) \,/\, \mathsf{L}$$

is the vector of type $[2^C \leftrightarrow 1]$ describing all traps of $\mathcal{N}$.

Using RELVIEW to compute the set of all deadlocks of the philosophers net (which coincides with the set of all traps due to the net's symmetric form), we obtain the following column-wise representation:



Since none of these 64 deadlocks can be reached from *init*, so the (marked) philosophers net is deadlock-free. To verify this by hand is troublesome; a mechanical verification with RELVIEW is easy. We only need to evaluate the expressions $\mathit{Reach}(R, S, \mathit{init}) \cap \mathit{deadlock}(R, S)$ and test the result for emptiness.

5.3 Contact-Freeness

Suppose we have a net and a present marking M. A necessary condition for an event to be executed is that no condition of its successor set is contained in M. If an execution of an event is prevented since the successors are marked, then one speaks of a contact situation. Contact situations are undesirable. A marking M is said to be *contact-free* if it is true for every event e that $\mathsf{pred}(e) \subseteq M$ implies $\mathsf{succ}(e) \subseteq C \setminus M$ and $\mathsf{succ}(e) \subseteq M$ implies $\mathsf{pred}(e) \subseteq C \setminus M$.

Assume an event e and a marking M. Then we have that $\mathsf{pred}(e) \subseteq M$ implies $\mathsf{succ}(e) \subseteq C \setminus M$ if and only if

$$(\forall c\ R_{ce} \rightarrow c \in M) \rightarrow (\forall c\ S_{ec} \rightarrow c \notin M)$$

holds, and that $\mathsf{succ}(e) \subseteq M$ implies $\mathsf{pred}(e) \subseteq C \setminus M$ if and only if

$$(\forall c\ S_{ec} \rightarrow c \in M) \rightarrow (\forall c\ R_{ce} \rightarrow c \notin M)$$

is valid. In the first case, the replacement of the logical and set-theoretic symbols with relational operations and the subscripts e and M proceeds as follows:

$$\begin{array}{lll}
& (\forall c\ R_{ce} \rightarrow c \in M) \rightarrow (\forall c\ S_{ec} \rightarrow c \notin M) & \\
\Longleftrightarrow & (\forall c\ R_{ce} \rightarrow \varepsilon_{cM}) \rightarrow (\forall c\ S^{\mathsf{T}}_{ce} \rightarrow \overline{\varepsilon}_{cM}) & (1),\ \varepsilon : C \leftrightarrow 2^C \\
\Longleftrightarrow & (R \backslash \varepsilon)_{eM} \rightarrow (S^{\mathsf{T}} \backslash \overline{\varepsilon})_{eM} & (2) \\
\Longleftrightarrow & (\overline{R \backslash \varepsilon} \cup (S^{\mathsf{T}} \backslash \overline{\varepsilon}))_{eM}\,. &
\end{array}$$

By exchanging the rôle of the two relations R and S^{T} in this derivation, the second predicate logic formula for contact-freeness is transformed into

$$(\overline{S^{\mathsf{T}} \backslash \varepsilon} \cup (R \backslash \overline{\varepsilon}))_{eM}\,.$$

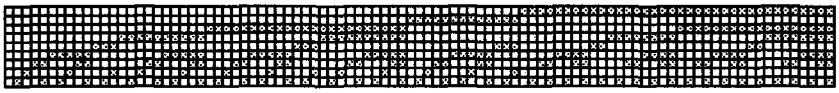
Now we combine these expressions and obtain

$$\begin{array}{lll}
& \forall e\ (\overline{R \backslash \varepsilon} \cup (S^{\mathsf{T}} \backslash \overline{\varepsilon}))_{eM} \wedge (\overline{S^{\mathsf{T}} \backslash \varepsilon} \cup (R \backslash \overline{\varepsilon}))_{eM} & \\
\Longleftrightarrow & \forall e\ ((\overline{R \backslash \varepsilon} \cup (S^{\mathsf{T}} \backslash \overline{\varepsilon})) \cap (\overline{S^{\mathsf{T}} \backslash \varepsilon} \cup (R \backslash \overline{\varepsilon})))_{eM} & \\
\Longleftrightarrow & (\overline{(\overline{R \backslash \varepsilon} \cup (S^{\mathsf{T}} \backslash \overline{\varepsilon})) \cap (\overline{S^{\mathsf{T}} \backslash \varepsilon} \cup (R \backslash \overline{\varepsilon}))} \backslash \mathsf{O})_e & (3),\ \mathsf{O} : E \leftrightarrow 1
\end{array}$$

which in turn yields the vector

$$\mathit{contactfree}(R, S) := \overline{(\overline{R \backslash \varepsilon} \cup (S^{\mathsf{T}} \backslash \overline{\varepsilon})) \cap (\overline{S^{\mathsf{T}} \backslash \varepsilon} \cup (R \backslash \overline{\varepsilon}))} \backslash \mathsf{O}$$

of type $[2^C \leftrightarrow 1]$ as component-free relational specification of the set of all contact-free markings of $\mathcal{N} = (C, E, R, S)$. As with deadlocks and traps, the complexity of this algorithm is exponential in time and space. Evaluating this term for the philosophers net reveals that there are exactly 95 contact-free markings. Their column-wise representation is:



Moreover, RELVIEW can check that $\mathit{Reach}(R, S, \mathit{init}) \subseteq \mathit{contactfree}(R, S)$ holds, thereby proving that no contact situation can be reached from the initial marking of the philosophers net. A marked net with this property is said to be *contact-free.*

6 Testing Structural Properties of Nets

Structural (or static) properties of a net can be decided from its definition as a bipartite graph without considering the token game. Their main purpose is to characterize subclasses of nets with nice characteristics. As an example, for the special subclass of nets called "synchronisation graphs" the reachability problem is polynomial in the size of the net [6]. By means of some examples, in this section we demonstrate how structural properties of nets can be decided using a relational approach.

6.1 Free Choice Nets

In general nets there may occur the situation that a marking can only enable an event if two further concurrently enabled events are executed in a specific order. To exclude such a confused situation, i.e., to allow that the choice of the event to execute is taken locally, the specific class of free choice nets has been introduced in [7]. Formally, a net $\mathcal{N} = (C, E, R, S)$ is called a *free choice net* if for all conditions c and events e from R_{ce} it follows that $\mathsf{succ}(c) = \{e\}$ or $\mathsf{pred}(e) = \{c\}$. This means that an event with a forward-branching predecessor may not be backwards-branching.

For a relation-algebraic specification of a net to be free choice we follow the pattern of Sec. 5.3. Hence, we start the formula

$$\forall e \, \forall c \; R_{ce} \rightarrow (\forall f \; R_{cf} \rightarrow e = f) \vee (\forall d \; R_{de} \rightarrow d = c)$$

expressing that $\mathcal{N}$ is a free choice net and remove then the universal quantification over e and c. A transformation of the resulting formula (now with free ocurrences of e and c) into a form with only relational operations and the subscripts e, c is obtained by

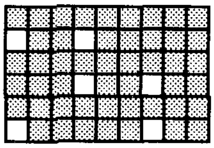
$$\begin{aligned} & R_{ce} \rightarrow (\forall f\ R_{cf} \rightarrow e = f) \vee (\forall d\ R_{de} \rightarrow d = c) \\ \Longleftrightarrow\ & \overline{R^{\mathsf{T}}}_{ec} \vee (\forall f\ R_{cf} \rightarrow \mathrm{I}_{ef}) \vee (\forall d\ R_{de} \rightarrow \mathrm{I}_{dc}) \\ \Longleftrightarrow\ & \overline{R^{\mathsf{T}}}_{ec} \vee (\mathrm{I} / R)_{ec} \vee (R \setminus \mathrm{I})_{ec} \qquad (2) \\ \Longleftrightarrow\ & (\overline{R^{\mathsf{T}}} \cup (\mathrm{I} / R) \cup (R \setminus \mathrm{I}))_{ec}\,. \end{aligned}$$

Note that we have used two identity relations during this development, viz. $\mathrm{I} : E \leftrightarrow E$ in the left residual I / R and $\mathrm{I} : C \leftrightarrow C$ in the right residual $R \setminus \mathrm{I}$. As an immediate consequence from the above derivation we obtain

$$\begin{aligned} & \mathcal{N} \text{ is a free choice net} \\ \Longleftrightarrow\ & \forall e\, \forall c\ (\overline{R^{\mathsf{T}}} \cup (\mathrm{I} / R) \cup (R \setminus \mathrm{I}))_{ec} \\ \Longleftrightarrow\ & \overline{R^{\mathsf{T}}} \cup (\mathrm{I} / R) \cup (R \setminus \mathrm{I}) = \mathsf{L} \qquad \mathsf{L} : E \leftrightarrow C\,. \end{aligned}$$

In the standard Boolean matrix model for relational algebra, the latter equality can be tested in a time complexity which is determined by the costs for computing the residuals.

The philosophers net is not a free choice net. Using RELVIEW, this can easily be verified and the system then yields:



This Boolean 6×9 matrix relates the events and conditions which fulfil the free choice property. A comparison of this matrix with the 6×9 universal matrix shows that this property is violated by exactly 6 pairs, viz. by (e_2, c_1), (e_2, c_4), (e_4, c_4), (e_4, c_7), (e_6, c_1), and (e_6, c_7).

6.2 Synchronisation Graphs and State Machines

We say that a net is a *synchronisation graph* (or *S-graph*) if every condition has at most one predecessor and at most one successor. Such nets model the branching (or splitting) of a process into concurrent threads and the synchronisation of these threads. Due to the absence of branching conditions for synchronisation graphs the reachability problem can be solved in polynomial time. The same holds in nets without branching events, i.e., in the case that every event has at most one predecessor and at most one successor. Such a net is said to be a *state machine* (or *T-graph*).

On account of our special representation of a net as a relational structure $\mathcal{N} = (C, E, R, S)$ we have that $\mathcal{N}$ is a synchronisation graph if and only if

S is injective and R is functional and that $\mathcal{N}$ is a state machine if and only if R is injective and S is functional. Efficient tests for Boolean matrices to be functional resp. injective inspect row by row resp. column by column, i.e., need only two nested loops.

Using the method outlined in the introduction, we are also able to develop relational specifications of the vectors of non-branching conditions resp. non-branching events such that the resulting algorithms are polynomial. In the case of a condition c, first we consider the property that it has at most one predecessor. The derivation of a relational specification from its predicate logic description proceeds as follows:

$$\begin{array}{lll} & \forall e\, \forall f\; S_{ec} \wedge S_{fc} \rightarrow e = f & \\ \Longleftrightarrow & \forall e\; S_{ec} \rightarrow \forall f\; S_{fc} \rightarrow \overline{\mathrm{I}}_{fe} & \mathrm{I} : E \leftrightarrow E \\ \Longleftrightarrow & \forall e\; S_{ec} \rightarrow (S \setminus \overline{\mathrm{I}})_{ce} & (2) \\ \Longleftrightarrow & \forall e\; (\overline{S^{\mathsf{T}}} \cup (S \setminus \overline{\mathrm{I}}))_{ce} & \\ \Longleftrightarrow & ((\overline{S^{\mathsf{T}}} \cup (S \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L})_c & (3),\ \mathsf{L} : 1 \leftrightarrow E\,. \end{array}$$

Next, we deal with the property that c has at most one successor. Its logical formalization is

$$\forall e\, \forall f\; R_{ce} \wedge R_{cf} \rightarrow e = f$$

and a replacement of S by R^{T} in the above derivation transforms it into

$$((\overline{R} \cup (R^{\mathsf{T}} \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L})_c\,.$$

It remains to put the two relational forms together and to remove the subscript c. In doing so, we arrive at

$$\textit{s-graph}(R, S) := ((\overline{S^{\mathsf{T}}} \cup (S \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L}) \cap ((\overline{R} \cup (R^{\mathsf{T}} \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L})$$

as the vector $\textit{s-graph}(R, S) : C \leftrightarrow 1$ of non-branching conditions. Hence, the net $\mathcal{N}$ is a synchronisation graph if and only if $\textit{s-graph}(R, S)$ equals the universal vector $\mathsf{L} : C \leftrightarrow 1$.

If we change the rôle of the relations R and S in the development of $\textit{s-graph}(R, S)$, then the result is the component-free specification

$$\textit{t-graph}(R, S) := ((\overline{R^{\mathsf{T}}} \cup (R \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L}) \cap ((\overline{S} \cup (S^{\mathsf{T}} \setminus \overline{\mathrm{I}})) \,/\, \mathsf{L})$$

of a vector of type $[E \leftrightarrow 1]$ for enumerating the non-branching events. In this specification we use an identity relation $\mathrm{I} : C \leftrightarrow C$ and an universal relation $\mathsf{L} : 1 \leftrightarrow C$.

For small examples these properties can easily be read off the matrix representation. For example, the philosophers net is neither a synchronisation graph nor a state machine, because R and S have both rows and columns with more than one entry.

6.3 Causal Nets

As a last structural property we consider causal nets introduced in [11]. A net is a *causal net* if it is a synchronisation graph and the set of its arcs, called its "flow relation", is cycle-free. The latter property implies that each event can occur only once. If we define a partial order on events by $e \leq f$ if and only if f can be executed only after e, then a causal net can be seen as the net-theoretic way to represent this partial order.

In Sec. 6.2 we have shown how to decide the property to be a synchronisation graph using relational algebra. Therefore, it remains to develop a similar test for cycle-freeness. To this end, let us represent the net $\mathcal{N} = (C, E, R, S)$ as an "ordinary" directed graph $\mathcal{N} = (V, F)$, where $V := C \cup E$ and the flow relation $F : V \leftrightarrow V$ has the special form[1]

$$F = \begin{pmatrix} \mathsf{O} & R \\ S & \mathsf{O} \end{pmatrix}.$$

The relation F is acyclic if and only if its transitive closure is contained in $\bar{\mathsf{I}} : V \leftrightarrow V$. A simple induction on the powers F^i of F shows the equation

$$F^+ = \begin{pmatrix} (R;S)^+ & (R;S)^+;R \\ (S;R)^+;S & (S;R)^+ \end{pmatrix}.$$

Thus, F is cyclefree if and only if $(R;S)^+ \subseteq \bar{\mathsf{I}}$ and $(S;R)^+ \subseteq \bar{\mathsf{I}}$. In the Boolean matrix model, the costs for these tests are the same as for computing the transitive closures, for instance we obtain cubic time complexity if S. Warshall's well-known algorithm is used.

7 Conclusion

We have captured many properties of condition/event nets in single-line relational programs which can be immediately executed in the RELVIEW system. This experience has taught us to use the RELVIEW system as a "programmable pocket calculator" for Petri nets. It cannot, of course, compete in machine efficiency with special purpose tools (although the complexities are usually the same). For structural properties such as causality and free choice the RELVIEW algorithms are easily sufficient whereas dynamic properties like reachability and liveness can only be tested for small to medium-sized nets. For example, the reachability relation for the philosophers net can be computed on a SUN workstation for up to five philosophers.

[1]We can also express F relation-algebraically in terms of R and S. But to achieve this, we need a relational specification of disjoint union which is beyond the aim of this article.

The real attraction of RELVIEW lies in its flexibility: New properties of nets (and new types of nets!), are introduced all the time and RELVIEW is an ideal tool for toying with new concepts while avoiding unnecessary overhead. We have used the system on many more examples, including a fair version of the philosophers net.

Even in those cases where the obvious transcription of a logical specification yields a relational algorithm of unacceptable complexity all hope is not lost. Relational algebra is a powerful transformation tool and it is often possible to derive an efficient algorithm from a prototype. A number of examples of this technique can be found in [2, 3].

We have performed the translation from logical specifications to relational programs manually, but our experience suggests that certain patterns occur very frequently, so that mechanical aid could be helpful. Of course, the transformation technique presented in this article is not sufficient for translating arbitrary first-order formulae to relational expressions and in some cases where a translation exists, a certain amount of creativity is required. Theoretically, completeness can be achieved by including the direct product of relations and adding appropriate rules. However, the use of products may lead to inefficient and obscure relational programs and is therefore best avoided.

For ease of presentation we have considered only condition/event nets in this paper, but other types of Petri nets can be explored in a similar way. In this context it is important to know that the natural numbers can be axiomatized very naturally within relational algebra [1], so that places with multiple tokens can be modelled.

8 References

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

[2] Berghammer R., Gritzner T., Schmidt G.: Prototyping relational specifications using higher-order objects. In: Heering, J., Meinke, K., Möller, B., Nipkow, T. (eds.): Proc. Int. Workshop on Higher Order Algebra, Logic and Term Rewriting (HOA 93), Amsterdam, The Netherlands, Sept. 1993, LNCS 816, Springer, 56-75 (1994)

[3] Berghammer R., von Karger B.: Algorithms from relational specification. In: Brink C., Schmidt G., Albrecht R. (eds.): Relational methods in Computer Science, Supplement volume of Computing (to appear 1996)

[4] Berghammer R., Schmidt G.: RELVIEW – A computer system for the manipulation of relations. In: Nivat M., Rattray C., Rus T., Scollo G.

(eds.): Proc. 3rd Conf. on Algebraic Methodology and Software Technology (AMAST 93), University of Twente, The Netherlands, June 1993, Workshops in Computing, Springer, 405-406 (1993)

[5] Dijkstra E.W.: Hierarchical ordering of sequential processes. Acta Informatica 1, 115-138 (1971)

[6] Genrich H.J., Lautenbach K.: Synchronisationsgraphen. Acta Informatica 2, 143-161 (1973)

[7] Hack M.: Analysis of production schemata by Petri nets. TR-94, MIT-MAC (1972)

[8] Lautenbach K.: Liveness in Petri nets. Bericht 02.1/75-7-29, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin (1975)

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

[10] Olderog E.-R.: Nets, terms and formulas. Cambridge Tracts in Theoret. Comput. Sci., Cambridge University Press (1991)

[11] Petri C.A.: Non-sequential processes. Bericht GMD-ISF-77-5, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin (1977)

[12] Schmidt G., Ströhlein T.: Relations and graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993)

[13] Tarski A.: On the calculus of relations. J. Symbolic Logic 6, 73-89 (1941)