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 Decision 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.

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 $R$ between $A$ and $B$ is a subset of the Cartesian product $A \times B$. We write $xRy$ if $x$ and $y$ are in relation $R$. 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 $(B, +, \cdot, 0, 1)$, 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 $(B, +, \cdot, 0, 1)$, with $B = \{0, 1\}$, $+$, 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 $f : B^n \rightarrow B$. Let $F_n(B)$ be the set of $n$-variable Boolean functions, then $F_n(B)$ is the power set of $B^n$ (the characteristic functions of subsets of $B^n$).

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

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

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 $f_0 = f(x_1, \ldots, x_{n-1}, 1, x_{n+1}, \ldots, x_n)$ and $f_{\overline{x}} = f(x_1, \ldots, x_{n-1}, 0, x_{n+1}, \ldots, x_n)$ are called the positive and negative cofactors of $f$ with respect to $x_i$. If $f : B^n \rightarrow B$ is an $n$-variable Boolean function, then Boole’s Expansion Theorem defines that

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

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

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 $N = (P, T, W, M_0)$, where $P = \{p_1, \ldots, p_n\}$ and $T = \{t_1, \ldots, t_m\}$ are finite sets of places and transitions (also called nodes) satisfying $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is the weighted flow relation, and $M_0$ is the initial marking. A marking is a function $M : P \rightarrow \mathbb{N}$. If $k$ is assigned to a place $p$ by $M$, we will say that $p$ is marked with $k$ tokens in $M$. If $W(u, v) > 0$, then there is an arc from $u$ to $v$ with weight $W(u, v)$. The pre and postset of a node are specified by a dot-notation: $u = \{v \in P \cup T \mid W_U(v, u) > 0\}$ is the preset of $u$ and $u^* = \{v \in P \cup T \mid W_U(v, u) > 0\}$ is 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 $M(t)$) if $\forall p \in \mathfrak{A} : M(p) \geq W(p, t)$. Once a transition $t$ is enabled at a marking $M$, it may fire, reaching a new marking $M'$ (denoted by $M(t)M'$), where $M'(p) = M(p) - W(p, t) + W(t, p)$.

A sequence of transitions $\sigma = t_1 \ldots t_k \in T^*$ is a firing sequence from a marking $M_1$ to a marking $M_k$ iff there exist markings $M_0, \ldots, M_{k-1}$ such that: $M_i[t_i]M_{i+1}$ for $1 \leq i \leq k-1$. Marking $M_0$ is said to be reachable from $M_k$ by firing $\sigma : M_k[\sigma]M_0$. $\{M\}$ is the set of markings reachable from $M$ by firing any sequence of transitions, i.e., $M' \in \{M\} \Leftrightarrow \exists \sigma \in T^* : M(\sigma)M'$. $\{M\}$ is the set of all markings reachable from $M_k$. The automaton that contains the set of reachable markings and all possible firing sequences of a PN is called the reachability graph. A transition $t \in T$ is live iff

$$\forall M \in \{M\} : \exists M' \in \{M\}$$

A PN is live iff every transition in the PN is live. A marking $M \in \{M\}$ is a home marking iff $\forall M' \in \{M\} : M \in \{M'\}$. A place
Additionally, any set of markings in \( M \in \mathcal{M} \) can be represented by a set of places \( m \) 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 \( p \in \mathcal{P} \) denotes the fact that there is a token in \( p \). Additionally, any set of markings in \( \mathcal{M} \) 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 PN.

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 \((2^M, \cup, \cap, \emptyset, M_p)\) 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 \( E : M_p \rightarrow B^n \). The image of every marking \( M \in M_p \) is encoded into a vertex \((p_1, \ldots, p_n) \in B^n \) 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 \((0, 1, 0, 0, 0, 0, 0, 1) \in B^7 \) and the cube \( \overline{p_1} p_2 \overline{p_3} \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} \) represent the marking in which \( p_2 \) and \( p_7 \) are marked and \( p_1, p_3, p_4, p_5, p_6 \) are not marked.

Extending the use of the encoding function \( E \), each set of markings \( M \in 2^M_p \) has a corresponding image \( V \in F_n(B) \) according to \( E \), defined by:

![Fig. 1. (a) A safe Petri net with its initial marking, (b) its corresponding reachability graph.](image-url)
Then, the characteristic function of the set \( M \) is a function \( \chi_M : B^n \rightarrow B \) that evaluates to 1 for those vertices that correspond to markings belonging to \( M \); that is, \( \chi_M = \chi_V \). From now on, and for the sake of simplicity, we will indistinctly use \( M \) and \( \chi_M \) to denote the characteristic function of a set of markings \( M \). All set manipulations can be applied as Boolean operations on \( M \) and \( \chi_M \).

For example, given the sets of markings \( M_1, M_2 \in M_P \), union \( \chi_{M_1} \cup \chi_{M_2} = \chi_{M_1} + \chi_{M_2} \), intersection \( \chi_{M_1 \cap M_2} = \chi_{M_1} \cdot \chi_{M_2} \), and complement \( \chi_{M^c} = \bar{\chi}_M \) can be implemented with the corresponding Boolean operators.

As an example, the set of markings

\[
M = \left\{ \{p_2, p_1\}, \{p_2, p_3\}, \{p_4, p_7\} \right\}
\]

has the characteristic function:

\[
\chi_M = \bar{p}_1 \cdot p_2 + \bar{p}_2 \cdot \bar{p}_3 + p_6 \cdot (p_3 \lor p_4) + \bar{p}_1 \cdot \bar{p}_3 \cdot p_6 + p_7 \cdot (p_2 \lor p_4) \cdot \bar{p}_1 = \bar{p}_2 \cdot p_3 + p_7 \cdot \bar{p}_1.
\]

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

\[
\forall(p_1, \ldots, p_n), (q_1, \ldots, q_n) \in B^n \chi_R(p_1, \ldots, p_n; q_1, \ldots, q_n) = 1 \iff \exists(M, M') \in R : (E(M) = (p_1, \ldots, p_n) \land E(M') = (q_1, \ldots, q_n)).
\]

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

\[
R_M = \left\{ M \in M \mid \exists M' \in M' : (M, M') \in R \right\},
\]

and its characteristic function \( \chi_{R_M} \) is computed as:

\[
\chi_{R_M}(p_1, \ldots, p_n) = \exists q_1, \ldots, q_n \chi_R(p_1, \ldots, p_n; q_1, \ldots, q_n).
\]

### 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 \( E_t \subseteq M_P \) be the set of markings in which transition \( t \) is enabled. The transition function for a transition \( t \in T \) is a partially defined function \( \delta^t : B^n \rightarrow B^n \) that transforms every marking \( M \in E_t \) into a new marking \( M' \in M_P \) by firing transition \( t \), i.e., \( M' = \delta^t(M) \). The image of \( \delta^t \) for markings outside \( E_t \) is undefined.

This concept is equivalent to the one-step reachability in PNs. The transition function \( \delta^t = (\delta_{1}^t, \ldots, \delta_{|P|}^t) \) for a transition \( t \in T \) 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 \( \forall i \in 1, \ldots, |P| \):

\[
\delta_i^t(p_1, \ldots, p_n) = \begin{cases} 
1 & \text{if } p_i \in t^* \\
0 & \text{if } p_i \in t^* \text{ and } p_i \notin t^* \\
p_i & \text{otherwise.}
\end{cases} \tag{2}
\]

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

\[
E_t = \bigcap_{p \in t^*} p_i. \tag{3}
\]

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 \( M_0 \) if there is a sequence of markings \( M_1, M_2, \ldots, M_{k-1} \) and a sequence of transitions \( t_1, t_2, \ldots, t_k \), such that \( \delta_i^t(M_{i-1}) = M_i, \forall i, 1 \leq i \leq k \). Following the example in Fig. 1 that requires the Boolean variables \( p_1, \ldots, p_7 \), the transition and enabling functions are:

\[
\begin{align*}
\delta_1^t(M) &= (0, 1, 1, p_4, p_5, p_6, p_7), \\
\delta_2^t(M) &= (0, p_2, p_3, 1, 1, p_6, p_7), \\
\delta_3^t(M) &= (p_1, 0, p_3, p_4, p_5, 1, p_7), \\
\delta_4^t(M) &= (p_1, p_2, 0, p_4, p_5, p_6, 1), \\
\delta_5^t(M) &= (p_1, p_2, p_3, 0, p_5, 1, p_7), \\
\delta_6^t(M) &= (1, p_2, p_3, p_4, p_5, 0, 0), \\
\delta_7^t(M) &= (1, p_2, p_3, p_4, p_5, p_6, p_7).
\end{align*}
\]

Therefore, firing transition \( t_1 \) from markings

\[
\begin{align*}
p_1 \quad &p_2 \quad &p_3 \quad &p_4 \quad &p_5 \quad &p_6 \quad &p_7
\end{align*}
\]

and

\[
\begin{align*}
p_1 \quad &p_2 \quad &p_3 \quad &p_4 \quad &p_5 \quad &p_6 \quad &p_7
\end{align*}
\]

where \( t_1 \) is enabled, results in:

\[
\begin{align*}
p_1 \quad &p_2 \quad &p_3 \quad &p_4 \quad &p_5 \quad &p_6 \quad &p_7
\end{align*}
\]

and

\[
\begin{align*}
p_1 \quad &p_2 \quad &p_3 \quad &p_4 \quad &p_5 \quad &p_6 \quad &p_7
\end{align*}
\]

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 \( \delta^t \) as a function that transforms a set of markings \( M \) into the set of markings \( M' \) that can be reached from \( M \) by firing \( t \). The constrained image of \( \delta^t \) is denoted by \( \text{Img}(t, M) \) and computed:

\[
\text{Img}(t, M) = \left\{ M' \in M_P : \exists M \in M \land E_t, \delta^t(M) = M' \right\}. \tag{4}
\]

Using the terminology for verification of sequential machines, function \( \text{Img} \) 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 $t \in T$:

$$NPM_t = \prod_{p_i \in \hat{E}_t} (\text{no predecessor of } t \text{ is marked}),$$

$$ASM_t = \prod_{p_i \in \hat{C}_t} (\text{all successors of } t \text{ are marked}),$$

$$NSM_t = \prod_{p_i \in \hat{M}_t} (\text{no successor of } t \text{ is marked}).$$

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

$$Img(t, M) = (\bigcap_{t \in \hat{E}_t} NPM_t) \cdot ASM_t. \tag{5}$$

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

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

we will calculate $M' = \delta(M, t_1)$. First, $M_{E_{t_1}}$ (the cofactor of $M$ with respect to $E_{t_1} = p_2$) selects those markings in which $t_2$ is enabled$^1$ and removes its predecessor places from the characteristic function:

$$M_{E_{t_1}} = \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} \overline{p_4} p_5 \overline{p_6} \overline{p_7}.$$  

Then, the product with $NPM_{t_2} = \overline{p_2}$ simulates the elimination of tokens in the predecessor places:

$$M_{E_{t_1}} \cdot NPM_{t_2} = \overline{p_2} \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} \overline{p_4} p_5 \overline{p_6} \overline{p_7}.$$  

Next, the cofactor with respect to $NSM_{t_3} = \overline{p_4}$ removes all successor places from the characteristic function:

$$(M_{E_{t_1}} \cdot NPM_{t_2})_{NSM_{t_3}} = \overline{p_2} \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} \overline{p_4} p_5 \overline{p_6} \overline{p_7}.$$  

Finally, the product with $ASM_{t_4} = p_6$ adds a token to all successor places of $t_5$:

$$(M_{E_{t_1}} \cdot NPM_{t_2})_{NSM_{t_3}} \cdot ASM_{t_4} = \overline{p_2} \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} p_3 \overline{p_4} \overline{p_5} \overline{p_6} \overline{p_7} + \overline{p_2} \overline{p_4} p_5 \overline{p_6} \overline{p_7},$$

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

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

$$R_t(p_1, \ldots, p_n, q_1, \ldots, q_n) = \prod_{i=1}^{\mid P \mid} (q_i \equiv \delta_i(p_1, \ldots, p_n)) \cdot E_t.$$  

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

$$Img(t, M) = \exists_{p_1, \ldots, p_n} [R_t(p_1, \ldots, p_n, q_1, \ldots, q_n) \cdot M]. \tag{6}$$

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

$$R_{t_1}(p_1, \ldots, p_r, q_1, \ldots, q_r) = (\overline{q_1} \cdot q_2 \cdot q_3 \cdot (q_4 \equiv p_4) \cdot (q_5 \equiv p_5) \cdot (q_6 \equiv p_6) \cdot (q_7 \equiv p_7)). \tag{7}$$

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

$$Img(PN, M) = \exists_{p_1, \ldots, p_n} \sum_{t \in T} \prod_{i=1}^{\mid P \mid} (q_i \equiv \delta_i(p_1, \ldots, p_n)) \cdot E_t \cdot M.$$  

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

The main computational problem in image computation with the transition relation method appears when taking the conjunction $\prod_{i=1}^{\mid P \mid}$. Even if the BDDs for $q_i$ and the final result $R_t$ 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 $M_0$ 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].
reachability set can be obtained by computing the least fix point of the following recurrence:

\[
\begin{align*}
S_0 &= M_0 \\
S_{i+1} &= S_i \cup \text{Img}(PN, S_i).
\end{align*}
\]

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 \(i\)th iteration of the outermost loop in the algorithm, the traversal obtains all markings reachable in one step from the set \(\text{From}_i\) (the index in the set denotes the iteration number). The algorithm applies the \(\text{Img}\) function for every transition in the PN (see Fig. 3). Only those markings that are new in the set of reachable markings (set \(\text{New}_i\)) are considered for the next iteration. Hence, the set \(\text{New}_i\) is transformed into the set \(\text{From}_{i+1}\) for the next iteration. The accumulation of sets of new markings (\(\text{New}_i\)) results in a monotonically increasing set of reachable markings (\(\text{Reached}_i\)). 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 \(M_0\) to itself.

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

\[
\begin{align*}
\text{To} &= \{\{p_2p_3\}, \{p_4p_5\}\} \\
\text{New} &= \{\{p_2p_3\}, \{p_4p_5\}\} - \{\{p_1\}\} = \{\{p_2p_3\}, \{p_4p_5\}\}.
\end{align*}
\]

and

\[
\text{Reached} = \{\{p_1\}, \{p_2p_3\}, \{p_4p_5\}\}.
\]

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

\[
\exists (M \in [M_0], t \in \mathcal{T}, p \in \mathcal{P}) : [M(t) \land p \in \mathcal{T} \land p \notin \mathcal{T} \land M(p) = 1].
\]

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 \( \text{From} \) 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:

\[
\forall t \in \mathcal{T} : \left[ \text{From} \cdot E_t \cdot \sum_{(p \in \mathcal{T}) \land (p \notin \mathcal{T})} p \right] = 0.
\]

### 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 \( \text{Img} \) 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 traversal.

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 (1), \( t_1 \) and \( t_2 \) will fire, reaching \( M_2 \) and \( M_3 \) respectively. In a second iteration (2), \( t_1 \) and \( t_2 \) will fire again, both reaching \( M_4 \). Finally, in the third iteration (3), \( t_3 \) 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 \( \text{Img} \) 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.

![Fig. 5. Symbolic PN traversal applying transition chaining.](image)

![Fig. 6. Application example for the chaining reachability techniques.](image)
TABLE 1

<table>
<thead>
<tr>
<th># tokens</th>
<th>one-hot encoding</th>
<th>binary encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p_0)</td>
<td>(p_0)</td>
</tr>
<tr>
<td>1</td>
<td>(p_0^1)</td>
<td>(p_1^0)</td>
</tr>
<tr>
<td>2</td>
<td>(p_0^2)</td>
<td>(p_2^0)</td>
</tr>
<tr>
<td>3</td>
<td>(p_0^3)</td>
<td>(p_3^0)</td>
</tr>
</tbody>
</table>

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 \(\mathcal{F}(t)\) 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, \(p^0, \ldots, p^{k_p}\), 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 \(p^0, \ldots, p^{k_p}\) (for \(K_p = \lceil \log_2(k+1) \rceil - 1\)). For any natural number \(N\) such that \(N = \sum_{i=0}^{k_p} n_i 2^i\), then:

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

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 \(p_1p_2^1p_3^0p_4^3\).

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:

\[
\delta'(p_1, \ldots, p_n) = \begin{cases} M(p_t) - W(p_t, t) & \text{if } p_t \in t \setminus t' \setminus t, \\ M(p_t) + W(t, p_t) & \text{if } p_t \in t' \setminus t', \\ M(p_t) - W(p_t, t) + W(t, p_t) & \text{if } p_t \in t' \cap t, \\ M(p_t) & \text{otherwise}. \end{cases}
\]

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

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

\[
M(p) \geq W(p, t) \equiv (p^{K_p} > w^{K_p}) + (p^{K_p} = w^{K_p}) \cdot (p^{K_p-1} > w^{K_p-1}) + (p^{K_p} = w^{K_p}) \cdot (p^{K_p-1} = w^{K_p-1}) \cdot (p^{K_p-2} > w^{K_p-2}) + \ldots + (p^{K_p} = w^{K_p}) \cdot (p^{K_p-1} = w^{K_p-1}) \cdot \ldots \cdot (p^1 = w^1) \cdot (p^0 > w^0) + (p^0 \equiv w^0).
\]

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

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

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

\[
E_{t_1} = [(p^1 \equiv 1) + (p^1 \equiv 0) \cdot (p^1 \equiv 1)] = p^1
\]

\[
E_{t_2} = [(p^2 \equiv 1) \cdot (p^2 \equiv 0) + (p^2 \equiv 1) \cdot (p^2 \equiv 0)]
\]

\[
= [(p^2 \equiv 0) \cdot (p^2 \equiv 0) + (p^2 \equiv 1) \cdot (p^2 \equiv 1)]
\]

\[
= (p^2 + \overline{p^2}),
\]

\[
E_{t_3} = [(p^3 \equiv 1) \cdot (p^3 \equiv 0) + (p^3 \equiv 1) \cdot (p^3 \equiv 0)]
\]

\[
= (p^3 + \overline{p^3}).
\]

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 \( p^0, \ldots, p^{K_p} \) and the number of tokens that must be subtracted is represented by another vector of Boolean constants \( w^0, \ldots, w^{K_p} \) (either subtracting \( W(p, t) \) or \( W(t, p) - W(p, t) \) if \( W(p, t) > W(t, p) \)). 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:

\[
\delta^t(M) = \begin{cases} 
(\overline{p^0} \oplus w^0), & \text{with } B_0 = \overline{p^0} \cdot w^0; \\
(p^1 \oplus w^1) \cdot B_0, & \text{with } B_1 = \overline{p^1} \cdot w^1 + B_0 \cdot (p^1 \equiv w^1); \\
\ldots \\
(p^{K_p} \oplus w^{K_p} \cdot B_{K_p-1}), & \text{with } B_{K_p} = \overline{p^{K_p}} \cdot w^{K_p} + B_{K_p-1}. 
\end{cases}
\]

On the other hand, the number of tokens that must be added is represented by another vector of Boolean constants \( w^0, \ldots, w^{K_p} \) (either adding \( W(p, t) \) or \( W(t, p) - W(p, t) \) if \( W(t, p) > W(p, t) \)). In those cases, the transition function

\[
\delta^{t'}(M) \text{ is equivalent to the addition of two natural numbers that can be described by the set of equations:}
\]

\[
\delta^{t'}(M) = \begin{cases} 
(p^1 \oplus w^1), & \text{with } C_1 = p^1 \cdot w^1; \\
p^2 \oplus w^2 \oplus C_1, & \text{with } C_2 = p^2 \cdot w^2 + C_1 \cdot (p^2 \oplus w^2); \\
\ldots \\
p^{K_p} \oplus w^{K_p} \cdot B_{K_p-1}), & \text{with } C_{K_p} = p^{K_p} \cdot w^{K_p} + C_{K_p-1}. 
\end{cases}
\]

In summary, the transition function \( \delta^{t'}(p_1, \ldots, p_n) = \begin{cases} 
p_i^j \oplus w^j + B_{j-1} \cdot (p_i^j \equiv w^j) \text{ if } p_i \in *t \cdot t^*, \\
(p_i^j \equiv w^j) \text{ if } p_i \in t^* \cdot t, \\
(p_i^j \oplus w^j + B_{j-1} \cdot (p_i^j \equiv w^j) \text{ if } p_i \in *t \cap t \wedge W(p_i, t) > W(t, p_i), \\
0 \text{ if } p_i \in t^* \cap t \wedge W(t, p_i) > W(p_i, t), \\
0 \text{ otherwise;}
\end{cases} \)

where the carry and borrow functions are defined as:

\[
\begin{align*}
C_j &= \begin{cases} 
p_i^j \cdot w^j + C_{j-1} \cdot (p_i^j \equiv w^j) \text{ if } j \geq 0, \\
0 \text{ otherwise}
\end{cases} \\
B_j &= \begin{cases} 
p_i^j \cdot w^j + B_{j-1} \cdot (p_i^j \equiv w^j) \text{ if } j \geq 0, \\
0 \text{ otherwise.}
\end{cases}
\]

Finally, the appropriate constant value \( w^j \) must be taken for each case, where \( W^j \) indicates the \( j \)th bit of the natural constant \( W \):

\[
W^j = \begin{cases} 
W^j(p_i, t) \text{ if } p_i \in *t \cdot t^*; \\
W^j(t, p_i) \text{ if } p_i \in t^* \cdot t, \\
(W^j(t, p_i) - W^j(p_i, t))^j \text{ if } p_i \in *t \cap t \wedge W(p_i, t) > W(t, p_i), \\
(W^j(t, p_i) - W^j(p_i, t))^j \text{ if } p_i \in t^* \cap t \wedge W(t, p_i) > W(p_i, t).
\end{cases}
\]

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

\[
R_e(q_1 \ldots q_n, p_1 \ldots p_n) = \prod_{i=1}^{n} \prod_{j=0}^{K_p} (q_i^j \equiv \delta_i^j(p_1, \ldots, p_n)).
\]

### 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,
overflow(t) = \left\{ \begin{array}{l} \text{From } E_t \cdot \sum_{j \in I_t} G_{i_j} \neq 0. \end{array} \right.

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 $N_0, \ldots, N_{K_p}$, the relation $M(p) > N$ is described by the equation:

\[ M(p) > N \equiv (p^{K_0} > N^{K_0}) + (p^{K_0} \equiv N^{K_0}) \cdot (p^{K_0-1} > N^{K_0-1}) + (p^{K_0} \equiv N^{K_0}) \cdot (p^{K_0-1} \equiv N^{K_0-1}) \cdot (p^{K_0-2} > N^{K_0-2}) + \ldots. \]

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

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

### 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 $[M_o]$ of the original net $N$ is derived using an inverted transformation on the set of reachable markings $[M_o]$ 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:

\[ R = R_{E_{t_{12}}} \cdot (E_{t_1} \cdot \overline{p} + NPM_{t_1} \cdot p) + R_{E_{t_{12}}} \cdot (E_{t_2} \cdot \overline{p}). \]

The inverse transformation partitions the reachable markings into two sets, where $t_{12}$ is enabled but not enabled, respectively. If $t_{12}$ is not enabled, then neither $t_1$ nor $t_2$ can be enabled in the original net ($E_{t_{12}}$ 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).

---

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

**Table 2**

<table>
<thead>
<tr>
<th>Reduction Name</th>
<th>Symbolic Inverse Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series places fusion (a)</td>
<td>$R = R_{E_{t_{12}}} \cdot (p_1 + p_2) + R_{E_{t_{12}}} \cdot (p_1 \overline{p})$</td>
</tr>
<tr>
<td>Series transitions fusion (b)</td>
<td>$R = R_{E_{t_{12}}} \cdot (E_{t_{12}} \cdot \overline{p} + NPM_{t_{12}} \cdot p) + R_{E_{t_{12}}} \cdot (E_{t_1} \cdot \overline{E_{t_2}})$</td>
</tr>
<tr>
<td>Parallel places fusion (c)</td>
<td>$R = R_{E_{t_{12}}} \cdot (p_1 \overline{p}) + R_{E_{t_{12}}} \cdot (\overline{p_1} \overline{p})$</td>
</tr>
<tr>
<td>Parallel transitions fusion (d)</td>
<td>$R = R'$</td>
</tr>
<tr>
<td>Self-loop place (e)</td>
<td>$R = R' \cdot p$</td>
</tr>
<tr>
<td>Self-loop transition (f)</td>
<td>$R = R'$</td>
</tr>
</tbody>
</table>

---
compute_SCC,TSCC (N, [M_o]) {
    /* \* R_N Transition Relation of N*/
    C_T := get_transition_closure (R_N);
    C_Y := C_T(x,y) · CT(y,x);
    C_NY := C_T(x,y) · CT(y,0);
    InSCC := \exists_y C_Y(x, y) · [M_o];
    InTSCC := InSCC · \exists_y C_NY(x, y) · [M_o];
    SCC1..m := extract_SCCs (InSCC);
    TSCC1..m := extract_SCCs (InTSCC);
}

extract_SCCs (Set(x)) {
    S := \emptyset;
    while (Set(x) \neq 0) do
        m(x) := get_marking (Set(x));
        S_i(y) := \exists_z (C_T(x,y) · m(x));
        rename S_i(x) \to y;
        if (S_i(x) = 0) remove m(x) from Set(x);
        else remove S_i(x) from Set(x);
        add 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). A transition \( t \) is said to:

- 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:

\[
\text{Deadlock}(PN) = [M_o] \times \prod_{t \in T} E_t .
\]

The set of markings where a transition \( t \) is potentially fireable is computed as:

\[
\text{Fireable}(t) = [M_o] \times E_t .
\]

Then, if \( \text{Fireable}(t) = 0 \), 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 \([M_o]\) (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 \( u \) (\( u \leadsto v \)) and \( u \) is reachable from \( v \) (\( v \leadsto u \)), 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 \([M_o]\) we will reach some TSCC, where \( t \) can be fired an infinite number of times. L4-liveness of transition \( t \) can be computed as follows:

\[
\text{L4-live}(t) \equiv \forall i \: (\text{TSCC}_i \cdot E_t \neq 0).
\]

A transition \( t \) is L3-live if there is an SCC, that contains two markings, \( M \) and \( M' \), and \( M(t)M' \), 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:

\[
\text{L3-live}(t) \equiv \sum_{i} [\text{SCC}_i(p_1, \ldots, p_n) \cdot E_t \cdot R_t \cdot \text{SCC}_i(q_1, \ldots, q_n)] \neq 0.
\]

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 \( y \) (\( x \leadsto y \)) [8]. Given \( C_T \), \( C_Y = C_T(x,y) \cdot C_T(y,x) \) and \( C_NY = C_T(x,y) \cdot C_T(y,x) \) can be computed, where \( C_Y(x,y) = 1 \) if \( x \leadsto y \) and \( y \leadsto x \) and \( C_NY(x,y) = 1 \) if \( x \leadsto y \), but there is no firing sequence leading from \( y \) to \( x \) (\( y \not\leadsto x \)). 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
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 SCC and TSSC components in a system 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 \( t_i \), the set of markings in which \( t_i \) is enabled is computed. Next, the set of markings reachable in one step by firing any transition different from \( t_i \) is obtained. If \( t_i \) 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 \( CR \) between pairs of transitions \( T \times T \) 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.,

\[
\forall t, t' \in T : (t_1, t_2) \in CR \iff \exists M \in [M_0] : M[t_1 t_2] \land M[t_2 t_1].
\]

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 \( Enabled = [M_0] \cdot E_{t_1} \cdot E_{t_2} \). 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.,

\[
\text{Reach}_{\text{t1}} := \text{Img}(t_1, Enabled) \cdot E_{t_2}
\]

and

\[
\text{Reach}_{\text{t2}} := \text{Img}(t_2, Enabled) \cdot E_{t_1}.
\]

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 \([M_0]\), 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
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 $M_0$ 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

<table>
<thead>
<tr>
<th>name</th>
<th>P</th>
<th>T</th>
<th>RS</th>
<th>V</th>
<th>maxDD</th>
<th>rsDD</th>
<th>ite</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>muller-30</td>
<td>120</td>
<td>60</td>
<td>$6.0 \times 10^7$</td>
<td>120</td>
<td>15312</td>
<td>2814</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>muller-60</td>
<td>240</td>
<td>120</td>
<td>$8.3 \times 10^{15}$</td>
<td>240</td>
<td>109023</td>
<td>10251</td>
<td>9</td>
<td>604</td>
</tr>
<tr>
<td>muller-80</td>
<td>320</td>
<td>160</td>
<td>$5.8 \times 10^{21}$</td>
<td>320</td>
<td>273963</td>
<td>18221</td>
<td>11</td>
<td>2818</td>
</tr>
<tr>
<td>phil-5</td>
<td>65</td>
<td>50</td>
<td>$8.5 \times 10^{4}$</td>
<td>65</td>
<td>2518</td>
<td>639</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>phil-10</td>
<td>130</td>
<td>100</td>
<td>$7.4 \times 10^{9}$</td>
<td>130</td>
<td>27097</td>
<td>7805</td>
<td>3</td>
<td>37</td>
</tr>
<tr>
<td>phil-15</td>
<td>195</td>
<td>150</td>
<td>$6.4 \times 10^{14}$</td>
<td>195</td>
<td>306749</td>
<td>87419</td>
<td>3</td>
<td>700</td>
</tr>
<tr>
<td>slot-5</td>
<td>50</td>
<td>50</td>
<td>$1.7 \times 10^{6}$</td>
<td>50</td>
<td>3072</td>
<td>540</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>slot-7</td>
<td>70</td>
<td>70</td>
<td>$8.0 \times 10^{8}$</td>
<td>70</td>
<td>7018</td>
<td>1014</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>slot-9</td>
<td>90</td>
<td>90</td>
<td>$3.8 \times 10^{11}$</td>
<td>90</td>
<td>9061</td>
<td>1632</td>
<td>5</td>
<td>43</td>
</tr>
</tbody>
</table>
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 $2 \times 10^4$ 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 ($\text{maxDD}$), the final number of BDD nodes to represent the reachability set ($\text{rsDD}$), the number of traverse iterations ($\text{ite}$), 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.
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

<table>
<thead>
<tr>
<th>PN name</th>
<th>P</th>
<th>T</th>
<th>RS</th>
<th>V</th>
<th>maxDD</th>
<th>rsDD</th>
<th>ite</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab.gesc</td>
<td>52</td>
<td>52</td>
<td>4.9 \times 10^3</td>
<td>52</td>
<td>3175</td>
<td>1198</td>
<td>52</td>
<td>8</td>
</tr>
<tr>
<td>elevator</td>
<td>47</td>
<td>51</td>
<td>1.9 \times 10^3</td>
<td>47</td>
<td>2756</td>
<td>1209</td>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>mutual</td>
<td>49</td>
<td>41</td>
<td>3.2 \times 10^3</td>
<td>49</td>
<td>3572</td>
<td>1500</td>
<td>53</td>
<td>6</td>
</tr>
<tr>
<td>parrow</td>
<td>66</td>
<td>48</td>
<td>8.0 \times 10^4</td>
<td>66</td>
<td>17483</td>
<td>6342</td>
<td>52</td>
<td>49</td>
</tr>
<tr>
<td>sdLarq</td>
<td>154</td>
<td>92</td>
<td>3.9 \times 10^3</td>
<td>154</td>
<td>15955</td>
<td>9532</td>
<td>121</td>
<td>95</td>
</tr>
<tr>
<td>rw1w4r</td>
<td>125</td>
<td>396</td>
<td>1.6 \times 10^6</td>
<td>125</td>
<td>83482</td>
<td>17466</td>
<td>79</td>
<td>2174</td>
</tr>
<tr>
<td>rw2w2r</td>
<td>220</td>
<td>1570</td>
<td>1.9 \times 10^6</td>
<td>220</td>
<td>1484801</td>
<td>92543</td>
<td>96</td>
<td>41336</td>
</tr>
<tr>
<td>par4</td>
<td>31</td>
<td>28</td>
<td>1.3 \times 10^3</td>
<td>31</td>
<td>1538</td>
<td>371</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>par8</td>
<td>66</td>
<td>52</td>
<td>1.6 \times 10^6</td>
<td>66</td>
<td>72332</td>
<td>13276</td>
<td>52</td>
<td>337</td>
</tr>
<tr>
<td>par16</td>
<td>130</td>
<td>100</td>
<td>2.8 \times 10^{12}</td>
<td>130</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DME8</td>
<td>137</td>
<td>128</td>
<td>3.1 \times 10^5</td>
<td>137</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JjReg03</td>
<td>248</td>
<td>249</td>
<td>1.1 \times 10^5</td>
<td>248</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dme3</td>
<td>295</td>
<td>492</td>
<td>6.5 \times 10^3</td>
<td>295</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 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 than 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


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.

IEEE Computer Society publications cited in this article can be found in our Digital Library at http://computer.org/publications/dlib.