# Model Checking Based on Kronecker Algebra

Peter Kemper, Ralf Lübeck
Informatik IV
Universität Dortmund
D-44221 Dortmund, Germany
{kemper,luebeck}@ls4.informatik.uni-dortmund.de

**Abstract**

Reachability analysis is a general approach to analyze Petri nets, but the state space explosion often permits its application. In the field of performance analysis of stochastic Petri nets, modular representations of reachability graphs using Kronecker algebra have been successfully applied. This paper describes how a Kronecker representation of the reachability graph is employed for exploration of the reachability set and subsequent model checking of ordinary Petri nets and Computational Tree Logic. A new and space efficient representation of the reachability set is given. A non-trivial model of a production cell is exercised to demonstrate the usefulness of the approach. Extensions towards certain colored Petri nets are briefly sketched.

**Keywords:** Colored Petri Nets, Reachability Set, Reachability Graph, Model checking, CTL, Kronecker Algebra

## 1 Introduction

A rich variety of analysis techniques and tools have been developed for the analysis of Petri nets (PNs). Heiner et al. [22] apply a set of available tools on a mid-size Petri net model of a production cell. The production cell itself is recognized as a reference model to compare various modeling formalisms, Lewerentz and Lindner collect 18 different approaches in [32] . One can draw several conclusions from [22]: a) large models are necessarily build in a modular way, b) each PN technique has its specific advantages, but none is powerful enough to handle the general problem, and c) even if they are used in combination, there remain some open problems, especially for analysis methods considering the complete reachability set ($RS$). Therefore we add in this paper a further analysis method which is able to handle the $RS$ of this model. The method uses the modular structure of a net for a modular representation of the reachability graph based on Kronecker algebra. Since such representations closely resemble the structure of a model, they are named “structured”.

Structured representations have been successfully applied for a modular description of a continous time Markov chain (CTMC) in performance analysis. A CTMC can be understood as a reachability graph (RG) with additional timing information assigned to arcs connecting states. The close relationship between RG and the embedded CTMC of stochastic Petri nets is shown in [1]. Structured representations allow performance analysis of CTMCs with several millions of states on current hardware configuration, e.g. a Sparc station with 64 MB primary memory. Since model checking does not require the same numerical precision as performance analysis, the technique should be applicable for nets with much larger $RS$ cardinalities.

Structured representations roughly divide into two groups:

**Modular representations,** where a set of components act via synchronisation of transitions and where no aggregated representation of a component exists. Modular representations result from a "divide and conquer" approach: in the divide step, RGs of components are explored in isolation using conventional $RS$ exploration methods; in the conquer step, RGs of components are encoded as matrices, these matrices are combined via Kronecker operations to take into account synchronisation between transitions and $RS$ exploration is performed using these Kronecker matrix structures. This yields a set of matrices combined by Kronecker operations to describe the RG and a representation of $RS$, see [27, 28, 29] for work related to reachability and [9, 20, 21, 29, 34, 35] for analysis of the embedded CTMC for stochastic Petri nets, resp. stochastic automata networks.

**Hierarchical representations,** where a notion of a (detailed) component and an aggregated component exists. Consideration of aggregated components at a higher level allows to control the interaction between the detailed components of lower levels. In fact, modular representations are a special case of a two-level hierarchical representation with trivial high level description. Hierarchical representations typically provide block-structured matrices, where blocks are build on Kronecker terms similar to modular representations. Hierarchical representations were originally introduced for asynchronous communication only [7], but recent publications demonstrate their suitability for synchronous communication as well, see [8, 12]. Hierarchical representations are often more effective than modular representations, since they imply more knowledge of the interaction between components. However this structural information is not trivially obtained, hence it can either come from the modeller or result from fixing certain structural requirements as in [3, 10, 12]. Only recently, an automatic derivation of a two-level hierarchy has been proposed [8, 11].

In this paper we follow a modular representation which is more straightforward to be applied for model checking. Its application just requires to have a partition of places, such that the induced subnets have finite RGs if considered in isolation. Bounded nets, especially conservative nets, fulfill this requirement simply by providing appropriate capacities for places of a partition. Extension of the technique to the hierarchical concept is clear, but it exceeds the currently available implementation, additionally the approach described here reveals to be sufficient for the analysis of non-trivial models like the production cell of [22, 32].

The paper is organized as follows: Section 2 defines Petri nets with a composition operation which allows to obtain the desired structure for a modular representation of RG. In Section 3 we define Kronecker operations and a structured representation of the RG of a given PN. Section 4 characterizes $RS$ exploration methods exploiting the structured representation. We discuss three representations of $RS$, namely a permutation vector, a searchtree and a folded searchtree, of which the latter is a new approach. Section 5 illustrates how the structured representation is used for model checking. We consider the computational tree logic (CTL) as a language to specify formulas. Section 6 exercises a non-trivial example to demonstrate the benefits of the new algorithm. Section 7 briefly sketches extensions towards certain colored PNs. Section 8 summarizes the results.

## 2 Basic Definitions

We assume that the reader is familiar with PNs and the related basic concepts. For details about these fundamentals we refer to [33].

**Definition 2.1** *A Petri net is a 5 tuple $PN = (P, T, I^-, I^+, M_0)$, where*

- $P = \{p_1, \ldots, p_n\}$ *is a finite and non-empty set of places,*
- $T = \{t_1, \ldots, t_m\}$ *is a finite and non-empty set of transitions ($P \cap T = \emptyset$),*
- $I^-, I^+ : P \times T \to \mathbb{N}$ *are the backward and forward incidence functions, and*
- $M_0 : P \to \mathbb{N}$ *is the initial marking.*

The initial marking is a special case of a marking $M : P \to \mathbb{N}$. A marking $M$ can be interpreted as an integer (row) vector which includes per place $p$ one element which describes the number of tokens on place $p$.

${}^\bullet t = \{p \in P | I^-(p,t) > 0\}$ gives the set of input places for a transition $t$, and $t^\bullet = \{p \in P | I^+(p,t) > 0\}$ gives the set of output places. Analogously we define ${}^\bullet p = \{t \in T | I^+(p,t) > 0\}$, $p^\bullet = \{t \in T | I^-(p,t) > 0\}$. The notion can directly be extended to sets. In the sequel we consider connected nets, i.e. each place, transition has at least one incoming and one outgoing arc. Transition $t \in T$ is enabled in marking $M$, iff $\forall p \in P : M(p) \geq I^-(p,t)$. Any transition enabled in $M$ can fire, changing the marking of any $p \in P$ to marking $M'(p) = M(p) + I^+(p,t) - I^-(p,t)$. This will be indicated by $M[t\!> M'$, $M[t\!>$ denotes that $t$ is enabled in $M$, and $M[\!>$ describes the set of enabled transitions in $M$. Considering firing sequences $s \in T^*$ yields to the definition of the language $L(PN) = \{s \in T^* | M_0[s\!>\}$ and $RS(PN) = \{M | \exists s \in T^* : M_0[s\!> M\}$, which is the set of all reachable markings for $PN$. The reachability graph $RG(PN)$ (or RG for short) contains nodes for every $M \in RS(PN)$ and an arc $M \xrightarrow{t} M'$, if $M[t\!> M'$. RG can be represented by a $(RS(PN) \times RS(PN))$ incidence matrix $\mathbf{R}(M, M') = 1 \iff \exists t \in T : M[t\!> M'$ and 0 otherwise. If transition identities are not relevant, $\mathbf{R}$ can be stored as a Boolean matrix, otherwise $\mathbf{R}$ can be described as sum of transition specific matrices $\mathbf{R} = \sum_{t \in T} \mathbf{R}|_t$ or it has to be redefined as an integer or real valued matrix, which includes the required information, e.g. by numerical indices of involved transitions. We will focus on boolean matrices in the sequel.

Superposed PNs are PNs, where additionally a partition of the set of places is defined, such that they can be seen as a set of PNs which are synchronized by certain transitions.

**Definition 2.2** *A Superposed PN is a 7 tuple $SupPN = (P, T, I^-, I^+, M_0, \Pi, TS)$ where*
*$(P, T, I^-, I^+, M_0)$ is a PN,*
*$\Pi = \{P^0, \ldots, P^{N-1}\}$ is a partition of $P$,*

*$TS \subseteq T$ is the set of synchronized transitions.*
*Moreover $\Pi$ induces on $T \backslash TS$ a partition of transitions. Such a SupPN contains $N$ components $PN^i = (P^i, T^i, I^{i-}, I^{i+}, M_0^i)$ for $i \in IS := \{0, 1, \ldots, N-1\}$, where $T^i := {}^\bullet P^i \cup P^{i\bullet}$ and $I^{i-}, I^{i+}, M_0^i$ are the functions $I^-, I^+, M_0$ restricted to $P^i$, resp. $T^i$. Let $IC(t) = \{i \in IS | t \in T^i\}$ denote the set of involved components, i.e. transition $t$ is involved in $PN^i$. For $t \in T \backslash TS : |IC(t)| = 1$ and for $t \in TS : |IC(t)| \geq 2$.*

Alternatively superposition can be defined as a composition operation using fusion sets for transitions.

*SupPN*s are naturally amenable for a modular $RS$ analysis, since partition $\Pi$ induces components, which are PNs themselves. In consequence state spaces of components can be explored independently and in parallel by $RS$ exploration. The underlying assumption is that enabling of a synchronized transition depends only on the marking of component $i$ during generation of $RG(PN^i)$, resp. $RS(PN^i)$. In [14] Christensen et al. consider modular colored PNs with transition fusion, which is a similar concept for colored nets. They argue (in our terms) that component state spaces $RS(PN^i)$ can be infinite, while the $RS$ of the complete model is finite. In general we cannot cure this problem; nevertheless if such a critical component $i$ is not covered by P-invariants in isolation but covered by P-invariants within the *SupPN*, these global P-invariants are suitable to achieve finiteness of $RS(PN^i)$ by providing place capacities. P-invariants are similarly exploited in [27, 29] to improve efficiency of a numerical method for stochastic PNs by enforcing state spaces of components to obey place capacities deduced from global P-invariants. So in the following we assume that a *SupPN* is given for which finite component state spaces $RS(PN^i)$ are calculated via conventional $RS$ exploration.

# 3 Kronecker representation of RG

In this section, we recall known result with respect to the structured representation of $RG$, hence we rather describe the idea of the construction than formally prove its correctness, for details we refer to [20, 21, 27, 29, 28]. Structured representations are based on Kronecker algebra to combine incidence matrices $\mathbf{R}^i$ of reachability graphs of components $RG(PN^i)$. Kronecker algebra is sometimes named tensor algebra as well, however the former term seems to be more usual. We define Kronecker products according to [19], but only for square matrices, since only square matrices are relevant in our context.

**Definition 3.1** *Kronecker product, Kronecker sum*

*Let $A^0, \ldots, A^{N-1}$ be square matrices of dimension $(k^i \times k^i)$ for $i \in \{0, \ldots, N-1\}$, then their Kronecker product $A = \bigotimes_{i=0}^{N-1} A^i$ is defined by $a(x, y) = \prod_{i=0}^{N-1} a^i(x^i, y^i)$ where $x = \sum_{i=0}^{N-1} x^i \cdot g^i$ and $y = \sum_{i=0}^{N-1} y^i \cdot g^i$ with weights $g^0 = 1, g^i = k^{i-1} \cdot g^{i-1}$ for $i > 0$.*

*The Kronecker sum $B = \bigoplus_{i=0}^{N-1} A^i$ is then given by $\bigoplus_{i=0}^{N-1} A^i = \sum_{i=0}^{N-1} I_{l^i} \bigotimes A^i \bigotimes I_{r^i}$ where $I_{l^i}, I_{r^i}$ are identity matrices of dimension $l^i \times l^i$, resp. $r^i \times r^i$ with $r^i = \prod_{j=0}^{i-1} k^j$, $l^i = \prod_{j=i+1}^{N-1} k^j$ and $I(a, b) = 1$ iff $a = b$ and $0$ otherwise.*

A Kronecker product formalizes the operation of multiplying every matrix element of one matrix with all matrix elements of the other matrices; these products of matrix elements are arranged in lexicographical order in the resulting matrix, for more details see, e.g., [37]. In case of boolean matrices, multiplication is equivalent to logical AND and addition is equivalent to logical OR. Note that the index transformation is a linearization of the index vectors $(x^0,\ldots,x^{N-1}) \in \times_{i=0}^{N-1}\{0,\ldots,k^i-1\}$ into $x \in \{0,\ldots,(\prod_{i=0}^{N-1} k^i)-1\}$ and that the lexicographical order of the index vectors is compatible with the natural order of this mapping. This index computation is a mixed radix number representation, $x = mix(x^0,\ldots,x^{N-1}) = \sum_{i=0}^{N-1} x^i g^i$.

We aim for a matrix representation of the $(RS(PN) \times RS(PN))$ incidence matrix $\mathbf{R}$ of $RG(PN)$ for a given $SupPN$ where $\mathbf{R}(M_x,M_y) = 1 \iff \exists t \in T : M_x[t\!> M_y$ and $\mathbf{R}(M_x,M_y) = 0$ otherwise. We consider a component $PN^i$ in isolation, which implies that we relax the enabling conditions of its synchronised transitions $TS \cap T^i$. Hence the resulting $RS(PN^i)$ is a superset of the projection of $RS(PN)|_{P^i}$. The $RS(PN)$ of a superposed PN is consequently a subset of $PS = \times_{i\in IS} RS(PN^i)$. Let the index for component markings in $RS(PN^i)$ be arbitrary but fixed for any component $i$ as well as the index $IS$ for components themselves. With a slight misuse of notation we will use $RS(PN)$ to denote the cardinality of $RS(PN)$ as well as the set of reachable markings, if its meaning is clear. The same holds for $PS$. We obtain four ways to describe a marking $M$: 1) as a vector of markings $M_x \in \mathbb{N}_0^P$, 2) as a vector of component markings $(M_x^0,\ldots,M_x^{N-1}) \in \times_{i\in IS} RS(PN^i)$, 3) as a vector of component indices $(x^0,\ldots,x^{N-1}) \in \times_{i=0}^{N-1}\{0,\ldots,RS(PN^i)-1\}$ and 4) as an index $x = mix(x^0,\ldots,x^{N-1}) \in \{0,\ldots,PS-1\}$. The correspondence between these descriptions is determined and consistent, if we define the index for vectors of markings $M_x \in \mathbb{N}_0^P$ to be index $x$. Formally index $x \in \{0,\ldots,PS-1\}$ covers only a finite subset of $\mathbb{N}_0^P$, but this subset contains $RS(PN)$ and we will consider only elements of $PS$ in the sequel. Since the correspondence between indices and markings is fixed and $mix$ is a bijective function, we can use these descriptions interchangeably.

Clearly we can define incidence matrices $\mathbf{R}^i$ for any $RG(PN^i)$ corresponding to $RS(PN^i)$. A transition $t \in T$ is enabled at marking $M$, if $M(p) \geq I^-(p,t)$ for all $p \in {}^\bullet t$. At the level of component matrices $\mathbf{R}^i$, a state transition in the $SupPN$ due to the firing of a transition $t \in T$ as

$$(M_x^0,\ldots,M_x^{N-1})[t\!>(M_y^0,\ldots,M_y^{N-1}),$$

takes place iff $M_x^j = M_y^j$ for all components $j$ not involved with $t$, i.e. for $j \in IS \backslash IC(t)$, and $M_x^i[t\!> M_y^i$ for all involved components $i \in IC(t)$. In the incidence matrix $\mathbf{R}$, a corresponding entry $\mathbf{R}(x,y)$ with $x = (x^0,\ldots,x^{N-1})$ and $y = (y^0,\ldots,y^{N-1})$ has a value 1 iff $\forall i \in IC(t) :$ $\mathbf{R}^i|_t(x^i,y^i) = 1$ and $\forall i \in IS\backslash IC(t) : x^i = y^i$. Note that the latter behavior is that of an environment "neutral" to the state transition, which is exactly described by an identity matrix $I$ of appropriate dimension with $I(x,y) = 1$ iff $x = y$ and 0 otherwise. The former behavior is described by a projection of the component matrix $\mathbf{R}^i|_t$ with respect to $t$. Since multiplication for boolean matrices is defined as AND, we obtain a Kronecker representation of the overall $(PS \times PS)$ matrix $\bar{\mathbf{R}}$, which includes $\mathbf{R}$ as a submatrix:

$$\bar{\mathbf{R}} = \sum_{t \in T} \bigotimes_{i \in IS} \mathbf{R}_t^i \tag{1}$$

where $\mathbf{R}_t^i$ is an $(RS(PN^i) \times RS(PN^i))$ identity matrix if $t \notin T^i$ and otherwise it is the incidence matrix $\mathbf{R}^i|_t$ of $RG(PN^i)$ projected to $t$. The representation splits entries of $\mathbf{R}^i = \sum_{t \in T^i} \mathbf{R}^i|_t$ according to the transition whose firing it describes. $\bar{\mathbf{R}}$ allows to obtain the set of successor indices for a given $x \in \{0, \ldots, PS-1\} : succ(x) = \{y | \exists \bar{\mathbf{R}}(x,y) \neq 0\}$ and a set of predecessor indices for $y \in \{0, \ldots, PS-1\} : pred(y) = \{x | \exists \bar{\mathbf{R}}(x,y) \neq 0\}$. According to (1),

$$\begin{aligned} succ(x) = \quad & \{y| \quad (y^0, \ldots, y^{N-1}) = mix^{-1}(y), (x^0, \ldots, x^{N-1}) = mix^{-1}(x), \\ & \exists t \in T : \forall i \in IC(t) : \mathbf{R}_t^i(x,y) \neq 0, \forall i \in IS \backslash IC(t) : x^i = y^i\} \end{aligned} \tag{2}$$

$succ(x) \subseteq RS(PN)$ is guaranteed for $x \in RS(PN)$ but $pred(y)$ can contain indices not contained in $RS(PN)$ even if $y \in RS(PN)$. Note that the term on the right side consists of at most $N \cdot T$ matrices of relatively small dimensions which contain at most one "true" (non-zero) entry per row. The term of the right side allows to identify the transition, which fires, since we keep every transition separate. If this level of detail is not relevant for analysis, we can safely aggregate terms for transitions $t \in T^i \backslash TS$ which are local to a component $i$ as $\mathbf{R}_l^i = \sum_{t \in T^i \backslash TS} \mathbf{R}_t^i$ and exploit the definition of Kronecker sum to obtain the following alternative representation:

$$\bar{\mathbf{R}} = \bigoplus_{i \in IS} \mathbf{R}_l^i + \sum_{t \in TS} \bigotimes_{i \in IS} \mathbf{R}_t^i \tag{3}$$

Both representations allow to construct a set of successors of a vector of component markings $(M_x^0, \ldots, M_x^{N-1})$ by consideration of matrix rows $x^0, \ldots, x^{N-1}$ for non-zero entries at at most $N \cdot T$ component matrices. These rows yield column indices $y^0, \ldots, y^{N-1}$ which are subsequently combined into $(M_y^0, \ldots, M_y^{N-1})$. The terms on the right side of (1) and (3) represent $RG(PN)$, however $RS(PN)$ is usually a small subset of $PS$, which needs to be identified to restrict analysis algorithms on $RS(PN)$. This can take place by a so-called $RS$-permutation which reindexes the reachable subset of $PS$, [27, 29]. This is important in the context of model checking, if we want to store for each reachable marking the formulas it fulfills.

**Definition 3.2** *For a given $SupPN$, a bijective function $perm : \{0, 1, \ldots, PS-1\} \to \{0, 1, \ldots, PS-1\}$ is a $RS$-permutation if*

$\forall M_x \in PS : perm(x) < |RS| \iff M_x \in RS$

*and*

$\forall M_x, M_y \in RS : perm(x) < perm(y) \iff x < y.$

A $RS$-permutation reorders states according to their reachability and with respect to the lexicographical order of component state indices. This order is compatible with the order induced by the definition of Kronecker product on indices of entries in $\mathbf{R}$. Hence an equivalent notion for $x \in RS(PN)$ is $perm(x) = |\{y \in RS(PN) | y < x\}|$, which counts the reachable states which have an index smaller than $x$ wrt natural order in $\bar{\mathbf{R}}$. For unreachable states, analysis algorithms usually require only to recognize them as such, hence a $RS$-permutation does not need to provide a specific order on them.

We describe three representations for $RS$-permutation and access functions. Figure 1 illustrates three concepts for an artificial example with $N = 4$ components, $RS(PN)$ contains 9 markings, and $PS = 72$. Matrices below characterize the structured representation: column $RS$ gives values of $RS(PN^i)$, vectors of component markings are given in columns $M_{12}$, ..., $M_{69}$. Rows $mix$ and $perm$ provide the values for $(x^0, x^1, x^2, x^3)$. Note that the index of IS need not coincide in general with the order of components chosen for levels.

| IS | level | $RS$ | $g^i$ |
|---|---|---|---|
| 0 | 0 | 2 | 36 |
| 1 | 1 | 3 | 12 |
| 2 | 2 | 4 | 3 |
| 3 | 3 | 3 | 1 |

| | $M_{12}$ | $M_{13}$ | $M_{15}$ | $M_{21}$ | $M_{40}$ | $M_{60}$ | $M_{61}$ | $M_{64}$ | $M_{69}$ |
|---|---|---|---|---|---|---|---|---|---|
| $x^0$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| $x^1$ | 1 | 1 | 1 | 1 | 0 | 2 | 2 | 2 | 2 |
| $x^2$ | 0 | 0 | 1 | 3 | 2 | 0 | 0 | 1 | 3 |
| $x^3$ | 0 | 1 | 1 | 0 | 2 | 0 | 1 | 1 | 0 |
| $mix$ | 12 | 13 | 15 | 21 | 40 | 60 | 61 | 64 | 69 |
| $perm$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Permutation vector** According to [27, 29], we consider a permutation vector $v$ of length $RS(PN)$. $v$ enumerates indices $x$ of $M_x \in RS(PN)$ in natural order. Indices of unreachable states remain unrepresented. The value of $perm(x)$ for $x \in RS(PN)$ is given by the position of $x$ in $v$. Hence $i = perm(x)$ takes $O(log(RS(PN)))$ for a binary search to locate $x$ at position $i$ or to decide that $x \notin RS(PN)$. The inverse $x = perm^{-1}(i)$ is evaluated in constant time. The permutation vector is easy to implement, but has the problem of using $log_2(PS)$ bits per entry, which results to 2,4, or 8 bytes per entry in practice. If $PS \geq 2^{64}$, appropriate integer arithmetic is a problem. Fig. 1 a) illustrates a permutation vector for the example.

**Searchtree** Alternatively, following [15, 13] one obtains a tree type structure of depth $N$ where nodes at level $i \in \{0, \ldots, N-1\}$ provide the reachable fraction of $RS(PN^i)$ in $RS(PN)$ for fixed values of $(M^0, \ldots, M^{i-1})$. Note that the number of entries in each node at level $i$ can vary in $[1, RS(PN^i)]$ according to reachability. Empty nodes are omitted. The nodes at level $N-2$ point to vectors with entries $x^{N-1}$. Hence $(x^0, \ldots, x^{N-1})$ is found in $O(\sum_{i \in IS} log_2(RS(PN^i)))$ steps, using binary search at each node, while searching for the corresponding path in the tree. Its position according to $perm(x)$ is then obtained by the distance to the lowest index, e.g. by pointer arithmetic if the leaf vectors are stored on consecutive memory. An advantage of this representation is that it closely follows the modular structure and takes at the lowest level (which is usually the most expensive in terms of memory) only $log_2(RS(PN^{N-1}))$ bits to represent entries $x^{N-1}$. This often results in 1-2 Bytes per entry in an implementation. The searchtree avoids the integer arithmetic limitation of the permutation vector by representing vectors of component indices as paths of the tree.

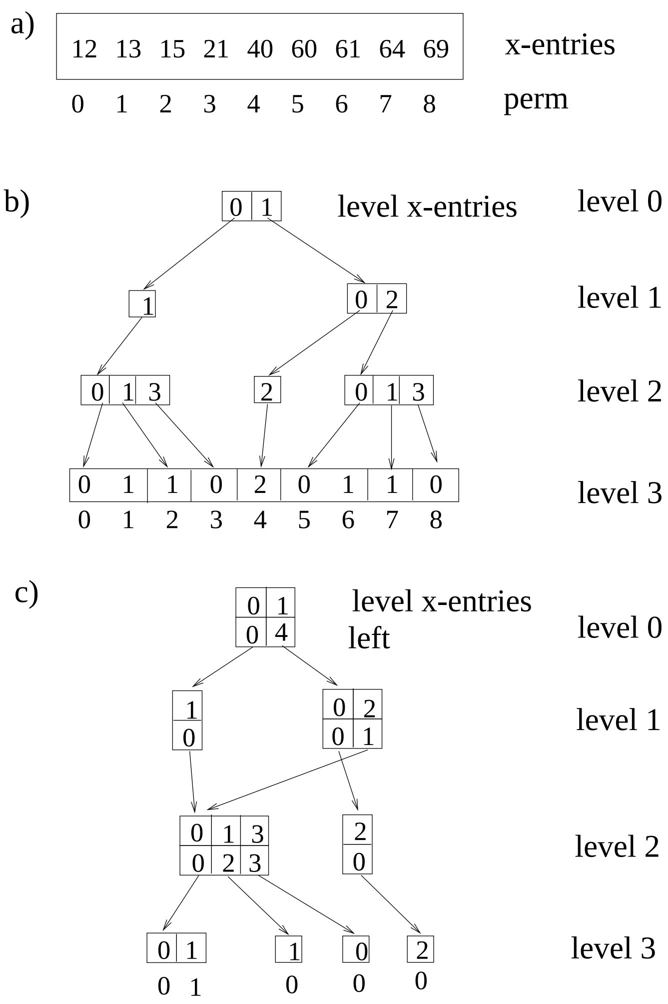


Figure 1: Illustration of data structures: a) permutation vector b) searchtree c) folded searchtree

**Folded searchtree** We consider a searchtree as above and define for each node $\nu^i$ at level $i$, $size(\nu^i, x^i)$ as the number of leaf entries in the subtree starting at $\nu^i$ and position $x^i$. So $\sum_{x^0 \in RS(PN^0)} size(\nu^0, x^0) = RS(PN)$. Let $left(\nu^i, x^i) = \sum_{y^i < x^i} size(\nu^i, y^i)$ denote the number of leaf entries in all subtrees starting at $\nu^i$ with smaller (left) entries than $x^i$. For a path along $(x^0, \ldots, x^{N-1})$ for an $x \in RS(PN)$ we can alternatively obtain $perm(x) = \sum_{i \in IS} left(\nu^i, x^i)$ by counting the number of reachable markings with smaller indices than $x$. Hence computing a path for $perm(x)$ takes $O(\sum_{i \in IS} log_2(RS(PN^i)))$, if we extend the searchtree by values for $left$. Additionally one can as well evaluate $x = perm^{-1}(z)$ by searching a path along $max_{y^i \in \nu^i}(left(\nu^i, y^i) \leq z - \sum_{j<i} left(\nu^j, x^j))$ with the same time complexity. An interesting observation is, that we need not distinguish between equal subtrees for the computation of $perm$ or $perm^{-1}$. Hence we use a folding operation similar to the one used in Binary decision diagrams (BDDs) to represent isomorphic subtrees just once [6]. The folding mechanism works bottom up, i.e. two leaf nodes are equal if they represent the same set of reachable states in $RS(PN^{N-1})$, two nodes of a level $i \in \{1, \ldots, N-2\}$ are equal if they represent the same set of reachable states in $RS(PN^i)$ and all of their subtrees

are equal. The latter is trivial to check, since the representation is unique and the procedure works bottom up, such that pointers need to be equal for equal subtrees, analogously to [6]. The folded searchtree offers the potential of significant savings in space requirements while computation times for $perm(x)$ remains similar to the ordinary searchtree. At the bottom level $left(\nu^{N-1}, x^{N-1})$ equals the position of $x^{N-1}$ in $\nu^{N-1}$, such that $left(\nu^{N-1}, x^{N-1})$ need not be explicitly stored. This is important since an ordinary searchtree gives the worst case scenario for a folded searchtree in terms of space complexity, and in this worst case the space for the bottom level typically dominates the space complexity of the data structure according to [15]. Furthermore the folded searchtree shows a high potential to employ further ideas known from BDDs.

# 4 RS-exploration based on Kronecker Algebra

Generation of $RS(PN)$ for a given $PN$ by conventional algorithms is typically a Breadth-First-Search or Depth-First-Search exploration of the $RG(PN)$, using a stack $S$ for reached but not yet explored markings and a set $RS$ to accumulate elements of $RS(PN)$. Markings are described in $RS$ and $S$ as vectors $M \in \mathbb{N}_0^P$, which typically results in a tree-type data structure for $RS$ [13, 15]. The crux is the space used for $RS$ and $S$ and the time complexity of member and insert operations on $RS$. We employ conventional $RS$ exploration for generation of $RS(PN^i)$ and matrices $\mathbf{R}_t^i$, $\mathbf{R}_l^i$ in (3). The following algorithm sketches the procedure:

Algorithm 1: Conventional $RS$ exploration, $M_x \in \mathbb{N}_0^P$
Init: $RS = \{M_0\}$, $S = \{M_0\}$
begin
  while $S$ is not empty
      take $M_x$ out of $S$
      foreach $t \in T : M_x[t> M_y$
          if $M_y \notin RS$
          then insert $M_y$ in $RS$ and $S$
  end

A structured representation provides a description of the incidence matrix $\mathbf{R}$ of the $RG(PN)$, but this matrix is contained in the $(PS \times PS)$ matrix $\bar{\mathbf{R}}$. Hence we reformulate the problem of computing $RS(PN)$ in terms of indices $x, y \in \{0, \ldots, PS-1\}$ by: compute the transitive closure of the reflexive relation $reach \subseteq PS \times PS$ where $reach(0,0)$ and $reach(x,y)$ if $\bar{\mathbf{R}}(x,y) \neq 0$. It is easily verified that the transitive closure provides $RS(PN)$. However computation of $RS(PN)$ based on a structured representation as in (3) can profit from several facts. A transition $t$ has only $IC(t) \leq {}^\bullet t$ enabling conditions, for $t \in T \backslash TS$ it is just one. Additionally several transitions are represented in matrices $\mathbf{R}_l^i$ which increases the chance to observe an enabled transition for an $x^i$. Furthermore $mix$ gives a perfect hash function. These advantages were first exploited in [28] for $RS$ exploration using structured representations and bitstate hashing. A straightforward integration of the computation of successor markings from a structured representation into the conventional algorithm yields:

Algorithm 2: $RS$ exploration based on structured representation, $x \in \{0,\ldots,PS-1\}$
Init: $RS = \{0\}$, $S = \{0\}$
begin
  while $S$ is not empty
    take $x$ out of $S$
    decode $x$ into $(x^0,\ldots,x^{N-1})$ by $mix^{-1}$
    foreach component $i \in IS$    // compute $succ(x)$ for local transitions
      foreach $\mathbf{R}_l^i(x^i, y^i) \neq 0$
        $y = x + (y^i - x^i) \cdot g^i$
        if $y \notin RS$
        then insert $y$ in $RS$ and $S$
    foreach $t \in TS$    // compute $succ(x)$ for synchronized transitions
      if $\forall i \in IC(t) : \exists \mathbf{R}_t^i(x^i, y^i) \neq 0$
(∗)     then foreach combination of elements $\mathbf{R}_t^i(x^i, y^i) \neq 0$ over IC(t)
        $y = x + \sum_{i \in IC(t)} (y^i - x^i) \cdot g^i$
        if $y \notin RS$
        then insert $y$ in $RS$ and $S$
end

$RS$ is represented by a bit vector of length $PS$ such that member and insert operations are performed in constant time, $S$ contains indices $x \in \{0,\ldots,PS-1\}$. We assume that the index of the initial marking is 0. In line $(*)$ we can at most observe one combination since each $\mathbf{R}_t^i$ contains at most one nonzero entry per row, i.e. $t$ is enabled in $x^i$ or not. However if the modeling formalism allows for probabilistic output bags or immediate transitions in case of stochastic Petri nets, more than one entry per row can occur, such that the combinations in $(*)$ describe all possible ways to fire a transition $t$. We kept this formulation for consistency with [28].

A limitation of this approach is the size of the hashtable $PS$, which can be prohibitive. Furthermore the encoding using $mix$ works efficiently for $PS < 2^{32}$ , it can be extended to $2^{64}$ if integer arithmetic with 64 bit is available, but for the price of additional memory to represent the stack $S$. In the following we discuss an extension of this technique to avoid relief problems. Note that:

- conventional $RS$ exploration is efficient for $RS$ several tenthousands of markings, such that it is suitable to explore component state spaces $RS(PN^i)$,
- $RS$ exploration based on the structured representation using $mix$ as a hashfunction and a hashtable of dimension $PS$, works well if $PS \leq 2^{32}$ and space for bitstate hashing ($PS$ bit) remains within primary memory (usually less than $2^{32}$ bit, so memory limits are more restrictive).

For $PN$ where $PS$ exceeds these limitations, one can select a different partition into components or employ net transformations based on P-invariants to remain within these limits as proposed in [29, 27]. This clearly works well only for certain $PNs$. Alternatively one can employ a tree-type data structure for $RS$ and describe markings $M_x$ as $(x^0,\ldots,x^{N-1})$ on $S$

and $RS$ in order to avoid encoding with $mix$ as proposed in [15]. This is a general approach, but for the price of an increased time complexity for member and insert operations on $RS$.

In the following we discuss a further approach, where $RS$ exploration based on a structured representation is used as a building block for an $RS$ exploration considering bunches of components. We aim for those models, where $RS(PN) << PS(PN)$. This effect clearly results from synchronisation between components; the idea is to consider a subset of $k$ components $B^i = \{PN^i, \ldots, PN^{i+k-1}\}$ in isolation, such that the components share a lot of synchronized transitions and $PS(B^i)$ fits into the limitations discussed above. The expectation is to observe $RS(B^i) << PS(B^i)$ after a $RS$ exploration based on a structured representation for $B^i$. In that case it makes sense to redefine $mix$ for the complete $PN$ to concern for the reachable fraction of $PS(B^i)$ only.

For the general case, we define a partition of components in $IS$ into $M$ non-empty subsets (bunches) $\Pi_b = \{B^0, B^1, \ldots, B^{M-1}\}$ with $M \leq N$. Clearly each bunch of components is a SupPN of its own and has a structured representation as in (1) and (3). Let $RS(B^i)$ result from a $RS$ exploration performed for $B^i$. Then we can define a new mixed radix number representation restricted to $RS(B^i)$, $mix_b : \times_{i=0}^{M-1}\{0, \ldots, RS(B^i)-1\} \longrightarrow \{0, \ldots, PS_b - 1\}$ with $PS_b = \prod_{i=0}^{M-1} RS(B^i)$. $mix_b(x_b^0, \ldots, x_b^{M-1}) = \sum_{i=0}^{M-1} x_b^i \cdot g_b^i$ with $g_b^i = \prod_{j=0}^{i-1} RS(B^j)$ if $i > 0$ and $g_b^0 = 1$. Since the structured representation $\bar{\mathbf{R}}$ remains unchanged, an algorithm which considers marking representations based on $mix_b$ must perform a base change to compute successor markings. Note that for a bunch $i$, an index $x_b^i = perm^i(mix^i(x_i^0, \ldots, x_i^{N_i-1}))$ is the result of a $RS$ permutation from the vector of indices of components contained in $i$, where the structured representation of this bunch provides functions $mix^i$ and $perm^i$ (defined as for structured representations before). Let $N_i$ denote the number of components in bunch $i$. Let $TS_b$ denote the subset of $TS$, whose transitions synchronize components of different bunches, and $IC_b(t)$ denote the set of bunches which contain at least one component $j \in IC(t)$.

Algorithm 3: $RS$ exploration using bunches, $x \in \{0, \ldots, PS_b - 1\}$
Init: $RS = \{0\}$, S={0}
begin
  while S is not empty
    take $x$ out of S
    decode $x$ into $(x_b^0, \ldots, x_b^{M-1})$ by $mix_b^{-1}$
    foreach bunch $B^i$ // compute $succ(x)$ for local transitions
      $x_i = (perm^i)^{-1}(x_b^i)$
      compute $succ(x_i)$ as in Algorithm 2 for $t \in T^i \backslash TS_b$ and $B^i$
      foreach $y_i \in succ(x_i)$
        $y = x + (perm^i(mix^i(y_i^0, \ldots, y_i^{N_i-1})) - x_i) \cdot g_b^i$
        if $y \notin RS$
        then insert $y$ in $RS$ and $S$
    foreach $t \in TS_b$ // compute $succ(x)$ for synchronized transitions
      foreach bunch $B^i \in IC_b(t)$
        $x_i = (perm^i)^{-1}(x_b^i)$
        compute $succ(x_i)$ as in Algorithm 2 for synchronized $t$ and $B^i$

if $succ(x_i) \neq \emptyset$ for all $B^i \in IC_b(t)$

(*) then foreach combination of elements in $succ(x_i)$ over $IC_b(t)$

$y = x + \sum_{i \in IC_b(t)} (perm^i(mix^i(y_i^0, \ldots, y_i^{N_i-1})) - x_i) \cdot g_b^i$

if $y \notin RS$

then insert $y$ in $RS$ and $S$

end

The notion of "$succ(x_i)$ as in Algorithm 2" is an abbreviation to retain readability, otherwise one needs to distinguish component indices with respect to bunch indices $x_i$. Similar to Algorithm 2, $RS$ is represented by a bit vector of length $PS_b$. $S$ is a stack containing $x \in \{0, \ldots, PS_b - 1\}$. Hence we inherit limitations for $PS_b$ as for $PS$ in Algorithm 2, but if bunches are build such that they internalize many synchronized transitions, we expect $PS_b << PS$.

Encoding and decoding for functions $mix_b$, $mix^i$ and $perm^i$ require additional computations, from which the time complexity of $perm^i$, resp. $(perm^i)^{-1}$ depends on the chosen data structure as described in Section 3:

| data structure | $perm^i$ | $(perm^i)^{-1}$ |
|---|---|---|
| permutation vector | $O(log(RS(B^i)))$ | $O(1)$ |
| searchtree and folded searchtree | $O(\sum_{j \in B^i} log(RS(PN^j)))$ | $O(\sum_{j \in B^i} log(RS(PN^j)))$ |

For the searchtree as such, $(perm^i)^{-1}$ is rather expensive, the given time complexity is obtained by storing appropriate $left$ values at nodes as for the folded searchtree. This yields equal time complexities on both kinds of trees. Several aspects for improvements are:

- As shown in [9], searchtrees offer the chance to reduce the time complexity by reusing intermediate results and considering elements in specific orders.
- A possible improvement results from the selection of bunches such that many synchronized transitions become local, which allows to exploit locality similar to the "improved algorithm" for structured representations described in [28].
- A generalization of the presented approach here to recursively consider bunches of bunches etc. is straightforward; an interesting point is that one needs only representations for $perm$ of one set of bunches. If for a bunch $i$, $RG(B^i)$ is relatively small, one can even transform the structured representation of $B^i$ into a set of sparse-matrices as for components before.

Further investigations of these aspects are subject to ongoing research.

In summary, if we build bunches of components, such that their cross product remains within memory limitations, one can still profit from $RS$ exploration based on the structured representation. The whole approach makes sense only if the $RS$ exploration for bunches is extremely fast, since it is applied $M$ times and just used to obtain a sufficiently small value of $PS_b$ for bit state hashing wrt $RS$ and hash function $mix_b$.

# 5 Model Checking

*Model checking* [18, 16] is a verification technique to determine whether a system possesses a desired property. In our context a system is specified as a Petri net and a property is expressed as a temporal logic formula [31, 36]. The model checking problem for a given $PN$ and a formula $f$ is to decide whether $M_0 \models f$ or not. A lot of model checkers have been developed, e.g. Prod [38], PEP [5], SMV [30], Spin [24] among others; see [18] for an overview. In the following we consider *computational tree logic* to express properties.

## 5.1 Computational Tree Logic (CTL)

CTL, e.g. in [2, 17], is a specification language for a propositional, branching-time temporal logic. We adopt the formal syntax of CTL from [17]. The underlying set of *atomic propositions* consists of expressions considering markings, e.g. $f = (M(p) > M(p'))$ is such an atomic proposition for $p, p' \in P$ of $PN$. Every atomic proposition is a CTL formula; if $f_1$ and $f_2$ are CTL formulas, then so are

(a) $(f_1)$, $\neg\, f_1$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $f_1 \Rightarrow f_2$, $f_1 \Leftrightarrow f_2$ *(with their usual meanings)*

(b) $AF\ f_1$, $EF\ f_1$, $AG\ f_1$, $EG\ f_1$, $AX\ f_1$, $EX\ f_1$, $A[f_1\ U\ f_2]$, $E[f_1\ U\ f_2]$.

The temporal operators are *nexttime* ($X$), *globally* ($G$), *sometimes* (or *finally*) ($F$), and *until* ($U$). $A$ is the *all path quantifier* and $E$ the *exists path quantifier*. $AX\ f$ ($EX\ f$) intuitively means, that $f$ holds in every (in some) direct successor $y \in succ(x)$ of marking $x$, resp. $M_x$.

$A[f_1\ U\ f_2]$ intuitively means, that *every path* has an initial prefix $\sigma$, such that $f_1$ holds at all markings of $\sigma$ apart from the last one and $f_2$ holds for the last marking of $\sigma$ ($f_1$ holds, until $f_2$ holds). $E[f_1\ U\ f_2]$ is analogously defined for *some path.*

Common abbreviations of CTL formulas, their definition and intuitive meanings are as follows:

| | |
|---|---|
| $AF\ f \equiv A[\text{true}\ U\ f]$ | $f$ is *inevitable*; *finally* $f$ |
| $EF\ f \equiv E[\text{true}\ U\ f]$ | $f$ *potentially* holds; $f$ is *possible* |
| $AG\ f \equiv \neg EF(\neg f)$ | $f$ is *invariant*; *always* $f$ |
| $EG\ f \equiv \neg AF(\neg f)$ | $f$ *potentially* holds *always* |

## 5.2 A classical CTL model checker combined with a structured representation

We apply classical model checking algorithms for CTL [17] and Petri nets, but use a structured representation for $RG$ and restrict the algorithms to $RS(PN)$, i.e. we employ a $RS$ permutation if $RS << PS$.

A CTL formula $f$ is evaluated in a bottom up way: $f$ is represented as a parse tree and the subformulas are proven using a postorder traversal. Thus one starts with an atomic proposition $f_1$ and evaluates for all $x \in RS(PN)$ which is straightforward. This yields a bit vector of length $RS(PN)$) which represents $f_1(x) = true$ by an entry with value 1 at position $x$. The bit vector of $f_1$ is used in the evaluation of the parent formula, in this manner one proceeds until $f$ is proven. Combinations of subformulas build on boolean operators as in (a) are trivial to evaluate by corresponding boolean operations on bit vectors of length $RS(PN)$). A subformula which consists of temporal operators (which will be combined with a path quantifier) depends on a sequence of markings, which need to be considered to evaluate the subformula. Some operators require to check the successors $y \in succ(x)$: *AX*, *EX*, others require to procede through $RG$ forward (*AU*, *AF*, *EG*) or backward (*EU*, *EF*, *AG*). Thus the structured representations based on Kronecker algebra must provide elements of $succ(x)$ resp. $pred(x)$ for $x \in RS(PN)$.

Efficiency of the model checker algorithm requires that every state $x$ is evaluated only once whether $x$ satisfies a (sub)formula $f$ or not. We do not describe algorithms for all operators, since they are very similar. We select $f = A[f_1\ U\ f_2]$ as an example since it covers most aspects. Let subformulas $f_1$ and $f_2$ be already evaluated and represented by two bit vectors (named $f_1$ and $f_2$) of length $RS(PN)$, with $f_i(x) =$ true if $x \models f_i$ for $i \in \{1, 2\}$. Index $x$ corresponds to a marking. The computation destroys values of $f_2$ to represent the result $f$ by the bit vector used for $f_2$ at termination. Note that initial values of $f_2$ are already correct for $x \models f$ if $f_2(x) =$ true. Hence the algorithm focuses at $x \in RS(PN)$ with $f_2(x) =$ false. A bit vector *done* of length $RS(PN)$ is employed to store for which $x$, a recursion *au(x)* has already started. Hence each $x$ causes at most once a recursion and *done* obtains initial values of $f_2$. The following algorithm is taken from [17] and modified for the context of a structured representation with $RS$ permutation.

procedure $eval\_au(f_1, f_2)$
begin
    Init: $\forall x \in RS(PN) : done(x) = f_2(x)$
    foreach $x \in RS(PN)$ and not $done(x)$
        $au(x)$
end
procedure $au(x)$
begin
    if $done(x)$ then return $f_2(x)$
    $done(x) =$ true
    if $f_2(x)$ then return true
    if $\neg f_1(x)$ then return false
    foreach $y \in perm(succ(perm^{-1}(x)))$
        if $\neg au(y)$ then return false
    $f_2(x) =$ true
    return true
end

Several improvements on the data structure representing a CTL formula are conceivable. Here are three ad hoc improvements:

**Recycle result vectors:** Transform the parse tree of a formula $f$ to a folded parse tree, which is a directed graph built from the parse tree, where identical nodes represent the same subformulas nested in the formula $f$ on different places. Hence every node includes a counter which indicates how often the subformula is used in $f$. A postorder traversal decrements the counter and "deletes" the node only if the counter is zero. An advantage of this representation is that it reduces the number of evaluation steps for $f$ for the price of a temporary but additional result vector. For example to evaluate the formula $f = EG(\alpha \Rightarrow (\beta \wedge AF\alpha))$ the bit vector for the inner most subformula $\alpha$ can be simply reused.

**State formula evaluation:** There are two types of formulas: *state formulas* representing properties, that are true or false at one point in time, and *path formulas* represent properties about a sequence of states. A state formula does not contain any temporal operator nor a path quantifier. Thus it depends only on one state whether the state satisfies the formula. So a state formula can be completely evaluated in one step.

**Combined path formula evaluation:** As mentioned above some path operations are implemented with a forward and others with a backward traversal algorithm. One improvement is to minimize the number of $RG$ traversals to evaluate a complete CTL formula. Hence consider several quantified temporal operators and combine their traversals into one optimized. Some interesting operator combinations are nested ones, like $A[AF(f_1)\ U\ f_2]$ and some, which are concatenated by boolean operators, like $AF(f_1) \wedge EG(f_2)$.

## 5.3 Classical PN analysis

A classical Petri net analysis [33] includes to prove properties like *boundedness*, *liveness*, and whether the initial marking $M_0$ is a *home state*.

A structured representation requires finite $RG(PN^i)$ for all components. In that case the reachability graph $RG(PN)$ of the complete Petri net is finite by construction. Thus boundedness can be deduced.

To decide liveness we use a variant of Tarjan's algorithm. The idea is to recognize strongly connected components in $RG(PN)$ and analyze them for liveness. Based on strongly connected components, it is simple to decide whether $M_0$ is a home state: $M_0$ is a home state, if the $RG$ is strongly connected.

Tarjan's algorithm is a DFS-algorithm on $RG(PN)$ which can be combined with a structured representation like the DFS-algorithm to evaluate $A[f_1\ U\ f_2]$.

## 5.4 Exploiting locality

If a subformula $f_s$ of a state formula only consists of places of a single component $i$, it can be proven locally. This means, that we can evaluate the subformula while we consider only $RG(PN^i)$ of the involved component $i$, because its environment cannot influence the result.

The resulting boolean vector of dimension $RS(PN^i)$ can then be projected to a boolean vector of dimension $RS(PN)$ using *mix* and *perm*. But how does this projection work? Let us consider the example illustrated in figure 1. If the state $x_2^1$ of component $PN^1$ holds for the formula $f_s$ then this result is stored in the bit vector $f_s$: $f_s(x_2^1) = \text{true}$. In figure 1 b) this state is represented at the right node $\nu^1$ at level 1 with index 2. Obviously the resulting value for $x_2^1$ holds for all elements in the subtree starting at $(\nu^1, x_2^1)$. Thus $f_s$ holds for an interval of $RS(PN)$, i.e. for global states 5, ..., 8 (in *perm* representation) resp. 60, 61, 64, and 69 (in *mix* representation), which complies to markings $M_{60}$, $M_{61}$, $M_{64}$, and $M_{69}$ of $RS(PN)$ (see table on page 7). The advantage is that truth values can be assigned to all elements of the interval without further evaluation.

# 6 Application Example

The production cell of [32] has been subject to modeling and analysis by a variety of tools and is known to be non-trivial. The production cell model originates from an existing production cell in an industrial setting, which physically consists of 5 components: an elevating rotary table, a rotable robot with two extendable arms, and two conveyor belts to feed the cell with raw materials and to deposit its products. The production cell performs transportation and processing of metal plates in a pipeline. It is a system open to an environment, which is assumed to produce raw plates and to consume processed plates readily. The feeding conveyor belt (feedbelt) transports metal plates to the elevating table, the table lifts plates for the robot, the robot takes and inserts plates into the press and takes them after pressing from the press onto the second conveyor belt (depositbelt). Originally plates leave the system via the depositbelt. Heiner et al. [22, 23] have modelled this production cell in great detail and in several variations in their PN editor PED. They evaluated several available Petri net analysis tools with it. In PED, refinement is used to organize a model of this size, however the dynamic behaviour of the model is not defined unless all refined subnets are available in full detail. This kind of hierarchy is very common for modeling purposes, but only helpful to obtain a structured representation in a semi-automated way. We consider two configurations of this model: a) as an open system in an environment which permanently provides raw plates and permanently waits to consume processed plates with a limit of at most 5 plates in the cell, and b) a closed system, where a crane is installed to put plates from the depositbelt onto the feeding belt, such that the number of plates within the system is constant. The number of plates in the closed model is set to 5. We start from a flat PN and manually integrate a partition into components. However a majority of this work could be derived automatically from the grouping of places in subnets due to refinement. This results into a partition according to the model structure itself, few places are left which are used to organize communication between components in the modeling style proposed by Heiner. We integrated these places into adjacent components. Consequently $RS$ cardinalities for our components are slightly different to those reported in [23], but the complete $RS$ is identical. For the open model we integrate the consumer component into the feedbelt component and the producer component into the despositbelt component due its marginal size (2 places each). From [22, 23] it is known that both nets are live and 1-bounded. The nets and partitions show the following characteristics (P = number of places, T = number of transitions, arcs = number of arcs, IS = number of components, TS = number synchronized transitions, and RS(PN) = number of states):

| model | P | T | arcs | IS | TS | RS(PN) |
|---|---|---|---|---|---|---|
| open | 198 | 176 | 730 | 5 | 10 | 2,776,936 |
| closed | 231 | 202 | 864 | 6 | 14 | 1,657,242 |

The structured representation is characterized in the table below, for each component of the open and closed model, it gives the number of places (P), transitions (TL) local to a component, synchronized transitions (TS) the component is involved with, the size of $RS(PN^i)$, and the space for matrices in the representation according to (3). Space is measured as the number of nonzero entries in incidence matrices of the components.

| | open | | | | | closed | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| component | P | TL | TS | RS | space | P | TL | TS | RS | space |
| $RS(crane)$ | - | - | - | - | - | 37 | 26 | 4 | 30 | 30 |
| $RS(feedbelt)$ | 20 | 14 | 2 | 104 | 248 | 18 | 12 | 4 | 192 | 544 |
| $RS(table)$ | 26 | 18 | 4 | 80 | 156 | 26 | 18 | 4 | 80 | 156 |
| $RS(robot)$ | 108 | 104 | 8 | 368 | 568 | 108 | 104 | 8 | 368 | 568 |
| $RS(press)$ | 24 | 16 | 4 | 256 | 736 | 24 | 16 | 4 | 256 | 736 |
| $RS(depositbelt)$ | 20 | 14 | 2 | 120 | 288 | 18 | 12 | 4 | 192 | 544 |

Obviously $RS$ cardinalities of components are very moderate, and their computation is performed in less than 2 s for all components. All computation times are wall clock times in seconds observed on a Sun Ultrasparc 1 with 160 MB primary memory and 143 MHz CPU.

However, $PS = 8.33 \cdot 10^{12}$ for the closed model and $PS = 9.41 \cdot 10^{10}$ for the open model such that Algorithm 2 fails due to memory limitations. One can build two bunches $B^0$ and $B^1$ from the components of the closed model: $B^0$ consists of table, robot and press, while $B^1$ covers feedbelt, crane and depositbelt. $RS$ exploration for $B^0$ takes about 2 seconds to generate $RS(B^0) = 22204$ and $RS(B^1) = 18336$. Since $PS_b = 4.07 \cdot 10^8$, one can apply Algorithm 3 on this structure, which generates $RS(PN)$ in 556 s.

For the different data structures of $perm$ we observe that the vector uses 13 257 936 bytes due to 8 bytes per index, the searchtree uses 5 774 642 bytes (8 743 446 bytes if extended with $left$-values) and the folded searchtree uses just 9270 bytes. The folded searchtree consists of 6 levels in the order crane, depositbelt, feedbelt, press, table, robot with very few nodes, i.e.

| level | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| nodes | 1 | 4 | 12 | 14 | 35 | 22 |

Obviously $RS(B^0)$ and $RS(B^1)$ are sufficiently small to represent them conventionally, i.e. one can alternatively restructure the closed model into two components and explore $RS(B^0)$, $RS(B^1)$ conventionally. Since $PS_b = 4.07 \cdot 10^8$ is sufficiently small, one can then directly apply Algorithm 2. In this case, generation times for $RS(PN)$ are clearly dominated by the generation times for component reachability sets.

For the open model we consider bunches $B^0$ with feedbelt and table, $B^1$ with robot, press, and depositbelt. Generation of $RS(B^0) = 744$, $RS(B^1) = 39566$ takes 2 s, while the generation of $RS(PN) = 2\,776\,936$ takes 809 s. Note that $RS(PN)$ remained an open problem in [23]. The *perm* representations take 22 215 488 bytes for the vector, 6 642 576 bytes for the searchtree (8 503 152 bytes if extended with *left*-values) and 3168 bytes for the folded searchtree. The number of nodes per level in the folded searchtree are for an order depositbelt, feedbelt, press, table, robot:

| level | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| nodes | 1 | 2 | 4 | 14 | 12 |

For both models we can check CTL formulas, e.g. of the following kind taken from [22]: if a robot arm is loaded ($\alpha$), its magnet is not deactivated ($\beta$) until the robot is in its unloading position ($\gamma$); formally

$$AG(\alpha \rightarrow A[not(\beta)U\gamma])$$

with $\alpha = (arm1\_mag\_on > 0) \wedge (arm1\_pickup\_angle > 0) \wedge (arm1\_pickup\_ext > 0)$, $\beta = (arm1\_mag\_off > 0)$ and $\gamma = (arm1\_release\_angle > 0) \wedge (arm1\_release\_ext > 0)$. An interesting observation is, that some of the propositions in this formula can be evaluated locally in $RS(robot)$; these are $\alpha$, $not(\beta)$, and $\gamma$, such that the resulting boolean vectors of dimension $RS(robot)$ can then be projected to a boolean vector of dimension $RS(PN)$ using *mix* and *perm*. Evaluation of the above formula takes 268 (302) seconds for the closed (open) model.

Additionally we implemented the classical algorithm to decide liveness for a PN as a variation of Tarjan's algorithm. Deciding liveness takes about 318 (460) s for the closed (open) model, which reveals that $RG(PN)$ is strongly connected and $M_0$ is a home state, once $RS(PN)$ has been explored.

# 7 Extension towards colored Petri nets

We assume that the reader is familiar with colored PNs (CPNs) and the related basic concepts [26]. We first consider CPNs where firing modes (colors) of transitions are explicitly enumerated, such that a static unfolding is straightforward.

**Definition 7.1** *A colored Petri net is a 7 tuple $CPN = (P, T, C, I^-, I^+, G, M^0)$, where $P$ is a finite and non-empty set of places, $T$ is a finite and non-empty set of transitions ($P \cap T = \emptyset$), $C$ is a color function mapping elements from $P \cup T$ into a finite and non-empty set of colors $W$ ($W \cap T = W \cap P = \emptyset$), $I^-, I^+ : P \times T \rightarrow [C(T) \rightarrow C(P)_{MS}]$ are the backward and forward incidence functions, where $C(P)_{MS}$ is the set of multisets over $C(P)$, $G : T \rightarrow [C(T) \times M \rightarrow \mathbb{B}]$ is the transition guard function ($\mathbb{B}$ denotes boolean values), and $M^0 : P \rightarrow C(P)_{MS}$ is the initial marking.*

The class of CPNs we use here includes guards as a general mechanism to restrict the firing of transitions. Transition $t \in T$ is enabled for color $c_t \in C(t)$ at marking $M$, iff

- $\forall p \in P$, $\forall c_p \in C(p)$: $M(p)(c_p) \geq I^-(p,t)(c_t)(c_p)$ and

- $G(t)(c_t, M) = true$.

If an enabled transition $t$ fires for $c_t$ at marking $M$, then a new marking is computed by

$$M'(p)(c_p) = M(p)(c_p) + I^{+}(p,t)(c_t)(c_p) - I^{-}(p,t)(c_t)(c_p)$$

for all $p \in P$ and $c_p \in C(p)$. A marking $M$ can be interpreted as an integer (row) vector which includes per place-color pair $(p, c_p)$ one element which describes the number of color $c_p$ tokens on place $p$. We use the short hand notation $M[t, c_t{>}\, M'$ if $t$ is enabled for color $c_t$ at $M$ and the firing yields successor marking $M'$.

$RS(CPN)$ is the set of all reachable markings for CPN. $RG(CPN)$ is the reachability graph containing nodes for every $M \in RS(CPN)$ and an arc $M \rightarrow M'$, if $M[t, c_t{>}\, M'$ for some $t \in T$ and $c_t \in C(t)$. If subsequent analysis methods require information about the cause of a state transition, arcs can be labeled with the corresponding transition-color pair.

A superposed CPN additionally provides a partition $\Pi_C = \{P^0, \ldots, P^{N-1}\}$ of places and a set of synchronized transitions $TS_C \subseteq T$ as for $SupPN$s. For $t \in TS_C$ we require that $G(t)(c_t, M) = \wedge_{i=0}^{N-1} G(t)(c_t, M^i)$ are formulated as a conjunction of guards for components. This is common for superposition of CPNs by transition fusion, see e.g. [25, 14]. A structured representation analogous to (1) and (3) is:

$$\bar{\mathbf{R}}_C = \sum_{t,c_t \in T,C} \bigotimes_{i=0}^{N-1} \mathbf{R}^i_{t,c_t} = \sum_{t,c_t \in TS_C,C} \bigotimes_{i=0}^{N-1} \mathbf{R}^i_{t,c_t} + \bigoplus_{i=0}^{N-1} \mathbf{R}^i_l \qquad (4)$$

where

$$\mathbf{R}^i_{t,c_t}(x^i, y^i) = \begin{cases} 1 & \text{if } t \in T^i \text{ and } M^i_x(p)(c_p) \geq I^{-}(p,t)(c_t)(c_p) \text{ for all } p \in P^i, c_p \in C(p) \\ & \text{ and } G(t)(c_t, M^i) = true \\ 1 & \text{if } t \notin T^i \text{ and } x^i = y^i \text{ and } G(t)(c_t, M^i) = true \\ 0 & \text{otherwise} \end{cases}$$

$\mathbf{R}^i_l = \sum_{t \in X} \mathbf{R}^i_{t,c_t}$ where $X \subseteq T^i \backslash TS_C$ denotes the subset of transitions local to $i$ where $G(t)(c_t, M) = G(t)(c_t, M^i)$, i.e. the guard of $t \in X$ depends only on $M^i$.

If color sets of transitions in a CPN are implicitly given as in [26], the firing mode of a transition results from the bindings of variables assigned to adjacent arcs in the CPN. In that case one can follow the approach described by Christensen and Petrucchi in [14] for modular CPNs which are composed by transition fusion. For a structured representation one needs to distinguish possible variable bindings. Let $< b >$ be an abbreviation for the binding of variables to enable a transition $t$ in a marking $M$, then (4) can be reformulated as

$$\bar{\mathbf{R}}_C = \sum_{t,<b> \in T} \bigotimes_{i=0}^{N-1} \mathbf{R}^i_{t,<b>} = \sum_{t,<b> \in TS} \bigotimes_{i=0}^{N-1} \mathbf{R}^i_{t,<b>} + \bigoplus_{i=0}^{N-1} \mathbf{R}^i_l$$

The approach basically performs an unfolding for synchronized transitions, which is rather natural at the level of $RG(CPN)$ anyway. However the notion of colors and bindings sharpens the problem of oversized, possibly infinite reachability sets for components even if $RS(CPN)$ is finite. Christensen and Petrucchi already referred to this problem in [14]. One might obtain place capacities from P-invariants similar to ordinary PNs, however invariant computation for CPNs is more complex.

The current implementation of our model checker is only able to support CPNs with explicit enumeration of transition colors and trivial guards.

# 8 Conclusion

Although structured representations based on Kronecker algebra have been originally developed for the numerical analysis of Markov chains, we demonstrate in this paper by the help of a non-trivial example taken from literature [22, 32, 23] that a structured representation of the reachability graph is a powerful mean for classical model checking based on $RS$. A structured representation provides a space efficient representation of RG, however $RS$ presentation needs further attention. We extend existing algorithms for $RS$ exploration based on a structured representation [28], such that together with the algorithm of [15] a choice for algorithms is available. We present three representations for $RS$, including the searchtree of [15] and a folded searchtree of which the latter is a new result, which exploits effective techniques from binary decision diagrams(BDDs) [6].

Future work is dedicated to integrate further BDD-methods into $RS$ exploration based on structured representations. Furthermore the combination of our algorithms with recently proposed reduction techniques for the $RS$ of components in [8] seems promising. So far we considered non-hierarchical structured representations, an extension of the technique towards hierarchical representations is subject to ongoing work. Finally the implemented algorithms need to be integrated into an available tool environment. It will become part of a set of analysis engines sharing the APNN format [4] for net descriptions. The current implementation supports colored PNs with explicitly enumerated firing modes in APNN format and so-called low-level nets in PEP format as well.

**Acknowledgements** We thank M. Heiner and J. Spranger for the various PN models of the production cell in APNN and PEP format, J. Esparza and S. Römer for providing source code for a parser of PEP low level nets and a selection of helpful test models for our approach.

# References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, Cambridge, 1986.

[2] A. Arnold. *Finite transition systems: semantics of communicating systems.* Prentice-Hall, Englewood Cliffs, 1994.

[3] F. Bause, P. Buchholz, and P. Kemper. QPN-Tool for the specification and analysis of hierarchically combined Queueing Petri nets. In *Quantitative Evaluation of Computing*

*and Communication Systems, 8th Int. Conf. Modelling Techniques and Tools*, LNCS 977, pages 224–238, 1995.

[4] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletters*, 49:9–27, Oct. 1995.

[5] E. Best and B. Grahlmann. PEP - programming environment based on Petri nets, documentation and user guide. Technical report, Universität Hildesheim, Institut für Informatik, Nov. 1995.

[6] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(*):677–691, Aug. 1986.

[7] P. Buchholz. A hierarchical view of GCSPNs and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15:207–224, 1992.

[8] P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop Petri Nets and Performance Models (PNPM'97), St-Malo (France), June 1997*, pages 81–90. IEEE CS Press, 1997.

[9] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. NASA/CR-97-206274, ICASE Report No 97-66, 1997. submitted for publication.

[10] P. Buchholz and P. Kemper. Numerical analysis of stochastic marked graph nets. In *Proc. 6th Int. Workshop Petri Nets and Performance Models*, pages 32–41. IEEE Computer Society Press, 1995.

[11] P. Buchholz and P. Kemper. Hierarchical reachability graph generation for Petri nets. Technical report No 660, Universität Dortmund, 1997. submitted for publication.

[12] J. Campos, M. Silva, and S. Donatelli. Structured solution of stochastic DSSP systems. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models*, pages 91–100. IEEE Computer Society Press, 1997.

[13] G. Chiola. Compiling techniques for the analysis of stochastic Petri nets. In *Proc. of the 4th Int. Conf. on Modeling Techniques and Tools*, pages 13–27, 1989.

[14] S. Christensen and L. Petrucci. Modular state space analysis of coloured Petri nets. In *Proc. 16th Int. Conf. Application and Theory of Petri Nets*, LNCS 935, pages 201–217. Springer, 1995.

[15] G. Ciardo and A. Miner. Storage alternatives for large structured state spaces. In R. Marie, et al. (eds.), *Proc. 9th Int. Conf. Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 1245, pages 44–57. Springer, 1997.

[16] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proc. of the Workshop on Logics of Programs*, LNCS 131, pages 52–71, New York, May 1981. Springer.

[17] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, Apr. 1986.

[18] E. M. Clarke, J. M. Wing, et al. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, Dec. 1996.

[19] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30(2):116–125, Feb. 1981.

[20] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–26, 1993.

[21] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proc. 15th Int. Conf. Application and Theory of Petri Nets*, LNCS 815, pages 258–277. Springer, 1994.

[22] M. Heiner, P. Deussen, and J. Spranger. A case study in developing control software on manufacturing systems with hierarchical Petri nets. M. Silva, R. Valette and K. Takahashi (eds.), *Workshop Manufacturing and Petri nets, within 17th int. Conf. Application and Theory of Petri nets*, Osaka, Japan, 1996.

[23] M. Heiner, P. Deusen; Petri net based design and analysis o f reactive systems; In: *Proc. 3rd Workshop on Discrete Event Systems (WoDES'96), pp. 308-313.*

[24] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[25] P. Huber, K. Jensen, and R. Shapiro. Hierarchies in coloured Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets*, LNCS 483. Springer, Berlin, 1990.

[26] K. Jensen. *Coloured Petri Nets, Basic Concepts*, volume 1 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin Heidelberg, 1992.

[27] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep. 1996.

[28] P. Kemper. Reachability analysis based on structured representations. In *Proc. 17th Int. Conf. Application and Theory of Petri Nets*, LNCS 1091. Springer, 1996.

[29] P. Kemper. *Superposition of generalized stochastic Petri nets and its impact on performance analysis.* PhD thesis, Universität Dortmund, LS Informatik 4, 1996. Krehl-Verlag, Münster, (in English).

[30] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[31] L. Lamport. "Sometime" is sometimes "Not never" – On the temporal logic of programs. In *Conf. Record of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 174–185, Las Vegas, Nevada, Jan. 1980. ACM, ACM Press.

[32] C. Lewerentz and T. Lindner. Formal development of reactive systems, case study production cell, 1995.

[33] T. Murata. Petri nets: properties, analysis and application. *Proc. of the IEEE*, 77:541–580, 1989.

[34] B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.

[35] B. Plateau and J. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12, 1991.

[36] A. Pnueli. The temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[37] W. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

[38] K. Varpaaniemi, J. Halme, K. Hiekkanen, and T. Pyssysalo. Prod reference manual. Technical Report B13, Helsinki University of Technology, Digital Systems Lab, Espoo, Finland, 1995.