Symbolic Analysis of Bounded Petri Nets

Enric Pastor, Jordi Cortadella, *Member*, *IEEE*, and Oriol Roig, *Member*, *IEEE*

Abstract—This work presents a symbolic approach for the analysis of bounded Petri nets. The structure and behavior of the Petri net is symbolically modeled by using Boolean functions, thus reducing reasoning about Petri nets to Boolean calculation. The set of reachable markings is calculated by symbolically firing the transitions in the Petri net. Highly concurrent systems suffer from the *state explosion problem* produced by an exponential increase of the number of reachable states. This state explosion is handled by using *Binary Decision Diagrams* (BDDs) which are capable of representing large sets of markings with small data structures. Petri nets have the ability to model a large variety of systems and the flexibility to describe causality, concurrency, and conditional relations. The manipulation of vast state spaces generated by Petri nets enables the efficient analysis of a wide range of problems, e.g., deadlock freeness, liveness, and concurrency. A number of examples are presented in order to show how large reachability sets can be generated, represented, and analyzed with moderate BDD sizes. By using this symbolic framework, properties requiring an exhaustive analysis of the reachability graph can be efficiently verified.

Index Terms—Petri nets, formal verification, symbolic methods, Binary Decition Diagrams.

1 Introduction

Petri nets are a graph-based mathematical formalism suitable to describe, model, and analyze the behavior of discrete event concurrent systems. More precisely, Petri nets can describe asynchronous sequential and nonsequential behaviors, including concurrency and nondeterministic choice, where sets of processes can interact, cooperate, and compete. Since their introduction by C.A. Petri in 1962 [11], Petri nets (PNs) have been extensively used in a wide range of areas such as communication protocols and networks, computer architecture, distributed systems, manufacturing planning, digital circuit synthesis and verification, and high-level synthesis.

The existing methods for the analysis of PNs can be mainly classified into four categories [9]: reachability tree methods, enumerative methods, matrix-equation methods, and reduction or decomposition methods. Traditionally, the first and second methods are only applicable to small PNs due to the explosion of the number of markings in concurrent systems, while the third and fourth methods are restricted to particular subclasses of PNs.

Enumerative methods permit verifying properties of finite systems. They can also be successfully combined with analysis methods based on the structure of the PN. However, the potentially huge number of reachable markings in highly concurrent systems becomes the main bottleneck for any enumerative approach. This limitation encourages the study and application of new efficient techniques to overcome this problem.

- *E. Pastor is with the Department of Computer Architecture, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain. E-mail: enric@ac.upc.es.*
- *J. Cortadella is with the Department of Software, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain. E-mail: jordic@lsi.upc.es.*
- *O. Roig is with Theseus Logic Inc., 710 Lakewood Dr., Suite 230, Sunnyvale, CA 94086. E-mail: oriol.roig@theseus.com.*

Manuscript received 14 May 1997; revised 5 Oct. 1999; accepted 10 Jan. 2001. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105056.

In this work, we present a symbolic approach that implicitly enumerates the reachable markings. This method is applicable to the analysis of any type of bounded PNs [10]. The proposed technique is based on the modeling of the PN by means of *Boolean algebras* [1]. Problems like *deadlock* detection, *liveness*, *boundedness*, and *persistence* can be checked by properly manipulating the functions that model the PN.

Methods based on the explicit enumeration of the reachable markings suffer from the state explosion problem due to the arbitrary interleaving of concurrent transitions. The inherent complexity involved in the enumeration of the exponential number of markings in a PN is alleviated by using *Binary Decision Diagrams* (BDD) [2]. BDDs have the capability of representing large sets of encoded data with small data structures and enable the efficient manipulation of those sets. The utilization of BDDs not only provides algorithms which are computationally capable of manipulating large systems (due to its efficient data representation), but also provide an extremely flexible mechanism to manipulate PNs.

Symbolic model checking techniques have already been proposed in [7], [14], [18] for the verification of digital circuits. More recently, [18] introduced special Boolean operators to perform efficient symbolic analysis based on *Zero-Suppressed* BDDs (ZBDDs). The theory presented in this paper is not restricted to any particular class of Decision Diagrams and can be applied to any framework based on Boolean manipulation of Petri nets.

The remainder of this work is organized as follows: In Section 2, we introduce basic definitions on Boolean algebras, Petri nets, and Binary Decision Diagrams. The modeling of PNs is discussed in Section 3. The Boolean functions that define the dynamic behavior of the PN are described in Section 4, while the symbolic reachability analysis algorithm required to efficiently generate the markings in the PN is outlined in Section 5. In order to manipulate bounded PNs, an extension of the basic model, as well as the required reachability techniques, is proposed in Section 6. Section 7 presents a set of PN reductions that can improve the efficiency in the calculation of the state space. Algorithms for the verification of properties, such as concurrency, liveness, and persistence, are presented in Section 8. Several experimental results are presented in Section 9, including a detailed analysis on the efficiency of the method.

2 Basic Definitions

2.1 Boolean Algebras

This section reviews some basic concepts on Boolean algebras. We refer the reader to [1] for a detailed compendium on the subject.

A *set* is a collection of objects called *elements*. In the sequel, we will only consider finite sets. The *cardinality* of a set A, written $|A|$, is the number of elements in the set. The *power set* of a set A is represented by 2^A. Given two sets A and B, a binary *relation* $\mathcal{R}$ between A and B is a subset of the Cartesian product $A \times B$. We write $x\mathcal{R}y$ iff x and y are in relation $\mathcal{R}$. A function f from A to B, $f : A \rightarrow B$, is a relation that associates exactly one element of B to each element of A. A is called the *domain* of the function. B is called the *codomain*. For every element $x \in A$, $f(x) \in B$ is called the *image* of x.

A *Boolean algebra* is a five-tuple $(\mathrm{B}, +, \cdot, 0, 1)$, where B is a set called the *carrier*, $+$ and $\cdot$ are binary operations on B, and 0 and 1 are elements of B. The elements in B satisfy the commutative, distributive, identity, and complement laws. The *algebra of subsets* of a set S, denoted $(2^S, \cup, \cap, \emptyset, S)$, is a Boolean algebra, where 2^S is the set of subsets of S and $\cup$, $\cap$ are the union and intersection operations.

The system $(\mathrm{B}, +, \cdot, 0, 1)$, with $\mathrm{B} = \{0, 1\}$, $+$, and $\cdot$ defined as the *logic OR* and *logic AND* operations, respectively, is a Boolean algebra, also known as the *switching algebra*.

For an integer $n \geq 0$, an n-variable Boolean function is a function $f : \mathrm{B}^n \rightarrow \mathrm{B}$. Let $F_n(\mathrm{B})$ be the set of n-variable Boolean functions, then $F_n(\mathrm{B})$ is the power set of B^n (the characteristic functions of subsets of B^n). The system

$$(F_n(\mathrm{B}), +, \cdot, \mathbf{0}, \mathbf{1}) \tag{1}$$

is the Boolean algebra of Boolean functions, in which "$+$" and "$\cdot$" represent disjunction and conjunction of n-variable Boolean functions and $\mathbf{0}$ and $\mathbf{1}$ represent the "*zero*" and "*one*" functions ($f(x_1, \ldots, x_n) = 0$ and $f(x_1, \ldots, x_n) = 1$). The cardinality of $F_n(\mathrm{B})$, that is, the number of different n-variable functions, is 2^{2^n}.

Let $V \subseteq \mathrm{B}^n$ be a set of elements in the Boolean algebra of n-variable Boolean functions. The *characteristic function* χ_V of the set V is an n-variable Boolean function that evaluates to $\mathbf{1}$ for those elements of B^n that are in V, i.e., $v \in V \Leftrightarrow \chi_V(v) = 1, \forall v \in F_n(\mathrm{B})$.

Theorem 1 (Stone's Representation Theorem). *Every finite Boolean algebra is isomorphic to the Boolean algebra of subsets of some finite set S.*

Stone's Representation Theorem establishes the basis of the approach presented in this work, that is, bounded PNs can be modeled with the Boolean algebra of Boolean functions.

Given the Boolean algebra of n-variable Boolean functions, with n symbols $x_1, \ldots, x_n$, we call each element of B^n a *vertex*. A *literal* is either a variable x_i or its complement $\overline{x_i}$. A *cube c* is a set of literals such that if $x_i \in c$, then $\overline{x_i} \notin c$ and vice versa. A cube is interpreted as the Boolean product of its literals. Note that the set of all cubes with n literals is in one-to-one correspondence with the vertices of B^n.

The Boolean functions $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ are called the positive and negative *cofactors* of f with respect to x_i. If $f : \mathrm{B}^n \rightarrow \mathrm{B}$ is an n-variable Boolean function, then Boole's Expansion Theorem defines that

$$\forall x_i, 1 \leq i \leq n, \quad f(x_1, \ldots, x_n) = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}.$$

The definition of cofactor can also be extended to cubes. Given a cube $c = \hat{x}_1 \cdot c_1$ composed of a literal $\hat{x}_1$ (either x_1 or $\overline{x_1}$) and another cube c_1, then the cofactor of a function with respect to c is recursively defined as: $f_c = (f_{\hat{x}_1})_{c_1}$.

Abstractions are of fundamental use in our framework. They have a direct correspondence to the existential and universal quantifiers applied to predicates in Boolean reasoning. The *existential* and *universal abstractions* of $f(x_1, \ldots, x_n)$ with respect to a variable x_i are defined $\exists_{x_i} f = f_{x_i} + f_{\overline{x_i}}$ and $\forall_{x_i} f = f_{x_i} \cdot f_{\overline{x_i}}$, respectively.

2.2 Petri Nets

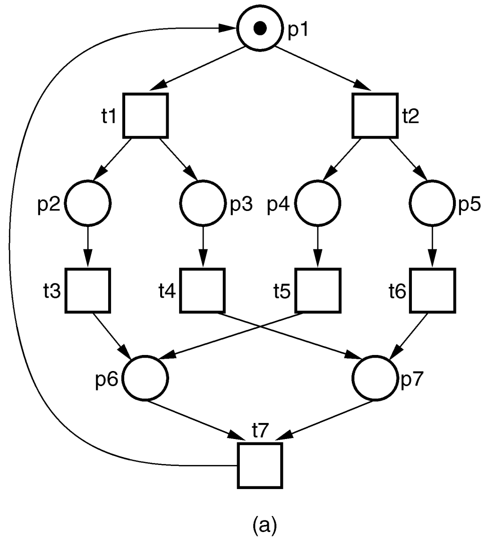
A *Petri net* [9], [12], [13] is a four-tuple $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}, M_o \rangle$, where $\mathcal{P} = \{p_1, \ldots, p_n\}$ and $\mathcal{T} = \{t_1, \ldots, t_m\}$ are finite sets of places and transitions (also called nodes) satisfying $\mathcal{P} \cap \mathcal{T} = \emptyset$ and $\mathcal{P} \cup \mathcal{T} \neq \emptyset$, $\mathcal{W} : (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}) \rightarrow \mathbb{N}$ is the *weighted flow relation*, and M_o is the *initial* marking. A *marking* is a function $M : \mathcal{P} \rightarrow \mathbb{N}$. If k is assigned to a place p by M, we will say that p is marked with k tokens in M. If $\mathcal{W}(u, v) > 0$, then there is an arc from u to v with *weight* $\mathcal{W}(u, v)$. The *pre* and *postset* of a node are specified by a dot-notation: ${}^\bullet u = \{v \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(v, u) > 0\}$ is called the preset of u and $u^\bullet = \{v \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(u, v) > 0\}$ is called the postset of u. Fig. 1a depicts a PN in which all arcs have weight one.

A transition t is *enabled* at a marking M (denoted by $M[t\rangle$) if $\forall p \in {}^\bullet t : M(p) \geq \mathcal{W}(p, t)$. Once a transition t is enabled at a marking M, it may *fire*, reaching a new marking M' (denoted by $M[t\rangle M'$), where $M'(p) = M(p) - \mathcal{W}(p, t) + \mathcal{W}(t, p)$.

A sequence of transitions $\sigma = t_1 \ldots t_{k-1} \in \mathcal{T}^*$ is a *firing sequence* from a marking M_1 to a marking M_k iff there exist markings $M_2, \ldots, M_{k-1}$ such that: $M_i[t_i\rangle M_{i+1}$ for $1 \leq i \leq k - 1$. Marking M_k is said to be reachable from M_o by firing $\sigma : M_o[\sigma\rangle M_k$. $[M\rangle$ is the set of markings reachable from M by firing any sequence of transitions, i.e., $M' \in [M\rangle \Leftrightarrow \exists \sigma \in \mathcal{T}^* : M[\sigma\rangle M'$. $[M_o\rangle$ is the set of all markings reachable from M_o. The automaton that contains the set of reachable markings and all possible firing sequences of a PN is called the *reachability graph*. A transition $t \in \mathcal{T}$ is *live* iff

$$\forall M \in [M_o\rangle : \exists M' \in [M\rangle \text{ such that} M'[t\rangle.$$

A PN is live iff every transition in the PN is live. A marking $M \in [M_o\rangle$ is a *home marking* iff $\forall M' \in [M\rangle : M \in [M'\rangle$. A place



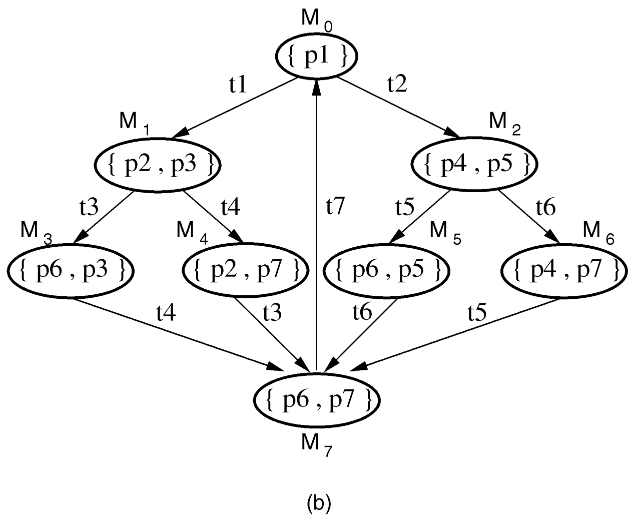


Fig. 1. (a) A safe Petri net with its initial marking, (b) its corresponding reachability graph.

$p \in \mathcal{P}$ is called *k-bounded* ($k \in \mathbb{N}$) iff $\forall M \in [M_o\rangle : M(p) \leq k$. A PN is *k*-bounded iff every place is *k*-bounded. A PN is bounded iff it is *k*-bounded for some *k*. A PN is *safe* iff it is 1-bounded.

The symbolic methods presented in this work require the PNs to be bounded. It is also required to know an upper bound of the number of tokens that a place may hold. In general, a conservative bound for a place can be calculated in advance by solving a specific *Linear Programming Problem* [4]. In case such a bound cannot be obtained, the user must provide estimated values. Note, however, that any estimated bound will be validated during the symbolic generation of the reachability set (see Section 5.1).

Fig. 1b depicts the reachability graph for the PN presented in Fig. 1a. This graph has eight reachable markings, each one annotated with the subset of places that are marked in it. In general, a marking in a safe PN can be represented by a set of places $m = \{p_1, \ldots, p_k\} \subseteq \mathcal{P}$, where $p \in m$ denotes the fact that there is a token in p. Additionally, any set of markings in $[M_o\rangle$ can be represented by a set $\mathcal{M} = \{m_1, \ldots, m_l\}$ of subsets of the places in the PN, every set of places $m_i \in \mathcal{M}$ corresponding to one of the markings.

2.3 Binary Decision Diagrams

Basic definitions for Binary Decision Diagrams were given in [2]. In this section, we review some of these definitions for reference.

A Binary Decision Diagram (BDD) is a directed acyclic graph with two sink nodes, labeled 0 and 1, representing the Boolean functions **0** and **1**. Each nonsink node is labeled with a Boolean variable v and has two out-edges labeled 1 (or then) and 0 (or else). Each nonsink node represents the Boolean function corresponding to its 1-edge if $v = 1$ or the Boolean function corresponding to its 0-edge if $v = 0$.

An *Ordered Binary Decision Diagram* (OBDD) is a BDD in which variables are totally ordered and every source to sink path in the OBDD visits the variables in ascending order. A *Reduced Ordered Binary Decision Diagram* (ROBDD) is an OBDD where each node represents a distinct logic function. ROBDDs are canonical representations of Boolean functions: For a fixed variable order, two functions are equivalent if and only if their BDDs are isomorphic.

ROBDDs have emerged as an efficient form to manipulate large functions (with hundreds of variables). It is known that the size of the ROBDD for a function depends on the chosen variable order [15]. However, this work does not tackle the variable ordering problem.

3 Symbolic Modeling of Safe Petri Nets

This section describes how a *safe* PN can be modeled using Boolean algebras. In Section 6, this model will be extended for bounded PNs.

Let $M_P = 2^{\mathcal{P}}$ be the set of all subsets of places representing markings of a safe PN with $|\mathcal{P}|$ places. The system $(2^{M_P}, \cup, \cap, \emptyset, M_P)$ is the Boolean algebra of sets of safe markings (see (1)). This system is *isomorphic* to the Boolean algebra of n-variable Boolean functions, where $n = |\mathcal{P}|$ (see Theorem 1).

Given this, there is a one-to-one correspondence between markings in M_P and vertices of B^n. Any marking $M \in M_P$ in a safe PN can be represented by a vertex of B^n and determined by an *encoding function* $\mathcal{E} : M_P \rightarrow \mathrm{B}^n$. The image of every marking $M \in M_P$ is encoded into a vertex $(p_1, \ldots, p_n) \in \mathrm{B}^n$ such that:

$$\forall i \in 1, \ldots, n \quad p_i = \begin{cases} 1 & \text{if } p_i \in M \\ 0 & \text{if } p_i \notin M. \end{cases}$$

Henceforth, we will use p_i to denote either a place in $\mathcal{P}$ or its corresponding Boolean variable and M to denote either a reachable marking or the corresponding set of places that hold tokens in the marking. As an example, in Fig. 1, both the vertex $(0, 1, 0, 0, 0, 0, 1) \in \mathrm{B}^7$ and the cube $\overline{p_1}\, p_2\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, p_7$ represent the marking in which p_2 and p_7 are marked and p_1, p_3, p_4, p_5, and p_6 are not marked.

Extending the use of the *encoding function* $\mathcal{E}$, each set of markings $\mathcal{M} \in 2^{M_P}$ has a corresponding *image* $V \in F_n(\mathrm{B})$ according to $\mathcal{E}$, defined by:

$$V = \{v \in \mathrm{B}^n \mid \exists M \in \mathcal{M} : v = \mathcal{E}(M)\}.$$

Then, the *characteristic function* of the set $\mathcal{M}$ is a function $\chi_{\mathcal{M}} : \mathrm{B}^n \to \mathrm{B}$ that evaluates to 1 for those vertices that correspond to markings belonging to $\mathcal{M}$; that is, $\chi_{\mathcal{M}} = \chi_V$. From now on, and for the sake of simplicity, we will indistinctly use $\mathcal{M}$ and $\chi_{\mathcal{M}}$ to denote the characteristic function of a set of markings $\mathcal{M}$. All set manipulations can by applied as Boolean operations directly to the characteristic functions. For example, given the sets of markings $\mathcal{M}_1, \mathcal{M}_2 \in M_P$, union $\chi_{\mathcal{M}_1 \cup \mathcal{M}_2} = \chi_{\mathcal{M}_1} + \chi_{\mathcal{M}_2}$, intersection $\chi_{\mathcal{M}_1 \cap \mathcal{M}_2} = \chi_{\mathcal{M}_1} \cdot \chi_{\mathcal{M}_2}$, and complement $\chi_{\overline{\mathcal{M}_1}} = \overline{\chi_{\mathcal{M}_1}}$ can be implemented with the corresponding Boolean operators.

As an example, the set of markings

$$\mathcal{M} = \{\{p_2\, p_3\}, \{p_2\, p_7\}, \{p_4\, p_7\}\}$$

has the characteristic function:

$$\chi_{\mathcal{M}} = \overline{p_1}\, p_2\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, (p_3 \oplus p_7) + \overline{p_1}\, \overline{p_3}\, \overline{p_5}\, \overline{p_6}\, p_7\, (p_2 \oplus p_4).$$

Characteristic functions can also be used to represent binary relations between sets of markings. Given two sets $\mathcal{M}$ and $\mathcal{M}'$, a binary relation $R \subseteq \mathcal{M} \times \mathcal{M}'$ can be represented by using two sets of Boolean variables to encode the elements of each set. Taking Boolean variables $p_1, \dots, p_n$ for $\mathcal{M}$ and $q_1, \dots, q_n$ for $\mathcal{M}'$, the characteristic function of R is defined by:

$$\begin{aligned}
&\forall (p_1, \dots, p_n), (q_1, \dots, q_n) \in \mathrm{B}^n \\
&\chi_R(p_1, \dots, p_n, q_1, \dots, q_n) = 1 \;\Leftrightarrow\; \exists (M, M') \in R : \\
&[\mathcal{E}(M) = (p_1, \dots, p_n) \wedge \mathcal{E}(M') = (q_1, \dots, q_n)].
\end{aligned}$$

Given the binary relation R between sets $\mathcal{M}$ and $\mathcal{M}'$, the elements of $\mathcal{M}$ that are in relation with some element of $\mathcal{M}'$ are defined by the set:

$$R_{\mathcal{M}} = \{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}' \;:\; (M, M') \in R\},$$

and its characteristic function $\chi_{R(\mathcal{M})}$ is computed as:

$$\chi_{R_{\mathcal{M}}}(p_1, \dots, p_n) = \exists_{q_1, \dots, q_n}\, \chi_R(p_1, \dots, p_n, q_1, \dots, q_n).$$

4 Dynamic Behavior of Safe Petri Nets

This section introduces the Boolean functions and relations that model the dynamic behavior of safe PNs. The enabling and firing of individual transitions are analyzed first. Based on these results, several algorithms are described to efficiently construct the reachability set and to verify the initial safeness assumption.

4.1 Transition Firing

The structure of a PN defines a set of local changes of state, called *transition firing functions*, that determine its dynamic behavior. Let $\mathrm{E}_t \subseteq M_P$ be the set of markings in which transition t is enabled. The *transition function* for a transition $t \in \mathcal{T}$ is a partially defined function $\delta^t : \mathrm{B}^n \to \mathrm{B}^n$ that transforms every marking $M \in \mathrm{E}_t$ into a new marking $M' \in M_P$ by firing transition t, i.e., $M' = \delta^t(M)$. The image of δ^t for markings outside E_t is undefined.

This concept is equivalent to the one-step reachability in PNs. The transition function $\delta^t = (\delta^t_1, \dots, \delta^t_{|\mathcal{P}|})$ for a transition $t \in \mathcal{T}$ defines how the contents of each place is transformed as a result of firing t at marking in which t is enabled. The function is defined as $\forall i \in 1, \dots, |\mathcal{P}|$:

$$\delta^t_i(p_1, \dots, p_n) = \begin{cases} 1 & \text{if } p_i \in t^\bullet \\ 0 & \text{if } p_i \in {}^\bullet t \text{ and } p_i \notin t^\bullet \\ p_i & \text{otherwise.} \end{cases} \tag{2}$$

The characteristic function E_t of the set of markings in which transition t is enabled is defined as:

$$\mathrm{E}_t = \prod_{p_i \in {}^\bullet t} p_i. \tag{3}$$

By firing transition t, the function returns 1 if p is in its postset and 0 if p is in its preset (but not a self-loop). Otherwise, the place remains with the same value.

A marking M_k is *reachable* in k steps from the initial marking M_o if there is a sequence of markings $M_1, M_2, \dots, M_{k-1}$ and a sequence of transitions $t_1, t_2, \dots, t_k$, such that, $\delta^{t_i}(M_{i-1}) = M_i$, $\forall i, 1 \le i \le k$. Following the example in Fig. 1 that requires the Boolean variables $p_1, \dots, p_7$, the transition and enabling functions are:

$$\begin{array}{lllllllll}
 & (\delta_1) & (\delta_2) & (\delta_3) & (\delta_4) & (\delta_5) & (\delta_6) & (\delta_7) & \\
\delta^{t_1}(M) = & (0, & 1, & 1, & p_4, & p_5, & p_6, & p_7), & E_{t_1} = p_1, \\
\delta^{t_2}(M) = & (0, & p_2, & p_3, & 1, & 1, & p_6, & p_7), & E_{t_2} = p_1, \\
\delta^{t_3}(M) = & (p_1, & 0, & p_3, & p_4, & p_5, & 1, & p_7), & E_{t_3} = p_2, \\
\delta^{t_4}(M) = & (p_1, & p_2, & 0, & p_4, & p_5, & p_6, & 1), & E_{t_4} = p_3, \\
\delta^{t_5}(M) = & (p_1, & p_2, & p_3, & 0, & p_5, & 1, & p_7), & E_{t_5} = p_4, \\
\delta^{t_6}(M) = & (p_1, & p_2, & p_3, & p_4, & 0, & p_6, & 1), & E_{T_6} = p_5, \\
\delta^{t_7}(M) = & (1, & p_2, & p_3, & p_4, & p_5, & 0, & 0), & E_{t_7} = p_6 p_7.
\end{array}$$

Therefore, firing transition t_1 from markings

$$p_1\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7}$$

and

$$\overline{p_1}\, p_2\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7},$$

where t_1 is enabled, results in:

$$\overline{p_1}\, p_2\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7} = \delta^{t_1}(p_1\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7})$$

and

$$\overline{p_1}\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7} = \delta^{t_1}(\overline{p_1}\, p_2\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7}).$$

We now consider the firing of transitions in sets of markings rather than using a marking-per-marking basis. Let us define the *constrained image* (or simply image) of δ^t as a function that transforms a set of markings $\mathcal{M}$ into the set of markings $\mathcal{M}'$ that can be reached from $\mathcal{M}$ by firing t. The constrained image of δ^t is denoted by $Img(t, \mathcal{M})$ and computed:

$$Img(t, \mathcal{M}) = \left\{M' \in M_P : \exists M \in \mathcal{M} \wedge \mathrm{E}_t, \delta^t(M) = M'\right\}. \tag{4}$$

Using the terminology for verification of sequential machines, function Img performs the *image computation* of a transition [5], [6]. We can now apply two different strategies to implement the image computation for transitions using BDDs: *topological image computation* and *transition relation*.

4.2 Topological Image Computation

Image computation for single transitions can be efficiently implemented by using the topological information of the PN and the characteristic function of some selected sets of markings. In addition to E_t (previously defined in (3)), we present the characteristic function of some important sets related to a transition $t \in \mathcal{T}$:

$$\mathrm{NPM}_t = \prod_{p_i \in {}^\bullet t} \overline{p_i} \quad \text{(no predecessor of } t \text{ is marked)},$$
$$\mathrm{ASM}_t = \prod_{p_i \in t^\bullet} p_i \quad \text{(all successors of } t \text{ are marked)},$$
$$\mathrm{NSM}_t = \prod_{p_i \in t^\bullet} \overline{p_i} \quad \text{(no successor of } t \text{ is marked)}.$$

Given these characteristic functions, the image computation for transitions is reduced to calculating the Boolean formula:

$$Img(t, \mathcal{M}) = (\mathcal{M}_{\mathrm{E}_t} \cdot \mathrm{NPM}_t)_{\mathrm{NSM}_t} \cdot \mathrm{ASM}_t. \qquad (5)$$

We will show with the example of Fig. 1a how this formula "simulates" firing a transition t. Given the set of markings $\mathcal{M} = \{\{p_2\, p_3\}, \{p_2\, p_7\}, \{p_4\, p_7\}\}$, with its characteristic function:

$$\mathcal{M} = \overline{p_1}\, p_2\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7} + \overline{p_1}\, p_2\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, p_7 + \overline{p_1}\, \overline{p_2}\, \overline{p_3}\, p_4\, \overline{p_5}\, \overline{p_6}\, p_7,$$

we will calculate $\mathcal{M}' = \delta(\mathcal{M}, t_3)$. First, $\mathcal{M}_{\mathrm{E}_{t_3}}$ (the cofactor of $\mathcal{M}$ with respect to $\mathrm{E}_{t_3} = p_2$) selects those markings in which t_3 is enabled[1] and removes its predecessor places from the characteristic function:

$$\mathcal{M}_{\mathrm{E}_{t_3}} = \overline{p_1}\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7} + \overline{p_1}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, p_7.$$

Then, the product with $\mathrm{NPM}_{t_3} = \overline{p_2}$ simulates the elimination of tokens in the predecessor places:

$$\mathcal{M}_{\mathrm{E}_{t_3}} \cdot \mathrm{NPM}_{t_3} = \overline{p_1}\, \overline{p_2}\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, \overline{p_7} + \overline{p_1}\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, \overline{p_6}\, p_7.$$

Next, the cofactor with respect to $\mathrm{NSM}_{t_3} = \overline{p_6}$ removes all successor places from the characteristic function:

$$(\mathcal{M}_{\mathrm{E}_{t_3}} \cdot \mathrm{NPM}_{t_3})_{\mathrm{NSM}_{t_3}} = \overline{p_1}\, \overline{p_2}\, p_3\, \overline{p_4}\, \overline{p_5}\, \overline{p_7} + \overline{p_1}\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, p_7.$$

Finally, the product with $\mathrm{ASM}_{t_3} = p_6$ adds a token to all successor places of t_3:

$$(\mathcal{M}_{\mathrm{E}_{t_3}} \cdot \mathrm{NPM}_{t_3})_{\mathrm{NSM}_{t_3}} \cdot \mathrm{ASM}_{t_3} = \overline{p_1}\, \overline{p_2}\, p_3\, \overline{p_4}\, \overline{p_5}\, p_6\, \overline{p_7} + \overline{p_1}\, \overline{p_2}\, \overline{p_3}\, \overline{p_4}\, \overline{p_5}\, p_6\, p_7,$$

generating the characteristic function of the set of markings $\mathcal{M}' = \{\{p_3\, p_6\}, \{p_6\, p_7\}\}$.

Note that (5) is correctly defined only for safe PNs. However, this safeness assumption can be verified, as will be shown in Section 5.1.

1. This implementation of the transition function satisfies the requirements imposed by the partial definition of (2), i.e., no image is generated for markings outside E_t.

4.3 Image Computation Based on Transition Relations

The transition function relates sets of markings $\mathcal{M}' = \delta^t(\mathcal{M})$ such that the markings in $\mathcal{M}'$ are reachable after firing transition t from the subset of $\mathcal{M}$ in which t is enabled. The relation induced by δ^t can be represented by a characteristic function R_t that requires two different sets of variables: $p_1, \ldots, p_n$ for $\mathcal{M}$ and $q_1, \ldots, q_n$ for $\mathcal{M}'$, respectively ($n = |\mathcal{P}|$). According to the definition of δ^t, its characteristic function is described by the binary relation:[2]

$$\mathcal{R}_t(p_1, \ldots, p_n, q_1, \ldots, q_n) = \prod_{i=1}^{|\mathcal{P}|} \big(q_i \equiv \delta_i^t(p_1, \ldots, p_n)\big) \cdot \mathrm{E}_t.$$

Finding the set of markings $\mathcal{M}'$ that can be reached after firing transition t from any marking in the set $\mathcal{M}$ (in which t is enabled) is reduced to computing:

$$Img(t, \mathcal{M}) = \exists_{p_1, \ldots, p_n} [\mathcal{R}_t(p_1, \ldots, p_n, q_1, \ldots, q_n) \cdot \mathcal{M}]. \qquad (6)$$

As an example, we provide the characteristic function for the transition relation of t_1 in Fig. 1:

$$\mathcal{R}_{t_1}(p_1, \ldots, p_7, q_1, \ldots, q_7) = (\overline{q_1} \cdot q_2 \cdot q_3 \cdot (q_4 \equiv p_4) \cdot (q_5 \equiv p_5) \cdot (q_6 \equiv p_6) \cdot (q_7 \equiv p_7)) \cdot (p_1).$$

For this transition, a token is removed from p_1 ($\overline{q_1}$) and tokens are added for p_2 and p_3 ($q_2 \cdot q_3$). No change occurs at any other place $((q_4 \equiv p_4) \cdot (q_5 \equiv p_5) \cdot (q_6 \equiv p_6) \cdot (q_7 \equiv p_7))$. Additionally, firing can only occur at markings where t_1 is enabled (p_1).

The one-step reachability relation of the whole PN is the union of the images of all transitions:

$$Img(PN, \mathcal{M}) = \exists_{p_1, \ldots, p_n} \sum_{\forall t_j \in \mathcal{T}} \left[\prod_{i=1}^{|\mathcal{P}|} \Big(q_i \equiv \delta_i^{t_j}(p_1, \ldots, p_n)\Big) \cdot \mathrm{E}_t \cdot \mathcal{M} \right].$$

This function computes all markings that can be reached in one step from $\mathcal{M}$.

The main computational problem in image computation with the transition relation method appears when taking the conjunction $\prod_{i=1}^{|\mathcal{P}|}$. Even if the BDDs for $(q_i \equiv \delta_i^t)$ and the final result $\mathcal{R}_t$ are small, the product may be too large in some intermediate result. A substantial increase in efficiency can be obtained using a *partitioned image computation* as described in [3].

5 Petri Net Traversal and Reachable Markings

The set of reachable markings from M_o can be efficiently calculated by using the image computation in a *symbolic traversal* algorithm. The objective of the symbolic manipulation is to fire multiple transitions simultaneously from sets of markings and, therefore, reduce the number of operations required to derive the reachability set. This section introduces an approach similar to *symbolic breadth-first-search* (BFS) traversal for Finite State Machines [5], [6]. The

2. Note that the operator $a \equiv b$ stands for a equivalent to b and it is defined as $a \overline{\oplus} b = ab + \overline{a}\overline{b}$.

```
$traverse\_Petri\_net$ ($N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}, M_o \rangle$) {
    $Reached$ := $From$ := $M_o$;
    repeat
        $To$ := 0;
        foreach $t \in \mathcal{T}$ do $To$ := $To + Img(t, From)$;
        $New$ := $To - Reached$;
        $From$ := $New$;
        $Reached$ := $Reached + New$;
    until ($New = 0$);
    return $Reached$; /*The set of all reached markings from $M_o$*/
}
```

Fig. 2. Algorithm for symbolic PN traversal using the Img function.

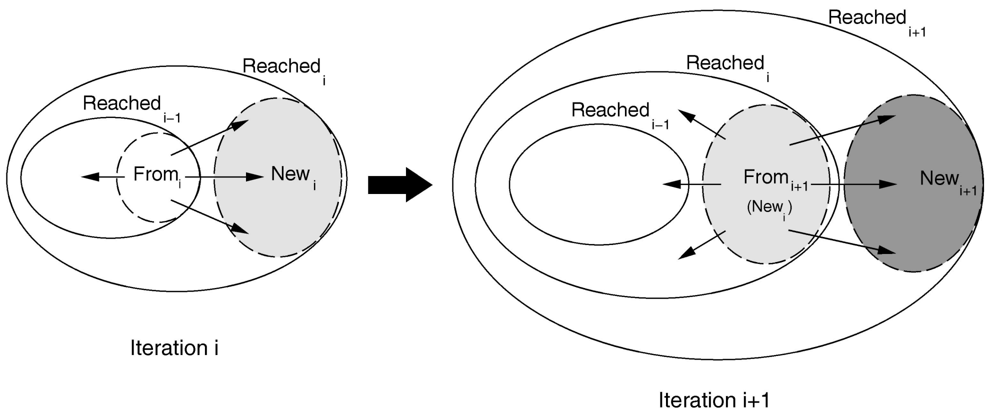


Fig. 3. Generic iteration for the symbolic traversal methodology from an initial marking.

reachability set can be obtained by computing the *least fix point* of the following recurrence:

$$\begin{aligned} S_o &= M_o \\ S_{i+1} &= S_i \cup Img(PN, S_i). \end{aligned}$$

The algorithm presented in Fig. 2 traverses the PN and calculates the reachability set from the initial marking. The union and difference of sets of markings are performed by manipulating their corresponding characteristic functions.

Given the ith iteration of the outermost loop in the algorithm, the traversal obtains all markings reachable in one step from the set From_i (the index in the set denotes the iteration number). The algorithm applies the Img function for every transition in the PN[3] (see Fig. 3). Only those markings that are new in the set of reachable markings (set New_i) are considered for the next iteration. Hence, the set New_i is transformed into the set From_{i+1} for the next iteration. The accumulation of sets of new markings (New_i) results in a monotonically increasing set of reachable markings ($\mathrm{Reached}_i$). The algorithm iterates until no new markings are generated, i.e., until it reaches a fixed point. The number of iterations performed by the traversal is determined by the maximum number of firings from the initial marking to the first occurrence of any of the reachable markings (called the *sequential depth* of the PN). The sequential depth of the PN in Fig. 1a is four, which is the number of transitions firing from M_o to itself.

3. An alternative approach could be to use the Img function of the whole PN. In that case, it is not necessary to iterate over all transitions in the net.

As an example, take the initial marking $\{p_1\}$ in Fig. 1. After the first iteration of the *repeat* loop (by firing transitions t_1 and t_2), the algorithm yields

$$\begin{aligned} To &= \{\{p_2p_3\}, \{p_4p_5\}\} \\ New &= \{\{p_2p_3\}, \{p_4p_5\}\} - \{\{p_1\}\} = \{\{p_2p_3\}, \{p_4p_5\}\}, \end{aligned}$$

and

$$Reached = \{\{p_1\}, \{p_2p_3\}, \{p_4p_5\}\}.$$

The final set of reachable markings is shown in Fig. 1b, where nodes represent markings and edges the firing of transitions.

5.1 Safeness Verification

Up to this point, the calculation of the reachability set by means of image computation has been implemented under the assumption that the PN is safe. This calculation is erroneous if the firing of a transition requires storing more than one token in any place. Unsafe PNs cannot be represented by encoding each place with one Boolean variable and, therefore, cannot be manipulated with the present model. However, this extension will be considered in Section 6.

Detecting unsafeness can be done by identifying a marking M in which a transition t is enabled and some

```
$is\_safe$ ($N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}, M_o \rangle$, $[M_o\rangle$) {
    foreach $t \in \mathcal{T}$ do
        $Succ\_p$ := 0;
        $Enabled$ := $[M_o\rangle \cdot \mathrm{E}_t$;
        foreach ($p_i \in t^\bullet \wedge p_i \notin {}^\bullet t$) do
            $Succ\_p$ := $Succ\_p + p_i$;
        if ($Enabled \cdot Succ\_p \neq 0$) then return false;
    return true;
}
```

Fig. 4. Algorithm for *safeness checking* of a PN.

successor place $p \in t^\bullet$ not included in a self-loop ($p \notin {}^\bullet t$) is already marked. In that situation, after firing transition t, place p will contain two tokens. Formally, a PN is not safe if:

$$\exists\,(M \in [M_o\rangle, t \in \mathcal{T}, p \in \mathcal{P}) :$$
$$[M[t\rangle \;\wedge\; p \in t^\bullet \;\wedge\; p \notin {}^\bullet t \;\wedge\; M(p) = 1].$$

Given the computed set of reachable markings, the algorithm depicted in Fig. 4 detects whether a PN is safe or not by checking one equation for each transition.

On the other hand, safeness can also be detected every time a transition t is going to be fired from a set of markings. Given the set of markings From in the algorithm of Fig. 2, the safeness of the PN can be verified at each iteration of the algorithm by checking that the following formula holds at the beginning of the loop:

$$\forall t \in \mathcal{T} : \left[\mathrm{From} \cdot \mathrm{E}_t \cdot \sum_{(p \in t^\bullet) \wedge (p \notin {}^\bullet t)} p\right] = 0. \qquad (7)$$

5.2 Improved Reachability Analysis

This section introduces an improvement over the reachability analysis algorithm in Fig. 3 based on the individual firing of each transition by its Img function. The proposed method, named *transition chaining*, is based on the fact that each iteration of the outermost loop only generates markings that are reachable in one step. Therefore, to traverse the whole PN, it is necessary to iterate the number of times indicated by the sequential depth on the PN. This limitation can be overcome if the new markings that are generated after applying the image computation are immediately reused in the traverse.

Assume the PN structure described in Fig. 6a which contains two concurrent transitions t_1 and t_2 that enable the firing of transition t_3. This PN has five reachable markings. The BFS reachability algorithm presented in Fig. 2 requires three iterations to generate all the markings (see Fig. 6b). Starting from M_1, in the first iteration (i1), t_1 and t_2 will fire, reaching M_2 and M_3, respectively. In a second iteration (i2), t_1 and t_2 will fire again, both reaching M_4. Finally, in the third iteration (i3), t_3 will fire, reaching M_5.

We can modify the basic BFS reachability algorithm in order to reuse the markings that are computed every time the Img function is evaluated. The newly generated markings, instead of being accumulated in the To set, will be placed back to the From set, to be reused in the same iteration (see Fig. 5)—this basic chaining technique is named *greedy chaining*.

```
foreach $t \in \mathcal{T}$ do                                 foreach $t \in \mathcal{T}$ do
    $\boxed{To := To + Img(t, From);}$                   $\boxed{From := From + Img(t, From);}$
$New$ := $To - Reached$;                     $\Rightarrow$      $New$ := $From - Reached$;
$From$ := $New$;                                          $From$ := $New$;
$Reached$ := $Reached + New$;                             $Reached$ := $Reached + New$;
```

Fig. 5. Symbolic PN traversal applying transition chaining.

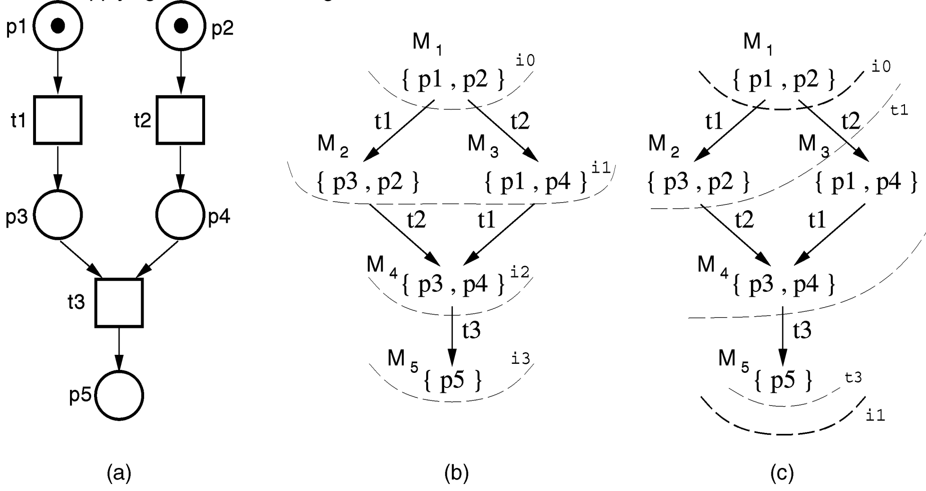


Fig. 6. Application example for the chaining reachability techniques.

TABLE 1
Encoding of Places for *k*-Bounded PNs (with $k = 3$)

# tokens	one-hot encoding		binary encoding
	pure	modified	
0	$\overline{p^3}\ \overline{p^2}\ \overline{p^1}\ p^0$	$\overline{p^2}\ \overline{p^1}\ \overline{p^0}$	$\overline{p^1}\ \overline{p^0}$
1	$\overline{p^3}\ \overline{p^2}\ p^1\ \overline{p^0}$	$\overline{p^2}\ \overline{p^1}\ p^0$	$\overline{p^1}\ p^0$
2	$\overline{p^3}\ p^2\ \overline{p^1}\ \overline{p^0}$	$\overline{p^2}\ p^1\ \overline{p^0}$	$p^1\ \overline{p^0}$
3	$p^3\ \overline{p^2}\ \overline{p^1}\ \overline{p^0}$	$p^2\ \overline{p^1}\ \overline{p^0}$	$p^1\ p^0$

The improved reachability algorithm will require only one outermost iteration and three innermost iterations to explore all possible markings in the example of Fig. 6c. Assume that we process transitions in the order t_1, t_2, t_3. Starting from M_1, in the first iteration transition, t_1 will fire, reaching M_2. From $\{M_1, M_2\}$, t_2 will fire, reaching M_3 and M_4. Finally, from $\{M_1, \ldots, M_4\}$, t_3 will fire, generating the whole reachability set.

In practice, the greedy chaining technique can reduce the number of iterations of the BFS algorithm in at least one order of magnitude (see the experimental results in Section 9). The method is especially effective if the appropriate firing order of the transitions is selected, i.e., given a transition t, all transitions in its preset $^\bullet(^\bullet t)$ should be fired first—named *BFS chaining*. Note, however, that this is a heuristic technique that does not guarantee an optimal chaining order. As an example, for the PN in Fig. 1a, we could select the firing order $t_1, t_3, t_4, t_2, t_5, t_6, t_7$.

6 Weighted and Bounded Petri Nets

This section extends the PN modeling and analysis techniques to weighted and bounded nets.

6.1 Place Encoding

A place $p \in \mathcal{P}$ that may contain up to k tokens can be represented by a set of Boolean variables, $p^0, \ldots, p^{\mathrm{K_p}}$, that encode the up-to-*k* possible number of tokens in *p*. The number of required variables depends on the type of encoding. Different types of encoding strategies can be considered, e.g., *binary encoding*, *one-hot encoding*, etc.

In a *one-hot* encoding scheme, $k+1$ variables are needed to encode a place. For example, in a 3-bounded PN, the number of tokens in place p could be represented by four variables (see Table 1). This scheme can be relaxed by using the zero code to encode a nonempty place. In that case, only k variables are required. When using a *binary encoding* scheme, $\lceil \log_2(k+1) \rceil$ Boolean variables are necessary for a *k*-bounded place, e.g., two variables are required in a 3-bounded PN (see Table 1). A conventional binary encoding can be used to encode the natural numbers $0, \ldots, k$ as a vector of Boolean variables $p^0, \ldots, p^{\mathrm{K_p}}$ (for $\mathrm{K_p} = \lceil \log_2(k+1) \rceil - 1$). For any natural number N such that $N = \sum_0^{\mathrm{K_p}} n_i\, 2^i$, then:

$$\forall i \in 0, \ldots, \mathrm{K_p}\ \ p^i = \begin{cases} 1 & n_i = 1 \\ 0 & n_i = 0. \end{cases}$$

Given that the number of variables, which is a critical parameter in the efficiency of BDD algorithms, is larger for one-hot encoding than for binary encoding, we will concentrate on the latter strategy.

Fig. 7 describes a bounded PN that will be used as an example along this section. The bounds for places p_1, p_2, p_3, and p_4 are 3, 2, 2, and 6, respectively. Therefore, places p_1, p_2, and p_3 are encoded with two variables and p_4 is encoded with three variables. The characteristic function of the initial marking of this PN when using a binary encoding is $p_1^1\overline{p_1^0}\ \ \overline{p_2^1}\,\overline{p_2^0}\ \ \overline{p_3^1}p_3^0\ \ \overline{p_4^2}\,\overline{p_4^1}\,\overline{p_4^0}$.

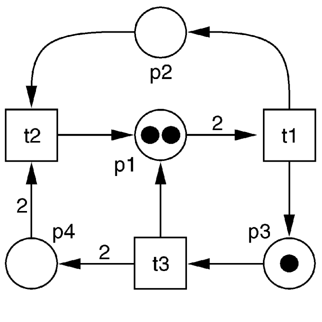
6.2 Transition Firing

This section introduces the transition functions and transition relations required to implement weighted PNs using a binary encoding. The transition function for a transition t (previously described in (2)) should be rewritten as:

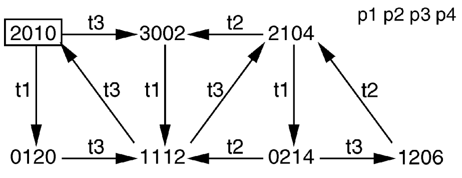
$$\delta_i^t(p_1, \ldots, p_n) = \begin{cases} M(p_i) - \mathcal{W}(p_i, t) & \text{if } p_i \in {}^\bullet t \backslash t^\bullet, \\ M(p_i) + \mathcal{W}(t, p_i) & \text{if } p_i \in t^\bullet \backslash {}^\bullet t, \\ M(p_i) - \mathcal{W}(p_i, t) + \mathcal{W}(t, p_i) & \text{if } p_i \in t^\bullet \cap {}^\bullet t, \\ M(p_i) & \text{otherwise.} \end{cases}$$

The characteristic function of the set of bounded markings in which transition t is enabled (E_t) is also rewritten as:

$$\mathrm{E}_t = \prod_{p \in {}^\bullet t} (M(p) \geq \mathcal{W}(p, t)).$$



(a)



(b)

Fig. 7. Bounded and weighted PN.

When using a binary encoding scheme, the number of tokens in place p is represented with a set of Boolean variables $p^0, \dots, p^{\mathrm{K_p}}$ and the weight $\mathcal{W}(p,t)$ is represented by the same number of binary encoded Boolean constants $w^0, \dots, w^{\mathrm{K_p}}$. Then, the relation $M(p) \geq \mathcal{W}(p,t)$ can be described by the equation:[4]

$$\begin{aligned}
&M(p) \geq \mathcal{W}(p,t) \equiv (p^{\mathrm{K_p}} > w^{\mathrm{K_p}}) \\
&\quad + (p^{\mathrm{K_p}} \equiv w^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} > w^{\mathrm{K_p}-1}) \\
&\quad + (p^{\mathrm{K_p}} \equiv w^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} \equiv w^{\mathrm{K_p}-1}) \cdot (p^{\mathrm{K_p}-2} > w^{\mathrm{K_p}-2}) \\
&\quad \cdots\cdots \\
&\quad + (p^{\mathrm{K_p}} \equiv w^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} \equiv w^{\mathrm{K_p}-1}) \cdot \ldots \cdot (p^1 \equiv w^1) \cdot (p^0 > w^0) \\
&\quad + (p^{\mathrm{K_p}} + w^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} \equiv w^{\mathrm{K_p}-1}) \cdot \ldots \cdot (p^1 \equiv w^1) \cdot (p^0 \equiv w^0).
\end{aligned}$$

Hence, the markings in which a transition t is enabled are defined by the characteristic function E_t:

$$\prod_{p \in {}^\bullet t} \left[\sum_{i=0}^{\mathrm{K_p}} \left[(p^i > w^i) \cdot \prod_{j=i+1}^{\mathrm{K_p}} (p^j \equiv w^j) \right] + \prod_{i=0}^{\mathrm{K_p}} (p^i \equiv w^i) \right].$$

Given the example in Fig. 7, the characteristic functions for each transition will be:

$$\begin{aligned}
E_{t_1} &= [(p_1^0\,\overline{0}) \cdot (p_1^1 \equiv 1) + (p_1^1\,\overline{0}) + (p_1^0 \equiv 0) \cdot (p_1^1 \equiv 1)] = p_1^1, \\
E_{t_2} &= [(p_2^0\,\overline{1}) \cdot (p_2^1 \equiv 0) + (p_2^1\,\overline{0}) + (p_2^0 \equiv 1) \cdot (p_2^1 \equiv 0)] \\
&\qquad [(p_4^0\,\overline{0}) \cdot (p_4^1 \equiv 1) \cdot (p_4^2 \equiv 0) + (p_4^1\,\overline{1}) \cdot (p_4^2 \equiv 0) + (p_4^2\,\overline{0}) \\
&\qquad + (p_4^0 \equiv 0) \cdot (p_4^1 \equiv 1) \cdot (p_4^2 \equiv 0)] \\
&= (p_2^0 + p_2^1) \cdot (p_4^1 + p_4^2), \\
E_{t_3} &= [(p_3^0\,\overline{1}) \cdot (p_3^1 \equiv 0) + (p_3^1\,\overline{0}) + (p_3^0 \equiv 1) \cdot (p_3^1 \equiv 0)] = p_3^0 + p_3^1.
\end{aligned}$$

For any marking in which transition t is enabled, it is necessary to effectively implement the transition firing by eliminating the corresponding tokens from any predecessor place and adding the corresponding tokens to any successor place.

The number of tokens in place p_i is represented with a vector of Boolean variables $p^0, \dots, p^{\mathrm{K_p}}$ and the number of tokens that must be subtracted is represented by another vector of Boolean constants $w^0, \dots, w^{\mathrm{K_p}}$ (either subtracting $\mathcal{W}(p_i,t)$ or $\mathcal{W}(t,p_i) - \mathcal{W}(p_i,t)$ if $\mathcal{W}(p_i,t) > \mathcal{W}(t,p_i)$). In those cases, the transition function $\delta^t(M)$ is equivalent to the subtraction of two natural numbers represented as binary vectors that can be described by the set of equations:

$$\begin{aligned}
&(\delta^{t^1}(M), \dots, \delta^{t^{\mathrm{K_p}}}(M)) = \\
&(p^0 \oplus w^0, && \text{with } B_0 = \overline{p^0} \cdot w^0 \\
&p^1 \oplus w^1 \oplus B_0, && \text{with } B_1 = \overline{p^1} \cdot w^1 + B_0 \cdot (p^1 \equiv w^1) \\
&\dots && \dots \\
&p_i^{\mathrm{K_p}} \oplus w^{\mathrm{K_p}} \oplus B_{\mathrm{K_p}-1}) && \text{with } B_{\mathrm{K_p}} = \overline{p^{\mathrm{K_p}}} \cdot w^{\mathrm{K_p}} + B_{\mathrm{K_p}-1} \\
& && \cdot (p^{\mathrm{K_p}} \equiv w^{\mathrm{K_p}}).
\end{aligned}$$

On the other hand, the number of tokens that must be added is represented by another vector of Boolean constants $w^0, \dots, w^{\mathrm{K_p}}$ (either adding $\mathcal{W}(t,p_i)$ or $\mathcal{W}(t,p_i) - \mathcal{W}(p_i,t)$ if $\mathcal{W}(t,p_i) > \mathcal{W}(p_i,t)$). In those cases, the transition function $\delta^t(M)$ is equivalent to the addition of two natural numbers that can be described by the set of equations:

$$\begin{aligned}
&(\delta^{t^1}(M), \dots, \delta^{t^{\mathrm{K_p}}}(M)) = \\
&(p^1 \oplus w^1, && \text{with } C_1 = p^1 \cdot w^1 \\
&p^2 \oplus w^2 \oplus C_1, && \text{with } C_2 = p^2 \cdot w^2 + C_1 \cdot (p^2 \oplus w^2) \\
&\dots && \dots \\
&p^{\mathrm{K_p}} \oplus w^{\mathrm{K_p}} \oplus C_{\mathrm{K_p}-1}) && \text{with } C_{\mathrm{K_p}} = p^{\mathrm{K_p}} \cdot w^{\mathrm{K_p}} + C_{\mathrm{K_p}-1} \\
& && (p^{\mathrm{K_p}} \oplus w^{\mathrm{K_p}}).
\end{aligned}$$

In summary, the transition function $\delta_i^{t^j}$ for the jth variable encoding the tokens in a place p_i is:

$$\delta_i^{t^j}(p_1, \dots, p_n) = \begin{cases}
p_i^j \oplus w^j \oplus B_{j-1} & \text{if } p_i \in {}^\bullet t \backslash t^\bullet, \\
p_i^j \oplus w^j \oplus C_{j-1} & \text{if } p_i \in t^\bullet \backslash {}^\bullet t, \\
p_i^j \oplus w^j \oplus B_{j-1} & \text{if } p_i \in t^\bullet \cap {}^\bullet t \ \wedge \ \mathcal{W}(p_i,t) > \mathcal{W}(t,p_i), \\
p_i^j \oplus w^j \oplus C_{j-1} & \text{if } p_i \in t^\bullet \cap {}^\bullet t \ \wedge \ \mathcal{W}(t,p_i) > \mathcal{W}(p_i,t), \\
p_i^j & \text{otherwise;}
\end{cases}$$

where the *carry* and *borrow* functions are defined as:

$$C_j = \begin{cases} p_i^j \cdot w^j + C_{j-1} \cdot (p_i^j \oplus w^j) & \text{if } j \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$B_j = \begin{cases} \overline{p_i^j} \cdot w^j + B_{j-1} \cdot (p_i^j \equiv w^j) & \text{if } j \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the appropriate constant value w^j must be taken for each case, where $\mathcal{W}^j$ indicates the jth bit of the natural constant $\mathcal{W}$:

$$w^j = \begin{cases}
\mathcal{W}^j(p_i,t) & \text{if } p_i \in {}^\bullet t \backslash t^\bullet, \\
\mathcal{W}^j(t,p_i) & \text{if } p_i \in t^\bullet \backslash {}^\bullet t, \\
[\mathcal{W}(p_i,t) - \mathcal{W}(t,p_i)]^j & \text{if } p_i \in t^\bullet \cap {}^\bullet t \wedge \mathcal{W}(p_i,t) > \mathcal{W}(t,p_i), \\
[\mathcal{W}(t,p_i) - \mathcal{W}(p_i,t)]^j & \text{if } p_i \in t^\bullet \cap {}^\bullet t \wedge \mathcal{W}(t,p_i) > \mathcal{W}(p_i,t).
\end{cases}$$

Image computation based on transition relations can be extended to bounded and weighted PNs by updating the characteristic function of the relation $\mathcal{R}_t$ (see (6)) into:

$$\mathcal{R}_t(q_1 \dots q_n, p_1 \dots p_n) = \prod_{i=1}^{|\mathcal{P}|} \prod_{j=0}^{\mathrm{K_{p_i}}} \Big(q_i^j \equiv \delta_i^{t^j}(p_1, \dots, p_n) \Big).$$

6.3 Boundedness Verification

The previous method works under the assumption that no place will hold more than an upper bound k of tokens. This bound can be either estimated by the designer or limited by the structure of the PN.

When using a binary encoding for places, a violation of the required boundedness condition for place p can be interpreted as an *overflow* in the operations to compute the actual number of tokens. Hence, each time a transition t is enabled to fire, the boundedness of the PN can be verified by using the specific *carry function* in each successor place of t, that is,

4. $(a \geq b)$ stands for $(a + \overline{b})$, $(a > b)$ for $(a\overline{b})$, and $(a \equiv b)$ for $(\overline{a \oplus b})$.

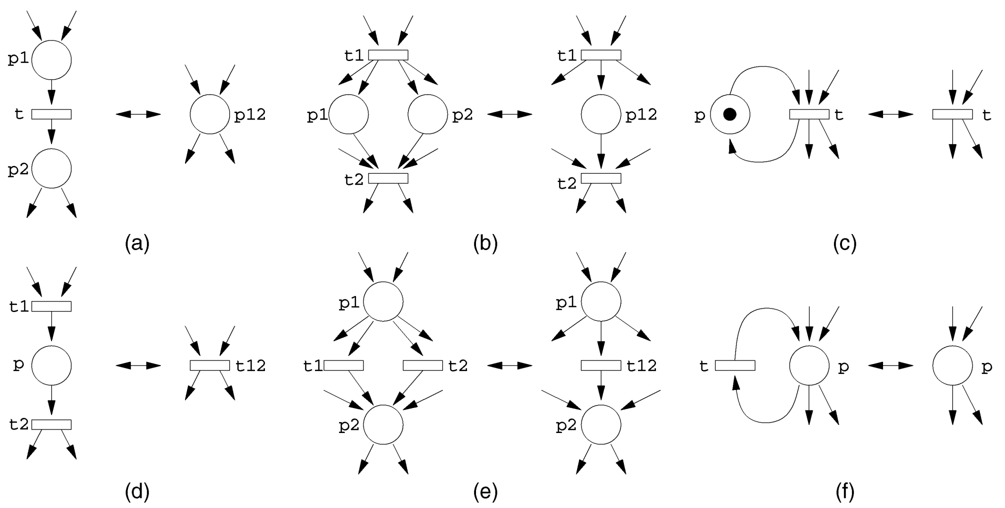


Fig. 8. Transformations preserving liveness, safeness, and boundedness of ordinary PNs.

TABLE 2
Petri Net Reductions for Ordinary PNs with Their Symbolic Inverse Transformations

Reduction Name	Symbolic Inverse Transformation
Series places fusion (a)	$R = R'_{p_{12}} \cdot (p_1 \oplus p_2) + R'_{\overline{p_{12}}} \cdot (\overline{p_1}\ \overline{p_2})$
Series transitions fusion (b)	$R = R'_{\mathrm{E}_{t_{12}}} \cdot (\mathrm{E}_{t_1} \cdot \overline{p} + \mathrm{NPM}_{t_1} \cdot p) + R'_{\overline{\mathrm{E}_{t_{12}}}} \cdot (\overline{\mathrm{E}_{t_1}} \cdot \overline{\mathrm{E}_{t_2}})$
Parallel places fusion (c)	$R = R'_{p_{12}} \cdot (p_1 p_2) + R'_{\overline{p_{12}}} \cdot (\overline{p_1}\ \overline{p_2})$
Parallel transitions fusion (d)	$R = R'$
Self-loop place (e)	$R = R' \cdot p$
Self-loop transition (f)	$R = R'$

$$\mathrm{overflow}(t) \equiv \left[\mathrm{From} \cdot \mathrm{E}_t \cdot \sum_{p \in t^\bullet} C_{\mathrm{K_p}}\right] \neq 0.$$

It is also possible to verify if a place p exceeds a particular token count N. If the upper bound N is described as a vector of constants $N^0, \ldots, N^{\mathrm{K_p}}$, the relation $M(p) > N$ is described by the equation:

$$\begin{aligned} M(p) > N \equiv\ & (p^{\mathrm{K_p}} > N^{\mathrm{K_p}}) \\ & + (p^{\mathrm{K_p}} \equiv N^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} > N^{\mathrm{K_p}-1}) \\ & + (p^{\mathrm{K_p}} \equiv N^{\mathrm{K_p}}) \cdot (p^{\mathrm{K_p}-1} \equiv N^{\mathrm{K_p}-1}) \cdot (p^{\mathrm{K_p}-2} > N^{\mathrm{K_p}-2}) \\ & + \ldots\ldots. \end{aligned}$$

Therefore, the markings that contain more tokens at p than the upper limit N are characterized by the equation:

$$B_N(p) = \sum_{i=0}^{\mathrm{K_p}} \left[(p^i > N^i) \prod_{j=i+1}^{\mathrm{K_p}} (p^j \equiv N^j) \right].$$

7 Petri Net Reductions

Petri nets can be reduced to simpler ones by using transformation rules that preserve the properties of the system being modeled. By applying these reductions, the complexity of the image computation can be effectively reduced. Among others, a set of six transformations that preserve the properties of liveness, safeness, and boundedness in ordinary PNs [9] are well-suited to be applied in the proposed symbolic framework (see Fig. 8). These transformations are a little bit more restrictive than those introduced in [9]. This section describes how these transformations can be used to reduce the number of reachable markings and the sequential depth of the PNs.

The original PN N is iteratively reduced into a smaller PN N' by applying these transformations. Once the image computation analysis has been completed on N', the set of reachable markings $[M_o\rangle$ of the original net N is derived using an inverted transformation on the set of reachable markings $[M'_o\rangle$ of N'. The inverted transformations are shown in Table 2.

As an example, Fig. 8b depicts how a PN can be reduced by *fusing* transitions t_1 and t_2 into transition t_{12} (eliminating place p). Given the reachable markings of the reduced PN (R'), the original set of reachable markings is derived as: $R = R'_{\mathrm{E}_{t_{12}}} \cdot (\mathrm{E}_{t_1} \cdot \overline{p} + \mathrm{NPM}_{t_1} \cdot p) + R'_{\overline{\mathrm{E}_{t_{12}}}} \cdot (\overline{\mathrm{E}_{t_1}} \cdot \overline{\mathrm{E}_{t_2}})$. The inverse transformation partitions the reachable markings into two sets, where t_{12} is enabled and not enabled, respectively. If t_{12} is not enabled, then neither t_1 nor t_2 can be enabled in the original net ($\overline{\mathrm{E}_{t_2}}$ implies that $p = 0$). If t_{12} is enabled, two additional situations arise. If t_1 was enabled, then $p = 0$. But, if t_1 has been just fired, all fanout places of t_1 are marked and, therefore, $p = 1$ (see NPM_{t_1} in Section 4).

```
compute_SCC_TSCC ($N$ , $[M_o\rangle$) {
    /*$\mathcal{R}_N$ Transition Relation of $N$*/
    $C_T$ := get_transitive_closure ( $\mathcal{R}_N$ );
    $C_Y$ := $C_T(x,y) \cdot C_T(y,x)$;
    $C_{NY}$ := $C_T(x,y) \cdot \overline{C_T(y,x)}$;
    InSCC := $\exists_y C_Y(x,y) \cdot [M_o\rangle$;
    InTSCC := InSCC $\cdot \overline{(\exists_y C_{NY}(x,y) \cdot [M_o\rangle)}$;
    $\mathrm{SCC}_{1 \ldots m}$ := extract_SCCs ( InSCC );
    $\mathrm{TSCC}_{1 \ldots n}$ := extract_SCCs ( InTSCC );
}
```

```
extract_SCCs ( Set(x) ) {
  $S$ := $\emptyset$;
  while (Set$(x) \neq 0$) do
    m$(x)$ := get_marking (Set(x));
    $\mathrm{S}_i(y)$ := $\exists_x (C_T(x,y) \cdot \mathrm{m}(x))$;
    rename $\mathrm{S}_i(x \leftarrow y)$;
    if ($\mathrm{S}_i(x) = 0$) remove m$(x)$ from Set$(x)$;
    else remove $\mathrm{S}_i(x)$ from Set$(x)$;
    add $\mathrm{S}_i(x)$ into $S$;
  return S;
}
```

Fig. 9. SCC and TSCC sets computation algorithm.

8 Verification of Properties

This section describes how different PN properties can be symbolically verified on the set of reachable markings. Three basic properties of PNs have been chosen as examples: *liveness*, *persistence*, and *concurrency*.

8.1 Liveness

The definition of liveness given in Section 2 can be further refined in five different levels (from L0-liveness to L4-liveness) 9. A transition t is said to be:

- L0-live (dead) if t can never be fired in any firing sequence.
- L1-live (potentially fireable) if t can be fired at least once in some firing sequence.
- L2-live if, given any positive integer k, t can be fired at least k times in some firing sequence.
- L3-live if t can be fired infinitely often in some firing sequence.
- L4-live (or simply live) if t is L1-live for every reachable marking.

Finally, a PN has a *deadlock* if there exists a marking where no transition can be fired.

The set of markings where a *deadlock* occurs can be computed as:

$$Deadlock(PN) = [M_o\rangle \cdot \prod_{t \in T} \overline{\mathrm{E}}_t.$$

The set of markings where a transition t is *potentially fireable* is computed as:

$$Fireable(t) = [M_0\rangle \cdot \mathrm{E}_t.$$

Then, if $Fireable(t) = 0$, transition t is L0-live; otherwise, it is at least L1-live.

Verifying that a transition can be fired an infinite number of times (L3-liveness) or an infinite number of times from any reachable marking of $[M_o\rangle$ (L4-liveness) requires a more complex analysis. Both problems are reduced to the computation of the *Strongly Connected Components* in the reachability graph.

A *Strongly Connected Component* (SCC) U of a directed graph $G = (V, E)$ is a maximal set of vertices $U \subseteq V$ such that, for every pair of vertices $u, v \in U$, v is reachable from u ($u \rightsquigarrow v$) and u is reachable from v ($v \rightsquigarrow u$), that is, vertices u and v are mutually reachable. A SCC is called trivial if it only contains a single vertex. A SCC U is called *terminal* (TSCC) if, from the vertices in U, it is not possible to reach any vertex outside U.

A transition t enabled in all the markings of all the TSCCs of the reachability graph is L4-live, because from any marking of $[M_o\rangle$ we will reach some TSCC_i, where t can be fired an infinite number of times. L4-liveness of transition t can be computed as follows:

$$\text{L4-live}(t) \equiv \forall i \; (\mathrm{TSCC}_i \cdot \mathrm{E}_t \neq 0).$$

A transition t is L3-live if there is an SCC_i that contains two markings, M and M', and $M[t\rangle M'$, that is, there is a cycle in which t can be fired an infinite number of times. L3-liveness for transition t can be calculated as follows:

$$\text{L3-live}(t) \equiv \sum_{\forall i} [\mathrm{SCC}_i(p_1, \ldots, p_n) \cdot \mathrm{E}_t \cdot \mathcal{R}_t \cdot \mathrm{SCC}_i(q_1, \ldots, q_n)] \neq 0.$$

The algorithm to compute the TSCCs and SCCs of a PN is shown in Fig. 9. Initially, the *Transitive Closure* (C_T) of the *Transition Relation* is computed, where $C_T(x, y) = 1$ if there is a firing sequence from x that leads to y ($x \rightsquigarrow y$) [8]. Given C_T, $C_Y = C_T(x,y) \cdot C_T(y,x)$ and $C_{NY} = C_T(x,y) \cdot \overline{C_T(y,x)}$ can be computed, where $C_Y(x,y) = 1$ if $x \rightsquigarrow y$ and $y \rightsquigarrow x$ and $C_{NY}(x,y) = 1$ if $x \rightsquigarrow y$, but there is no firing sequence leading from y to x ($y \not\rightsquigarrow x$). Next, the sets of markings that are in any SCC (*InSCC*) or in any TSCC (*InTSCC*) are computed. Finally, each individual SCC (TSCC) is obtained from *InSCC* (*InTSCC*).

The individual SCC extraction is implemented sequentially. Function *extract_SCCs* randomly takes one marking and generates its associated SSC. The SCC is calculated by multiplying the marking with the transitive closure, abstracting the variables representing the current state, and, finally, renaming the next state variables into current

```
is_persistent (N, [M_o⟩) {
  foreach $t_1 \in \mathcal{T}$ do
    Enabled := $[M_o\rangle \cdot \mathrm{E}_{t_1}$;
    foreach $t_2 \in \mathcal{T}$, $t_2 \neq t_1$ do
      To := $Img(t_2, Enabled)$;
      Not_enabled := $To \cdot \overline{\mathrm{E}_{t_1}}$;
      if (Not_enabled $\neq 0$)
        return false;
  return true;
}

TT_concurrency (N, [M_o⟩, CR) {
  /*CR is the concurrency relation.*/
  foreach $t_1 \in \mathcal{T}$ do
    foreach $t_2 \in \mathcal{T}$ do
      Enabled := $[M_o\rangle \cdot \mathrm{E}_{t_1} \cdot \mathrm{E}_{t_2}$;
      Reach_t1 := $Img(t_1, Enabled) \cdot \mathrm{E}_{t_2}$;
      Reach_t2 := $Img(t_2, Enabled) \cdot \mathrm{E}_{t_1}$;
      if (Reach_t1 $\neq 0 \wedge$ Reach_t2 $\neq 0$)
        put $(t_1, t_2)$ in $\mathcal{CR}$;
}
```

Fig. 10. Algorithms to verify PN persistence and compute transition concurrency.

state variables. Note that trivial SCCs without a reflexive arc are eliminated in this step. The SCC (or the individual marking if no SCC was generated) is eliminated from the overall set of markings. The process is repeated until all markings have been covered.

More recently, Xie et al. presented an algorithm to compute the SSC and TSSC components in a sysstem without using its transitive closure, which is the main computational step [17]. Their algorithm can be directly applied in this framework.

8.2 Persistence

A PN is said to be *persistent* if, for any two enabled transitions, the firing of one transition will not disable the other. The algorithm depicted in Fig. 10 verifies the persistence of each transition in a PN. For each transition t_1, the set of markings in which t_1 is enabled is computed. Next, the set of markings reachable in one step by firing any transition different from t_1 is obtained. If t_1 is not enabled in any of those markings, then the net is not persistent.

8.3 Concurrency Relations

The dynamic behavior of a PN is indirectly defined by analyzing which pairs of transitions are *concurrent*, i.e., if there exists a reachable marking where both transitions can fire without disabling each other. Formally, the *Concurrency Relation* is a binary relation $\mathcal{CR}$ between pairs of transitions $\mathcal{T} \times \mathcal{T}$ of the PN. Two transitions t_1 and t_2 are concurrent if there exists a marking in which both transitions are enabled and the firing of t_1 or t_2 does not disable the other, i.e.,

$$\forall t, t' \in \mathcal{T} : (t_1, t_2) \in \mathcal{CR} \;\;\Leftrightarrow\;\; \exists M \in [M_o\rangle : M[t_1t_2\rangle \wedge M[t_2t_1\rangle.$$

The algorithm to compute the pairs of transitions that are concurrent is described in Fig. 10. For each pair (t_1, t_2) that must be tested, the set of markings in which both transitions are enabled is computed as $Enabled = [M_o\rangle \cdot \mathrm{E}_{t_1} \cdot \mathrm{E}_{t_2}$. From these markings, both transitions t_1 and t_2 are independently fired. The sets of reached markings are further restricted to those in which transitions t_2 and t_1 are still enabled, i.e.,

$$Reach\_t1 := Img(t_1, Enabled) \cdot \mathrm{E}_{t_2}$$

and

$$Reach\_t2 := Img(t_2, Enabled) \cdot \mathrm{E}_{t_1}.$$

The emptiness of either of the sets indicates the non-concurrency between both transitions.

9 Experimental Results

In this section, we illustrate the power of using Boolean reasoning and BDDs for the analysis of PNs. We have chosen several scalable examples to show how the approach can easily analyze large nets without taking advantage of its regularity. Some fairly large nonscalable PNs are also included. CPU times have been obtained by executing the algorithms on a Sun ULTRA 30 workstation with 128 Mbyte main memory. We present the results corresponding to the calculation of the reachable set, which dominates the complexity of the analysis. Most properties can be verified in a straightforward manner from $[M_o\rangle$, as shown in Section 8.

We have considered three different scalable examples. The first example is the well-known *dining philosophers* paradigm, n being the number of philosophers, represented by the PN shown in Fig. 11a. The second example is the Muller's C-element pipeline depicted in Fig. 11b, n being the number of cells in the pipeline. The third models a slotted ring protocol for Local Area Networks, shown in Fig. 11c, n being the number of nodes in the network.

This section presents three sets of experiments. First, we analyze the general behavior of the symbolic traversal algorithms, the number of iterations, the evolution of the BDD sizes, and computation times. The second experiment justifies the benefits of the transition chaining heuristic. We show that this method can drastically reduce the number of iterations required to complete the traversal and indirectly the CPU computation times. The final experiment presents traversal results for a wide range of nets, comparing iterations, CPU times, and number of BDD nodes for the chained/nonchained symbolic traversal of a variety of safe and bounded nets.

A good variable ordering for the BDD manipulation is extremely important to achieve an efficient implementation. In all examples, we let the BDD package choose the

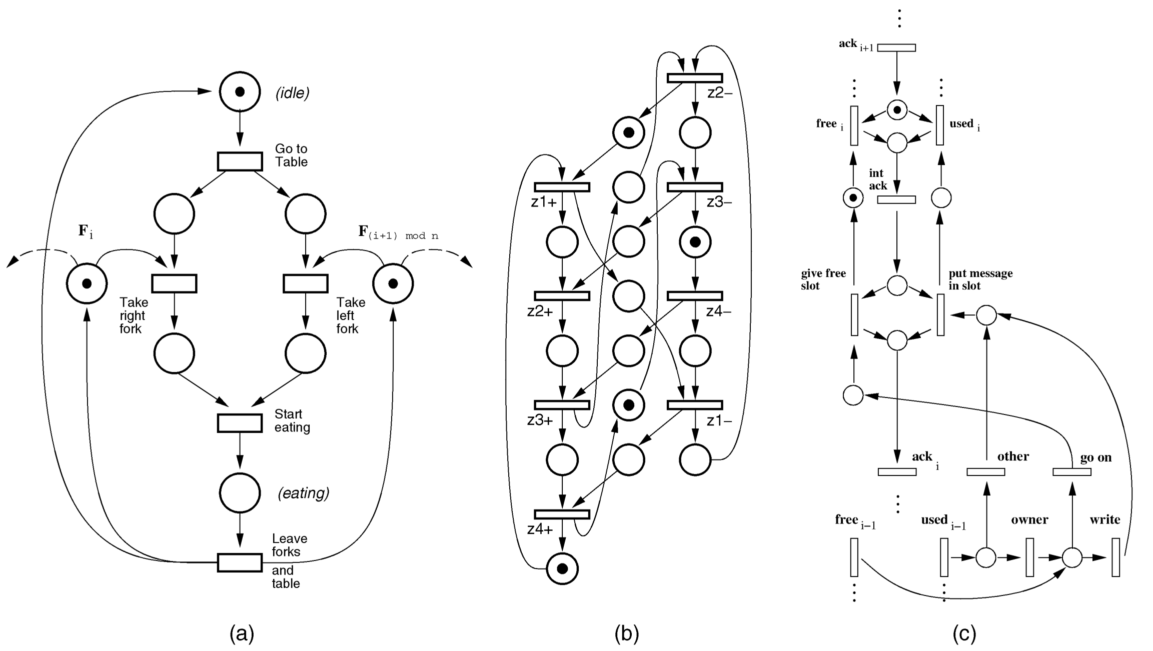


Fig. 11. Petri net specification for (a) a dining philosopher, (b) Muller's C-element pipeline, (c) slotted ring protocol for one node.

ordering with some initial support from the structure of the PN (the P-invariants of the net). We also include some comparison values between optimized orderings and nonoptimized orderings.

9.1 Symbolic Traversal Behavior

Table 3 shows the number of markings of the philosophers PN. The BDD representing $[M_o\rangle$ has been calculated by using the traversal algorithm presented in Fig. 3. Columns (maxDD) depicts the maximal number of BDD nodes required by the package and (rsDD) the final number of BDD nodes to represent the reachability set.

Fig. 12 depicts the number of reachable markings in the original PN compared to the reachable markings in the reduced net (without chaining). For that particular example, the number of markings is reduced an order of magnitude on average. Fig. 12 also depicts the number of reached markings at each iteration of the traversal algorithm for the reduced net. The slope between iterations 20 to 30 illustrates the ability of the symbolic approach to process large sets of markings in parallel.

Although the number of reached markings is small, the size of the BDD "*Reached*" at intermediate iterations can be larger than the final BDD. This is a usual phenomenon in the traversal of sequential machines using BDDs. Fig. 13 demonstrates that the peak size of the BDDs during the traversal algorithm may grow much further than the final size. Note that, in the worst case, the peak size of the number of BDD nodes is more than 10 times larger than the final size of the BDD. Fig. 13 also shows the relation between the BDD sizes in both the original PN and its reduced version. In this particular example, only a small reduction in the BDD sizes is achieved by reducing the net.

9.2 Transition Chaining

The second set of experiments is applied to the Muller's C-element pipeline. For this benchmark, we have analyzed the influence of chaining techniques. Chaining drastically reduces the number of traversal iterations required to

TABLE 3
Results for Scalable Safe PNs

name	P	T	RS	V	maxDD	rsDD	ite	CPU
muller-30	120	60	6.0×10^{7}	120	15312	2814	5	20
muller-60	240	120	8.3×10^{15}	240	109023	10251	9	604
muller-80	320	160	5.8×10^{21}	320	273963	18221	11	2818
phil-5	65	50	8.5×10^{4}	65	2518	639	3	2
phil-10	130	100	7.4×10^{9}	130	27097	7805	3	37
phil-15	195	150	6.4×10^{14}	195	306749	87419	3	700
slot-5	50	50	1.7×10^{6}	50	3072	540	5	8
slot-7	70	70	8.0×10^{8}	70	7018	1014	5	17
slot-9	90	90	3.8×10^{11}	90	9061	1632	5	43

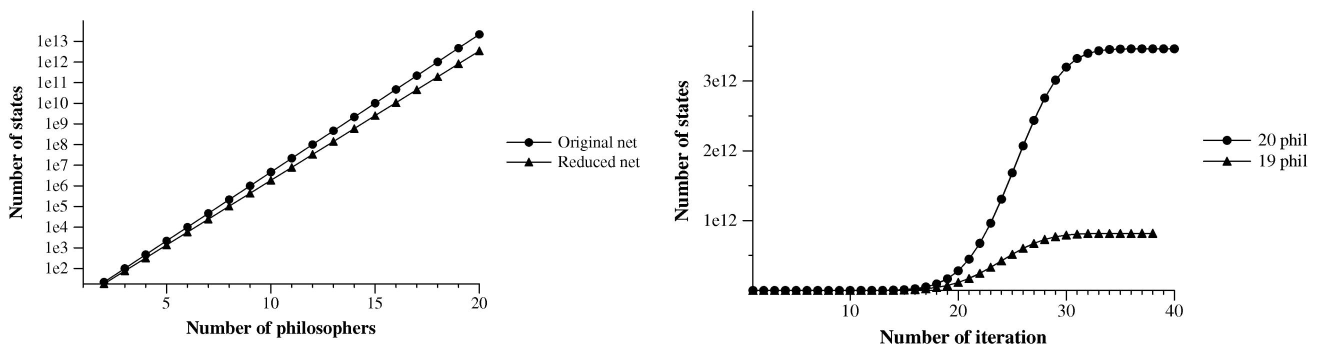


Fig. 12. Number of markings reached at each traverse iteration and the total number of markings in the original and reduced net for the dining philosophers example.

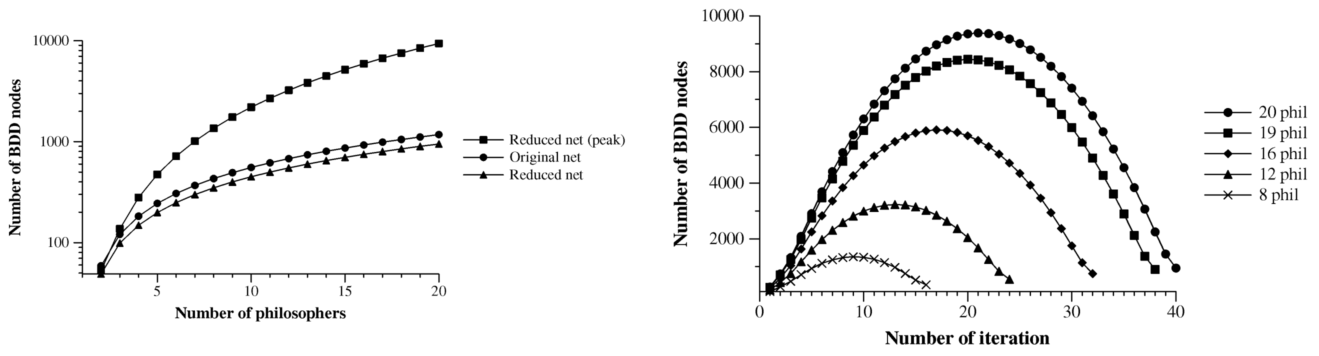


Fig. 13. Number of BDD nodes in the original and reduced net and size of the BDD "Reached" at each traverse iteration for the dining philosophers example.

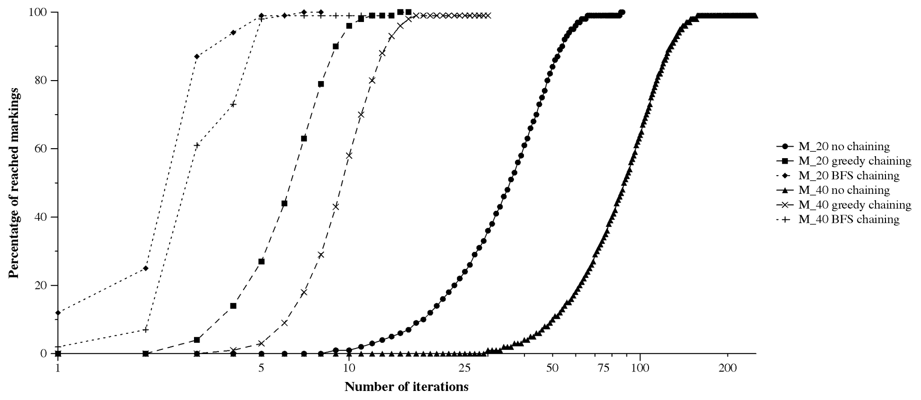


Fig. 14. Percentage of markings reached after each traverse iteration for Muller's pipeline.

compute the reachability set. Fig. 14 shows how, for the pipeline with 40 elements, the number of iterations is reduced from 250 in the initial methodology to 30 applying greedy chaining and to 12 iterations applying BFS chaining techniques.

It is also important to note that chaining techniques have a direct impact on the peak sizes of the BDDs. The addition of an extremely large number of markings at each traverse iteration offers more options for paths recombinations in the BDDs, thus drastically reducing its size. Fig. 15 shows how the BDD peak sizes are one order of magnitude smaller and, therefore, closer to the final BDD sizes. The combination of the previous advantages, reduce the number of iterations and reduce the number of BDD sizes, has a direct impact on the performance of the traversal algorithms. Fig. 16 shows how the CPU computation times are reduced several orders of magnitude, allowing to complete in less than 2×10^4 seconds the verification of a Muller's pipeline with 100 stages.

9.3 Versatility

In this section, we present experimental results to demonstrate the robustness of the proposed method. We analyze the cost of generating the reachability set for various types of PNs, both safe and bounded. For all cases, we provide the number of Boolean variables required by the encoding (V), the maximum number of BDD nodes required by the package (maxDD), the final number of BDD nodes to represent the reachability set (rsDD), the number of traverse iterations (ite), and the computation times (CPU). All tables compare the results for nonoptimized traversal and optimized traversal when using chaining techniques and allowing the BDD manager to improve the variable order.

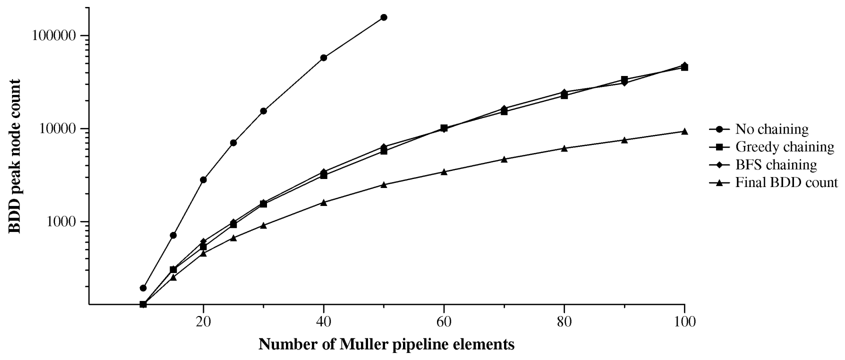


Fig. 15. BDD peak sizes for Muller's pipeline with and without applying chaining.

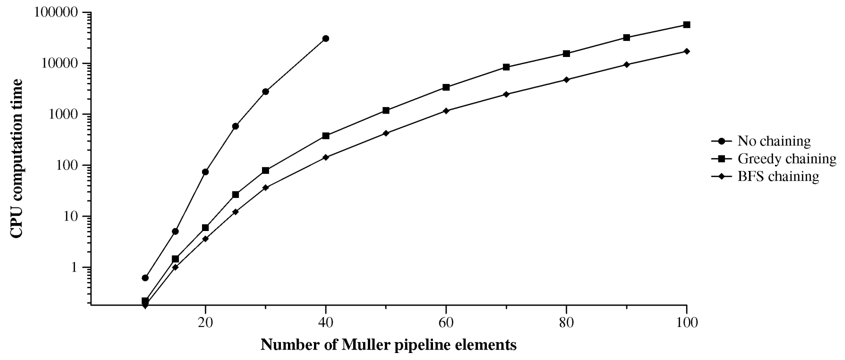


Fig. 16. CPU computation times for Muller's pipeline with and without applying chaining.

Table 3 presents optimized results for scalable safe nets, including the dining philosophers, the Muller's pipeline, and the slotted ring protocol.

Table 4 presents the results for nonscalable safe nets. The upper half of the table includes PNs modeling software programs, among others, mutual exclusion algorithms, readers and writers, etc. For these sets of PNs, results are promising, chaining appears to be quite effective, but some additional effort is needed to reduce the BDD sizes. The lower half includes PNs modeling hardware devices, including parallelizers, distributed mutual exclusion elements (DME), and registers. For this set of PNs, results are much worst. Most nonoptimized experiments do not complete due to an Out-of-Memory (OM) condition. When optimization is used, all experiments are completed, but require extremely large BDD sizes. The main reason for this behavior is that most examples include pairs of places to describe the behavior of digital signals. This type of PNs are extremely sensitive of the variable order. Note that, for all safe nets, the linear programming techniques provide enough information to determine before the symbolic traversal that the PNs are actually safe.

TABLE 4
Results for Safe PNs

PN					no-chaining				chaining & good order			
name	P	T	RS	V	maxDD	rsDD	ite	CPU	maxDD	rsDD	ite	CPU
ab_gesc	52	52	4.9×10^3	52	3175	1198	52	8	2756	1198	11	2
elevator	47	51	1.9×10^3	47	2756	1209	39	2	1885	457	9	1
mutual	49	41	3.2×10^3	49	3572	1500	53	6	2785	1477	9	2
parrow	66	48	8.0×10^4	66	17483	6342	52	49	14840	6655	7	10
sdl_arq	154	92	3.9×10^3	154	15955	9532	121	95	18525	9532	14	36
rw1w4r	125	396	1.6×10^6	125	83482	17466	79	2174	72066	10631	14	541
rw2w2r	220	1570	1.9×10^6	220	1484801	92543	96	41336	858331	63873	21	16032
par4	31	28	1.3×10^3	31	1538	371	28	1	373	90	2	1
par8	66	52	1.6×10^6	66	72332	13276	52	337	713	191	2	1
par16	130	100	2.8×10^{12}	130	OM				1401	383	2	1
DME8	137	128	3.1×10^5	137	OM				474946	265147	3	687
JJreg03	248	249	1.1×10^5	248	OM				291460	217809	20	1517
dme3	295	492	6.5×10^3	295	OM				423029	411134	20	2959

TABLE 5
Results for Bounded PNs

PN					no-chaining				chaining & good order			
name	P	T	RS	V	maxDD	rsDD	ite	CPU	maxDD	rsDD	ite	CPU
robot1	17	8	1.6×10^{2}	28	1172	389	26	1	1200	334	9	1
robot2	15	6	4.8×10^{1}	24	728	243	18	1	704	187	7	1
robot12	24	14	1.3×10^{3}	40	5327	2849	28	3	5606	2342	9	2
robot3	12	13	3.8×10^{3}	32	3864	945	30	3	3598	479	10	2
rapp-2-1-2	34	29	3.9×10^{3}	60	3842	589	21	2	2207	589	5	1
rapp-2-4-2	52	50	9.3×10^{5}	142	34757	3895	45	109	13312	3895	8	8
rapp-4-1-2	66	57	7.7×10^{8}	148	45073	2835	57	382	9187	2835	7	9
rapp-4-4-2	96	96	1.1×10^{13}	288	265995	11540	105	7044	36780	11540	10	70

Table 5 presents a mixture of results for nonscalable and scalable bounded nets. The upper half of the table includes simple PNs modeling a robot in a flexible manufacturing environment. For these sets of PNs, the number of BDD nodes required to represent the reachability set could be even larger that the number of markings. These results show that symbolic techniques have a constant overhead that makes them efficient only for highly concurrent PN, where the implicit parallelism of the method can be fully exploited. The second set of scalable PNs models a pipelined implementation of the instruction decoder of a real microprocessor [16], the suffix in the name indicating the number of columns, rows, and length of each cell. This example shows that complex bounded PNs (on average, three bits are used to encode each place) can be analyzed in reasonable computation times.

10 Conclusions and Future Work

This paper presents the combination of Boolean reasoning and BDD algorithms to manage the *state explosion* produced in Petri net analysis. It has been shown that BDDs can efficiently represent and manipulate large sets of reachable markings with a small number of BDD nodes. Once the reachable markings have been generated, several properties, such as safeness, boundedness, liveness, persistence, and concurrency, can be verified since they are reduced to the computation of well-known Boolean formulas. Therefore, BDDs are proposed as an alternative to the reachability tree or other enumerative techniques, providing a compact representation of the markings of any bounded PN.

Many issues are still under research to increase the applicability of the approach. The ordering of variables is a topic of major interest that must be studied in order to reduce even more the size of the BDDs, thus speeding-up BDD operations. As mentioned in Section 6, the encoding methods for k-bounded nets must be further explored. The combination of symbolic methods with structural and linear programming techniques as described in [11], more powerful reduction methodologies, improved chaining traversal, and more compact BDD variable encoding are clear areas for future development.

Acknowledgments

This work has been supported in part by CICYT (grant TIC98-0410-CO2-01).

References

[1] F.M. Brown, *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic, 1990.

[2] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys,* vol. 24, no. 3, pp. 293-318, Sept. 1992.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Proc. Fifth Ann. Symp. Logic in Computer Science,* June 1990.

[4] J.M. Colom and M. Silva, "Improving the Linearly Based Characterization of P/T Nets," *Proc. Advances in Petri Nets 1990,* G. Rozenberg, ed., pp. 113-145, 1991.

[5] O. Coudert, C. Berthet, and J.C. Madre, "Verification of Sequential Machines Using Boolean Functional Vectors," *Proc. IFIP Int'l Workshop Applied Formal Methods for Correct VLSI Design,* pp. 111-128, Nov. 1989.

[6] O. Coudert, C. Berthet, and J.C. Madre, "Formal Boolean Manipulations for the Verification of Sequential Machines," *Proc. European Conf. Design Automation (EDAC),* pp. 57-61, Mar. 1990.

[7] K. Hamaguchi, H. Hiraishi, and S. Yajima, "Design Verification of Asynchronous Sequential Circuits Using Symbolic Model Checking," *Proc. Int'l Symp. Logic Synthesis and Microprocessor Architecture,* pp. 84-90, July 1992.

[8] Y. Matsunaga, P.C. McGeer, and R.K. Brayton, "On Computing the Transitive Closure of a State Transition Relation," *Proc. ACM/IEEE Design Automation Conf.,* pp. 260-265, June 1993.

[9] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, pp. 541-574, Apr. 1989.

[10] E. Pastor, O. Roig, J. Cortadella, and R.M. Badia, "Petri Net Analysis Using Boolean Manipulation," *Proc. 15th Int'l Conf. Application and Theory of Petri Nets,* pp. 416-435, June 1994.

[11] E. Pastor, J. Cortadella, and M.A. Peña, "Structural Methods to Improve the Symbolic Analysis of Petri Nets," *Proc. 20th Int'l Conf. Application and Theory of Petri Nets,* pp. 26-45, June 1999.

[12] C.A. Petri, "Kommunikation mit Automaten," PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962 (technical report Schriften des IIM Nr. 3).

[13] W. Reisig, *Petri Nets, An Introduction.* Springer-Verlag, 1985.

[14] O. Roig, J. Cortadella, and E. Pastor, "Verification of Asynchronous Circuits by BDD-Based Model Checking of Petri Nets," *Proc. 16th Int'l Conf. Application and Theory of Petri Nets,* pp. 374-391, June 1995.

[15] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design,* Nov. 1993.

[16] A. Xie, S. Kim, and P.A. Beerel, "Bounding Average Time Separations of Events in Stochastic Timed Petri Nets with Choice," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems,* pp. 94-107, Apr. 1999.

[17] A. Xie and P.A. Beerel, "Implicit Enumeration of Strongly Connected Components," *Proc. Int'l Conf. Computer-Aided Design,* pp. 37-40, Nov. 1999.

[18] T. Yoneda, H. Hatori, A. Takahara, and S. Minato, "BDDs vs. Zero-Suppressed BDDs: For CTL Symbolic Model Checking of Petri Nets," *Proc. Formal Methods in Computer-Aided Design,* pp. 435-449, 1996.

Enric Pastor received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1991 and 1996, respectively. He is an associate professor in the Department of Computer Architecture at the Universitat Politècnica de Catalunya. He was a visiting scholar at the University of Colorado at Boulder and the Inter-university Microelectronics Centre (IMEC), Belgium, in 1992 and 1994, respectively. In 1998, he was a Leverhulme Trust Fellow visiting the University of Newcastle upon Tyne, United Kingdom. His research interests include formal methods for the computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits and concurrent systems.

Jordi Cortadella received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively. He is a professor in the Department of Software of the same university. In 1988, he was a visiting scholar at the University of California, Berkeley. His research interests include computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits, concurrent systems, co-design, and parallel architectures. He has coauthored more than 80 research papers in technical journals and conferences. He has served on the technical committees of several international conferences in the field of design automation and concurrent systems. Dr. Cortadella organized the Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems as a symposium cochair. He is a member of the IEEE.

Oriol Roig (M'98) received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1991 and 1997, respectively. He is at Theseus Logic, Sunnyvale, California, where he leads the EDA Software Development team. Prior to joining Theseus, he was at National Semiconductor as a member of the Methodology Group and at the Universitat Politècnica de Catalunya as an assistant professor. His primary research interests include various aspects of computer-aided design, with emphasis on formal verification, testing, and asynchronous circuit design. He is a member of the IEEE.