Efficient Encoding Schemes for Symbolic Analysis of Petri Nets

Enric Pastor
Department of Computer Architecture
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
enric@ac.upc.es

Jordi Cortadella
Department of Software
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
jordic@lsi.upc.es

Abstract

Petri nets are a graph-based formalism appropriate to model concurrent systems such as asynchronous circuits or network protocols. Symbolic techniques based on Binary Decision Diagrams (BDDs) have emerged as one of the strategies to overcome the state explosion problem in the analysis of systems modeled by Petri nets. The existing techniques for state encoding use a variable-per-place strategy that leads to encoding schemes with very low density. This drawback has been partially mitigated by using Zero-Suppressed BDDs, that provide a typical reduction of BDD sizes by a factor of two.

This work presents novel encoding schemes for Petri nets. By using algebraic techniques to analyze the topology of the net, sets of places "structurally related" can be derived and encoded by only using a logarithmic number of boolean variables. Such approach allows to drastically decrease the number of variables for state encoding and reduce memory and CPU requirements significantly.

1 Introduction

Petri nets (PN) are a graph-based mathematical formalism adequate to describe, model and analyze the behavior of concurrent systems. PNs allow the description of sequential and non-sequential behaviors (including concurrence and non-deterministic choice). Since its introduction by C.A.Petri in 1962, 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. In particular, they play an increasingly important role in the synthesis and verification of digital asynchronous circuits [17, 10, 4]. and have been recently proposed to specify and synthesize systems in hardware/software codesign frameworks [9, 5].

Recently, it has been proposed an efficient enumerative technique for the analysis of bounded Petri Nets [8, 16]. The proposed technique defines an isomorphism between PNs and Boolean algebras. Each marking in the PN is described by means of boolean variables, and the specification of its behavior by means of *boolean functions*. The potential state explosion derived from the enumeration of markings is managed by using *Binary Decision Diagrams* (BDD) [2, 1]. Experiments show that large sets of encoded markings can be represented with small BDDs, and therefore PNs can be efficiently analyzed manipulating those sets.

The existing techniques for the symbolic analysis of safe PNs use naive schemes to encode the markings [8, 16]. Each place is represented by means of a boolean variable that is asserted in case the place is marked. This scheme results in a very sparse state space. Zero-suppressed BDDs have been proposed to efficiently handle this sparsity [18]. It is also well-known that the number of encoding variables is one of the crucial factors that influence on the efficiency of BDD techniques. Encoding schemes that reduce the number of variables provide more compact representations, and therefore allow the analysis of more complex systems.

In this work we propose a dense encoding scheme that results in an increased efficiency for the symbolic analysis of Petri Nets. It is based on a drastic reduction of the number of variables that produces very dense encodings. Moreover, such density is achieved without introducing significant encoding overhead and complexity in the analysis of the dynamic behavior of the net.

The paper is organized as follows. In Section 2 we review some basic properties of PNs and sketch techniques for symbolic analysis. Section 3 overviews the new method by means of an example. The new encoding scheme is presented in Section 4. Section 5 describes the basic boolean equations required for BDD-based symbolic analysis. Experimental results are presented in Section 6. Section 7 concludes the paper including a discussion of the current and future scope of this work.

2 Petri nets

An ordinary Petri Net (PN) is a 4-tuple $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_0 \rangle$, where $\mathcal{P}$ and $\mathcal{T}$ are disjoint and finite sets of *places* and *transitions*, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the *flow relation*, and $M_0 : \mathcal{P} \rightarrow \mathbb{N}$ is the initial marking. The set of places and transitions is called the set of *nodes* of the net. The *pre-* and *post-sets* of nodes are specified by a dot notation, where ${}^\bullet u = \{v \in \mathcal{P} \cup \mathcal{T} | (v, u) \in \mathcal{F}\}$ is called the pre-set of u, and $u^\bullet = \{v \in \mathcal{P} \cup \mathcal{T} | (u, v) \in \mathcal{F}\}$ is called the post-set of u. The pre-set of a place (transition) is the set of input transitions (places). The post-set of a place (transition) is the set of output transitions (places). A *marking* of a PN is an assignment of a non-negative integer to each place. If k is assigned to place p by marking M, we will say that p is marked with k tokens, i.e. $M(p) = k$.

PNs are graphically represented by drawing places as circles, transitions as boxes (or sometimes bars), the flow relation as directed arcs, and tokens as dots. Figure 1 depicts a PN with the set of places $\mathcal{P} = \{p_1, \ldots, p_7\}$, the set of transitions $\mathcal{T} = \{t_1, \ldots, t_7\}$, and the flow relation $\mathcal{F} = \{(p_1, t_1), (p_1, t_2), (t_1, p_2), \ldots\}$. In the initial marking M_0, place p_1 is marked.

A transition t is *enabled* in a marking M, denoted by $M[t\rangle$, when all places in ${}^\bullet t$ are marked. An enabled transition in M is allowed to *fire*. When it fires, it removes a token from each place in ${}^\bullet t$ and adds a token to each place in $t^\bullet$, reaching a new marking M' ($M[t\rangle M'$). A marking M is *reachable* from M_0 if there is a sequence of *firings* $t_1 t_2 \ldots t_n$ that transforms M_0 into M ($M_0[t_1 t_2 \ldots t_n\rangle M$), hence $t_1 t_2 \ldots t_n$ is a *feasible* sequence. The set of reachable markings from M_0 is denoted by $[M_0\rangle$.

The finite automata that contains the set of reachable markings and all the possible firing sequences of a PN is called the *reachability graph*. Figure 1.b depicts the reachability graph for the PN previously presented in Figure 1.a. There are a total of eight reachable markings in $[M_0\rangle$, each one represented by the subset of marked places.

A PN is *safe* if no marking in $[M_0\rangle$ can assign more than one token to any place. This paper only covers the analysis of safe PNs, although the extension to unsafe PNs is straightforward [16].

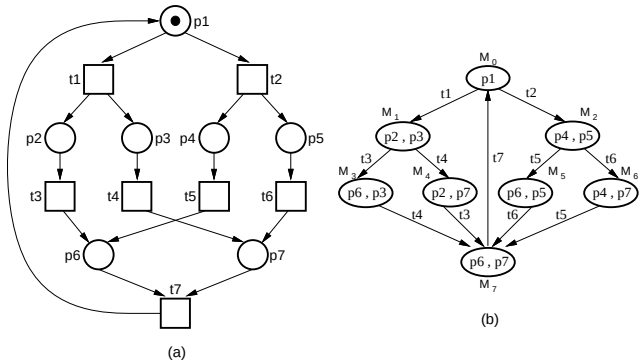

Figure 1: (a) A Petri net with its initial marking, (b) its corresponding reachability graph.

2.1 Algebraic analysis of Petri nets

Let us represent the sets $\mathcal{P}$ and $\mathcal{T}$ as vectors, $[p_1 \dots p_{|\mathcal{P}|}]$ and $[t_1 \dots t_{|\mathcal{T}|}]$, with some arbitrary order of places and transitions. Given a subset of places $R \subseteq \mathcal{P}$, $\chi[R]$ denotes the characteristic vector of R w.r.t. $\mathcal{P}$, defined as,

$$\chi[R](p) = \begin{cases} 1 & \text{if } p \in R \\ 0 & \text{if } p \notin R \end{cases}$$

The structure of a PN net can be represented by an *incidence matrix*. The incidence matrix $C : \mathcal{P} \times \mathcal{T} \rightarrow \{-1, 0, 1\}$ is defined as: $\forall t \in \mathcal{T} : C(-, t) = \chi[t^\bullet] - \chi[{}^\bullet t]$ A marking M is represented as a $|\mathcal{P}| \times 1$ column vector $[M(p_1) \dots M(p_{|\mathcal{P}|})]$, where the ith element of M denotes the token count of p_i. The incidence matrix of PN depicted in Figure 1 is the following:

$$C = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -1 \end{pmatrix}$$

with $M_0 = [1\,0\,0\,0\,0\,0\,0]$.

The interpretation of the incidence matrix is as follows. The non-zero elements of each row indicate the effect of the corresponding transitions on the token count of the corresponding place: input transitions put a token (1), whereas output transitions remove a token (-1).

Given a marking M and a firing sequence of transitions σ such that $M[\sigma\rangle M'$ then

$$M' = M + C \cdot \vec{\sigma} \tag{1}$$

where $\vec{\sigma}$ is the *firing count vector*, i.e. $\vec{\sigma}[i]$ is the number of occurrences of transition t_i in σ. Equation (1) is known as the *state equation* of the PN.

2.2 Place-invariants and State Machines Components

Every solution $X \in \mathbb{Q}^{|\mathcal{P}|}$ of the equation $X^T \cdot C = 0$ is said to be a P-invariant [15]. A P-invariant I is called semi-positive if $I \geq 0$ and $I \neq 0$. The support of a semi-positive invariant I, denoted by $\langle I \rangle$, is the set of places p satisfying $I(p) > 0$. A semi-positive invariant I is minimal if no other semi-positive invariant J satisfies $\langle J \rangle \subset \langle I \rangle$.

In the example of Figure 1, the vector $I = [2\,1\,1\,1\,1\,1\,1]$ is a semi-positive P-invariant. However it is not minimal. The vectors $I_1 = [1\,1\,0\,1\,0\,1\,0]$ and $I_2 = [1\,0\,1\,0\,1\,0\,1]$ are minimal semi-positive P-invariants[1].

[1]Note that $I = I_1 + I_2$.

A *State Machine* (SM) is a PN such that each transition has exactly one input place and one output place. Given a PN $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_0 \rangle$ and a subset of places $\mathcal{P}' \subseteq \mathcal{P}$ a new PN $N' = \langle \mathcal{P}', \mathcal{T}', \mathcal{F}', M_0' \rangle$ can be generated as follows:

- $\mathcal{T}' = \{t \in {}^\bullet p \cup p^\bullet | p \in \mathcal{P}'\}$
- $\mathcal{F}' = \mathcal{F} \cap ((\mathcal{P}' \times \mathcal{T}') \cup (\mathcal{T}' \times \mathcal{P}'))$
- $M_0'(p) = M_0(p)$ for every $p \in \mathcal{P}'$.

The PN N' generated by a subset of places is said to be a State Machine Component (SMC) of N if N' is a strongly connected State Machine. Figure 2.e shows two SMCs generated from the PN of Figure 1. The SMCs have been generated by the sets of places $\{p_1, p_2, p_4, p_6\}$ (invariant $I_1 = [1\,1\,0\,1\,0\,1\,0]$) and $\{p_1, p_3, p_5, p_7\}$ (invariant $I_2 = [1\,0\,1\,0\,1\,0\,1]$) respectively.

The places of an SMC are said to be covered by the SMC. A PN is said to be decomposable into SMCs if there is a set of SMCs that cover all places of the PN. It is known that some classes of PNs are decomposable into SMCs [7]. In general, only a subset of places can be covered by SMCs.

A key theorem for the contribution of this work is the following.

Theorem 2.1 ([6])
Let $N' = \langle \mathcal{P}', \mathcal{T}', \mathcal{F}', M_0' \rangle$ be a State Machine Component of a Petri Net N. Then $\chi[\mathcal{P}']$ is a minimal semi-positive P-invariant of N.

Informally this means that the token count of an SMC is preserved for all reachable markings. As a consequence, if an SMC contains only one token, then one and only one place of the SMC will be marked in all reachable markings of the PN. This suggests that efficient encodings can be found for SMCs.

Calculating SMCs that cover selected places can be efficiently done by using linear programming techniques [14, 11]. An SMC covering a place p can be obtained by computing a minimal semi-positive P-invariant I_{SM} that includes p, with the additional restriction of only containing one token in the initial marking M_0. This minimal P-invariant is computed by solving the linear system of equations [11]:

$$\begin{cases} min\ \ I_{\text{SM}} & \geq & 0 \\ I_{\text{SM}} \cdot C & = & 0 \\ I_{\text{SM}}(p) & \geq & 1 \\ \sum I(p) \cdot M_0(p) & = & 1 \end{cases}$$

2.3 PN Symbolic Analysis

A marking in an ordinary and safe PN can be represented by a set of places $M = \{p_1, \dots, p_k\} \subseteq \mathcal{P}$, where $p_i \in M$ denotes the fact that there is a token in p_i. Let M_P be the union of all potential sets of places representing markings of a PN with $|\mathcal{P}|$ places ($|M_P| = 2^{|\mathcal{P}|}$).

The methods proposed so far to represent markings of a safe PN [16, 18] have used the fact that each marking can be represented by the characteristic function of a subset of places. Thus, by using a boolean variable for each place, any of the $2^{|\mathcal{P}|}$ safe markings corresponds to a minterm of $B^{|\mathcal{P}|}$. With an abuse of notation, let us call p_i the boolean variable representing the marking of place p_i.

The *transition function* $\delta = (\delta_1, \dots, \delta_{|\mathcal{P}|})$ for a transition $t \in \mathcal{T}$ defines how the contents of each place is transformed as a result of firing t ($M' = \delta(M, t)$). $\delta_i(p_1, \dots, p_{|\mathcal{P}|}, t)$ is a function only defined for those markings in which t is enabled:

$$\delta_i(p_1, \dots, p_{|\mathcal{P}|}, t) = \begin{cases} 1 & \text{if } p_i \in t^\bullet \\ 0 & \text{if } p_i \in {}^\bullet t \text{ and } p_i \notin t^\bullet \\ p_i & \text{otherwise} \end{cases} \tag{2}$$

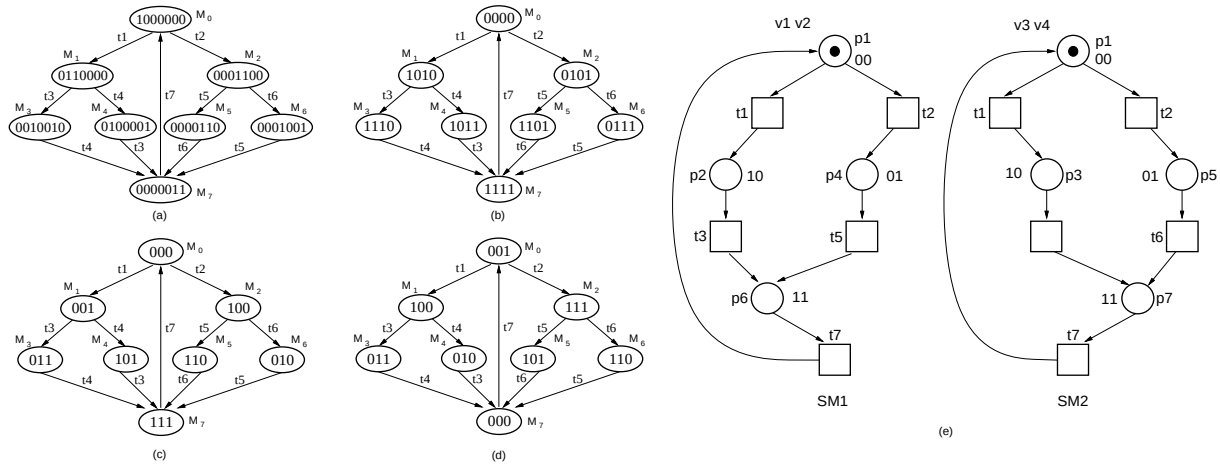

Figure 2: Encoding schemes: (a) one variable per place, (b) SMC-based, (c,d) with minimum number of variables. (e) two SMCs covering the PN.

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

$$E_t = \bigwedge_{p_i \in {}^\bullet t} p_i$$

The function δ also induces a binary relation between markings. Thus $M \mathcal{R}_t M'$ iff $M' = \delta(M, t)$. By using two different sets of variables, $\mathsf{P} = \{p_1, \ldots, p_{|\mathcal{P}|}\}$ and $\mathsf{Q} = \{q_1, \ldots, q_{|\mathcal{P}|}\}$, to represent the current and the next marking before firing a transition t, the characteristic function of $\mathcal{R}_t$ can be represented by a *transition relation*[2]:

$$\mathcal{R}_t(\mathsf{P}, \mathsf{Q}) = \bigwedge_{i=1}^{|\mathcal{P}|} \Big(q_i \equiv \delta_i(\mathsf{P}, t) \Big)$$

Finding the set of markings M' that can be reached after firing transition t from any marking in the set M (the *image computation* for transitions) is reduced to compute: $M' = \exists_{\mathsf{P}} \left[M \cdot \mathcal{R}_t(\mathsf{P}, \mathsf{Q}) \right]$. The transition relation of the PN for the calculation of all markings reachable after firing one transition is

$$\mathcal{R}(\mathsf{P}, \mathsf{Q}) = \bigvee_{t \in \mathcal{T}} \mathcal{R}_t(\mathsf{P}, \mathsf{Q}) \tag{3}$$

Expression (3) suggests that $\mathcal{R}(\mathsf{P}, \mathsf{Q})$ can be efficiently manipulated by representing it as partitioned transition relation in disjunctive form [3].

3 Overview

The proposed encoding scheme is based on using the information that can be known a priori from the PN structure. This information allows to discard sets of unreachable markings and find more efficient encodings for those that are still potentially reachable.

Multiple encoding schemes can be applied to model the same system depending on the objectives of the application. The ratio between the number of states in the system and the variables used to represent each state defines the density of the encoding.

We now describe and compare different encoding schemes for the example of Figure 1.a.

one variable per place: each marking is represented by the characteristic vector of the marked places (see Figure 2.a). The number of variables is $|\mathcal{P}|$.

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

optimal number of variables: the markings are arbitrarily encoded with $\lceil \log_2 |[M_0\rangle| \rceil$ variables (see Figures 2.c and 2.d).

SMC-based encoding: each SMC with k places is encoded with $\lceil \log_2 k \rceil$ variables. Each code corresponds to one place of the SMC, the one containing the token. Figure 2.b shows an encoding based on the SMCs SM1 and SM2 shown in Figure 2.e. The encoding corresponds to $p_1 = 00$, $p_2 = 10$, $p_4 = 01$ and $p_6 = 11$ for SM1 and $p_1 = 00$, $p_3 = 10$, $p_5 = 01$ and $p_7 = 11$ for SM2.

Deriving optimal encoding schemes with minimum number of variables is not a viable strategy because it requires knowing the existing markings a priori, that it is in fact the problem that was originally posed. Hence, the goal of this work is to propose alternative encoding schemes for PNs, that lay in between the conventional one-variable-per-place and the optimal schemes. The proposed methodology should reduce the number of variables, while maintaining a reasonable computation effort.

Besides minimizing the number of variables to represent reachable markings, the proposed scheme will also attempt to reduce the switching activity of the transitions, in other words, reducing the Hamming distance between adjacent markings of the reachability graph. The goal of this strategy is to take advantage of the efficiency of "ad-hoc" BDD procedures that have been specially devised for the manipulation of Petri nets (see Section 5.2).

Figures 2.c and 2.d depict two possible assignments using a binary encoding scheme with three variables. The assignment proposed in (c) requires switching $15/11$ bits on average every time a transition is fired, while the assignment in (d) requires $19/11$ bits. Therefore encoding (c) would be preferable.

Next, the method proposed in this paper for encoding reachable markings of PNs efficiently is sketched:

1. A set of SMCs of the Petri net is calculated. Algebraic and linear programming techniques will be used for such purpose. The primary goal will be to maximize the subset of places covered by SMCs. The calculation of the SMCs is out of the scope of this paper. We refer the reader to [14, 11] for more details.
2. A SMC-based encoding is derived for the places covered by SMCs. The rest of places are encoded by using the conventional one-variable-per-place scheme.
3. Calculate the transition relation of the PN and the reachability graph by using symbolic traversal techniques.

4 SMC-based encoding

The proposed encoding scheme is based on the fact that Petri nets can be totally or partially decomposed into SMCs that contain

an invariant number of tokens. The places in each SMC can be encoded separately using a logarithmic encoding technique. After combining the variables in each component, the result is a reduced number of boolean variables compared to the conventional sparse techniques.

We first describe how an SMC can be encoded by using an optimal number of variables. Given this result, it is necessary to determine the set of SMCs that allows to encode the overall PN while minimizing the total number of required variables. Two different algorithms to select the set of SMCs are proposed, a simple algorithm that does not consider the interactions between SMCs, and a more elaborated algorithm that takes into account those interactions.

4.1 Encoding SMCs

Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the subset of places covered by one SMC S_i containing only one token. Since one and only one of the places of $\mathcal{P}_i$ will be marked at each reachable marking, a logarithmic encoding can be found for the places. Thus, any injective encoding function $\mathcal{E}_{S_i} : \mathcal{P}_i \rightarrow \mathrm{B}^n$, where $n = \lceil \log_2 |\mathcal{P}_i| \rceil$, is appropriate.

4.2 Selecting SMCs

The number of variables required to encode a PN will directly depend on the selected SMCs. Given that the same place may be covered by different SMCs, the *density* of the encoding may decrease because different sets of variables are used to encode the same place at those components. To achieve a dense encoding it is important to select a set of SMCs that minimize the over-encoding of common places.

As an example, consider a PN with one of the SMCs covering 4 places. However 3 out of these 4 places are already covered by other SMCs. If we strictly apply the SMC-based encoding, we would require two additional variables for the new SMC. On the other hand, encoding the only uncovered place with only one variable would result in a smaller total number of variables.

Let $SM = \{S_i\}$ be a set of SMCs that (totally or partially) cover the places of the PN. The problem of finding an optimal subset of SM to encode the PN can be formulated as a *Unate Covering Problem*[13] as follows:

1. Let us take $SM \cup \mathcal{P}$ as the set of covering objects and $\mathcal{P}$ as the set of covered objects. Each S_i covers a subset of places $\mathcal{P}_i \subseteq \mathcal{P}$. Each place $p_i \in \mathcal{P}$ covers itself.
2. For each $S_i \in SM$, define cost$(S_i) = \lceil log_2 |\mathcal{P}_i| \rceil$.
3. For each $p_i \in \mathcal{P}$, define cost$(p_i) = 1$.
4. Find a minimum cost cover of SMCs and places.

4.3 Example

To illustrate the proposed encoding scheme we will use the PN depicted in Figure 4 as example. This PN has 14 places, 22 reachable markings and can be decomposed into six SMCs that cover all places (see Figure 3). The following minimum cost encoding (with density $D = 5/10 = 0.5$) can be found:

- SM_1 covering places $\{p_1, p_2, p_6, p_8\}$ (2 variables).
- SM_3 covering places $\{p_9, p_{10}, p_{12}, p_{14}\}$ (2 variables).
- SM_4 covering places $\{p_9, p_{11}, p_{13}, p_{14}\}$ (2 variables).
- The rest of places encoded with one variable per place (p_3, p_4, p_5 and p_7).

4.4 Improved encoding

The encoding scheme presented in the previous section can be further improved by taking into account that some place may be covered by more than one SMC. In that case, the place can be over-encoded, resulting in a less dense encoding scheme. Intuitively, each place only needs to be encoded once even though it can be covered by several SMCs.

The improved encoding scheme can be implemented as follows. Let us assume that a subset of SMCs, $\{S_1, \ldots, S_{i-1}\}$ is

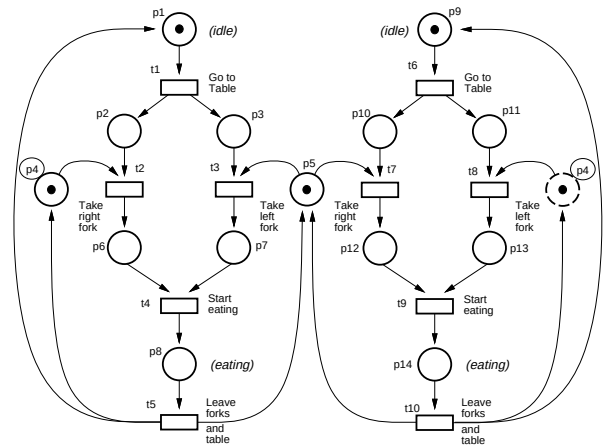

Figure 4: PN for two dining philosophers (two instances of p_4 are depicted for clarity).

already used to encode some places of the PN. Let us include now a new SMC S_i covering the places $\mathcal{P}_i$. We can partition $\mathcal{P}_i$ into two subsets $\mathcal{P}_i = \mathcal{P}_{cov} \cup \mathcal{P}_{new}$. $\mathcal{P}_{cov}$ contains all those place already covered by $\{S_1, \ldots, S_{i-1}\}$, whereas $\mathcal{P}_{new}$ contains the places only covered by S_i.

A valid encoding for S_i would be any function $\mathcal{E}_{S_i} : \mathcal{P}_i \rightarrow \mathrm{B}^n$, where $n = \lceil \log_2 |\mathcal{P}_{new}| \rceil$, such that for $p, p' \in \mathcal{P}_{new}$ and $p \neq p'$: $\mathcal{E}_{S_i}(p) \neq \mathcal{E}_{S_i}(p')$.

Note that for each place $p \in \mathcal{P}_{new}$ there may be a set of places $\mathcal{P}_p$ with the same code as p, i.e.

$$\mathcal{P}_p = \{p' \in \mathcal{P}_{cov} | \mathcal{E}_{S_i}(p) = \mathcal{E}_{S_i}(p')\}$$

This ambiguity is only apparent since the marking of p can be determined by the marking of the other SMCs encoding the places of $\mathcal{P}_p$. The calculation of the characteristic function corresponding to each place will be discussed in the next section.

An example on how to use the improved encoding scheme will be presented in Section 5.4.

5 Symbolic Model Checking

This section describes how the characteristic functions for places and transition functions are derived. These functions are the basic elements to execute BDD-based symbolic traversal algorithms for the analysis of the PN.

5.1 Characteristic functions of places

In general, every place p can be covered by several SMCs. By using the improved encoding approach presented in Section 4.4, only one of the SMCs will be used to encode p, whereas the other SMCs will merely assign p a code already used for other places.

Let us call S_p the SMC used to encode place p and $X_{S_p} = x_1 \ldots x_k$ the set of variables used to encode the places of S_p. The characteristic function of place p (markings with p marked) will be the following:

$$\chi[p] = (X_{S_p} = \mathcal{E}_{S_p}(p)) \cdot \bigwedge_{p' \neq p : \mathcal{E}_{S_p}(p) = \mathcal{E}_{S_p}(p')} \overline{X_{S'_p} = \mathcal{E}_{S'_p}(p')} \qquad (4)$$

Informally, A place p is marked when some of the places with code $\mathcal{E}_{S_p}(p)$ is marked in S_p (first factor of the product) but none of the places with the same code is marked in their corresponding encoding SMCs (second factor of the product).

5.2 Toggling activity

Moving from one marking M_1 to another marking M_2 results in switching some variables from 0 to 1 and some variables from 1 to

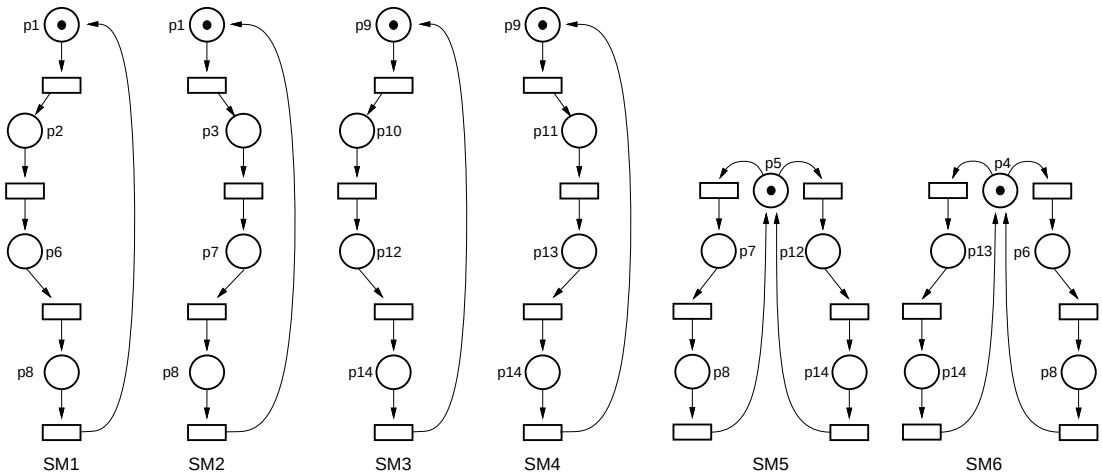

Figure 3: SM decompositions for the dining philosophers example.

SMC / place variables	SM1 x_1x_2	SM3 x_3x_4	SM2 x_5	SM4 x_6	p_4 x_7	p_5 x_8
Encoding	$p_1 = 00$	$p_9 = 00$	$p_1 = 0$	$p_9 = 0$	$p_4 = 1$	$p_5 = 1$
	$p_2 = 01$	$p_{10} = 01$	$p_3 = 0$	$p_{11} = 0$		
	$p_6 = 11$	$p_{12} = 11$	$p_7 = 1$	$p_{13} = 1$		
	$p_8 = 10$	$p_{14} = 10$	$p_8 = 1$	$p_{14} = 1$		

Table 1: PN encoding.

$\chi[p_1] = \overline{x_1} \cdot \overline{x_2}$	$\chi[p_8] = x_1 \cdot \overline{x_2}$
$\chi[p_2] = \overline{x_1} \cdot x_2$	$\chi[p_9] = \overline{x_3} \cdot \overline{x_4}$
$\chi[p_3] = \overline{x_5} \cdot (x_1 + x_2)$	$\chi[p_{10}] = \overline{x_3} \cdot x_4$
$\chi[p_4] = x_7$	$\chi[p_{11}] = \overline{x_6} \cdot (x_3 + x_4)$
$\chi[p_5] = x_8$	$\chi[p_{12}] = x_3 \cdot x_4$
$\chi[p_6] = x_1 \cdot x_2$	$\chi[p_{13}] = x_6 \cdot (\overline{x_3} + x_4)$
$\chi[p_7] = x_5 \cdot (\overline{x_1} + x_2)$	$\chi[p_{14}] = x_3 \cdot \overline{x_4}$

Table 2: Characteristic functions for the places.

0. Implementing the firing of one transition with BDD operations can be reduced to toggling some variables in the BDD. Informally, toggling one variable can be performed by simply interchanging the *then* and *else* arcs of the nodes labeled with the variable.

We will omit the details on how this is performed. We refer the reader to [18] for a similar approach implemented for BDDs and Zero-suppressed BDDs. The important aspect of this strategy is that minimizing the switching activity for each transition results in a speed-up of BDD operations for that transition. Therefore, one of the goals of the encoding scheme is to minimize the number of toggling bits for each transition.

The strategy used in this work is based on using a Gray-like encoding for the places of each SMC component in such a way that the firing of a transition will only produce the toggling of one of the variables used to encode the SMC.

5.3 Transition functions

Given the encoding for each of the places of a Petri net, we only need now to derive the expressions for the transitions functions to be able to perform a symbolic traversal and calculate the reachability graph.

The enabling function E_t for each transition t is simply obtained as follows:

$$E_t = \bigwedge_{p \in {}^\bullet t} \chi[p] \tag{5}$$

Let us now derive expressions for $\delta_i(X, t)$, corresponding to variable x_i of the encoding. Let us call $S = \langle \mathcal{P}', \mathcal{T}', \mathcal{F}', M_0' \rangle$ the SMC using variable x_i for encoding. In case $t \in \mathcal{T}'$ (covered by S), let us call p the output place of t in S, i.e. $\{p\} = t^\bullet \cap \mathcal{P}'$. Thus, the transition function, partially defined over the markings in which t is enabled, is the following:

$$\delta_i(X, t) = \begin{cases} 1 & \text{if } t \in \mathcal{T}' \text{ and } \chi[p] \Rightarrow x_i \\ 0 & \text{if } t \in \mathcal{T}' \text{ and } \chi[p] \Rightarrow \overline{x_i} \\ x_i & \text{if } t \notin \mathcal{T}' \end{cases} \tag{6}$$

Informally, the value of x_i will not change if t is not covered by S, and will take the corresponding encoding value of the output place of t in S otherwise. The transition function for variables corresponding to places not covered by SMCs is identical to the one described by equation (2).

5.4 Example (cont.)

The conventional sparse encoding scheme requires 14 variables for encoding each place of the PN in Figure 4. In Section 4.3 an encoding with 10 variables was proposed. We now propose an improved encoding.

Figure 3 shows all SMCs of the PN. The encoding described by Table 1 can be derived for the places of the PN. The characteristic function for each place is shown in Table 2. In total, 8 variables are required.

Initially, SM1 and SM3 are taken as SMCs with all places not covered by previously selected SMCs. Next, SM2 and SM4 cover some new places but partially overlap with SM1 and SM3. In Table 1, codes in boxes correspond to places encoded by the SMC. Finally, places p_4 and p_5 are encoded with one variable each, since no reduction in variables can be obtained by using new SMCs.

Note that each SMC is encoded using a Gray-like strategy according to the adjacency of the places in the SMC. This strategy allows to reduce the toggling activity of the variables for each transition.

6 Experimental Results

The efficiency of the proposed encoding technique will be measured in terms of the BDD node count reduction to represent the reachability set of the PNs, and the speed-up for that computation. Two experimental scenarios will be analyzed. First, number of variables, BDD sizes and CPU times are compared between the conventional sparse encoding and the proposed dense encoding schemes. Second, the improvements achieved by using the more dense code representation offered by ZDDs (as proposed by Yoneda et al. [18]) are compared against the dense encoding

PN		Sparse encoding			Dense encoding		
name	markings	V	BDD	CPU	V	BDD	CPU
muller-30	6.0×10^7	120	4475	585	60	1315	32
muller-40	4.6×10^{10}	150	4897	7046	80	2339	131
muller-50	3.6×10^{13}	200	-	t.o.	100	3651	449
phil-5	8.5×10^4	65	640	2	35	155	3
phil-8	7.8×10^7	104	2933	12	56	373	19
phil-10	7.4×10^9	130	1689	90	70	425	285
slot-5	1.7×10^6	50	492	14	25	131	5
slot-7	7.9×10^8	70	807	109	35	239	9
slot-9	3.8×10^{11}	90	-	t.o.	45	400	110

Table 3: Comparison between sparse and dense encoding schemes.

PN		ZDD [18]			Dense encoding		
name	markings	V	ZDD	CPU*	V	BDD	CPU
DMEspec8	7.8×10^5	137	32178	14	85	1748	12
DMEspec9	3.5×10^6	154	71602	39	94	2544	20
DMEcir5	8.5×10^5	491	92214	622	249	47952	418
DMEcir7	9.0×10^7	687	504324	10205	347	394334	7584
JJreg-a	1.8×10^6	251	952246	2326	122	17874	836
JJreg-b	1.1×10^5	248	181701	42	120	24355	397

Table 4: Comparison between ZDD compaction and dense encoding schemes ($*$ CPU times for ZDD usage HP-9000 (120MHz, 650MB)).

scheme. CPU times have been obtained by executing the algorithms using the BDD library developed by David Long [12] on a Sun SPARC 20 workstation (128MB).

6.1 Sparse v.s. dense encoding

Table 3 shows the experimental results obtained onto a number of scalable PNs (Muller pipeline, dining philosophers and slotted ring protocol) when using both sparse (one variable per place) and dense encoding schemes. Columns labeled V show the number of boolean variables required for each type of encoding. Columns labeled BDD show the final size of the reachability set. Since it is well known that BDD size strongly depend on variable ordering, no special initial order has been used, while dynamic reordering has been applied at each iteration for both encoding schemes. Columns labeled CPU denote the total computation times for both schemes, including the encoding time itself, which roughly takes 1% of the total computation time in most cases.

The results show a variable reduction around 50% and a BDD node reductions ranging from 2 to 4. CPU times are also reduced at least one order of magnitude for *muller* and *slot*. Even with the BDD node reduction, the computation time increases for *phil*. This is produced by the cost of variable reordering, due to an extremely bad initial order for this set of benchmarks. Deriving good initial orders is still an open line of research.

6.2 ZDDs v.s. dense encoding

Using ZDDs instead of BDDs allows a more compact representation of data without requiring any special encoding strategy. The results in Table 4 show a number of PNs with BDD and ZDD sizes for the reachability set, and computation times as published in [18]. We have executed those benchmarks in our framework obtaining results for both the sparse and dense encoding schemes. The experimental results show important variable reductions (around 40%) that result in significant BDD node reductions compared to ZDDs [18].

7 Conclusions

As PNs become more popular in the specification, synthesis and analysis of concurrent systems there is an increasing need of manipulating them in an efficient way.

This paper has presented an encoding scheme that drastically improves the efficiency of symbolic methods for the analysis of PNs. The structural theory is the key basis for this scheme, which allows to immediately identify sets of markings that will never be reachable. This study is based on the identification of State Machine components with only one token. The efficiency of the encoding scheme lies on the fact that two places in the same State Machine component will never be marked simultaneously.

The structural theory of PNs goes beyond the theory of P-invariants and State Machines components. Although the structure is not enough for a detailed analysis of the PN, it provides crucial information that can be efficiently combined with symbolic enumeration techniques. The authors are now studying a more general framework that combines the efficiency of the structural theory with the accuracy of the symbolic enumeration techniques.

Acknowledgments

We thank T. Yoneda for providing the benchmarks used in their work.

References

[1] K. S. Brace, R. E. Bryant, and R. L. Rudell. Efficient implementation of a BDD package. In *Proc. DAC*, pages 40–45, 1990.

[2] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[3] Jerry R. Burch, Edmund M. Clarke, D. E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Trans. on CAD*, 13(4):401–424, 1994.

[4] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri Nets from State-Based Models. In *Proc. ICCAD*, pages 164–171. IEEE Computer Society Press, November 1995.

[5] J. Cortadella, L. Lavagno, and Ellen Sentovich. Logic synthesis techniques for embedded control code optimization. In *Proc. Int. Workshop on Logic Synthesis*, 1997.

[6] J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, Cambridge, Great Britain, 1995.

[7] M. Hack. *Analysis of production schemata by Petri nets*. M.s. thesis, MIT, February 1972.

[8] Kiyoharu Hamaguchi, Hiromi Hiraishi, and Shuzo Yajima. Design verification of asynchronous sequential circuits using symbolic model checking. In *International Symposium on Logic Synthesis and Microprocessor Architecture*, pages 84–90, July 1992.

[9] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, Alex Taubin, and A. Yakovlev. Coupling asynchrony and interrupts: Place chart nets. In *18th Int. Conf. on Application and Theory of Petri Nets*, volume 1248 of *LNCS*. Springer-Verlag, June 1997.

[10] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig, and A. Yakovlev. Checking signal transition graph implementability by symbolic BDD traversal. In *Proc. EDAC-ETC-EuroASIC*, pages 325–332, Paris, March 1995.

[11] K. Lautenbach. Linear algebraic techniques for place/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986*, volume 254 of *LNCS*, pages 142–167. Springer-Verlag, 1987.

[12] David E. Long. *A binary decision diagram (BDD) package*, June 1993. Manual page.

[13] E. J. McCluskey. Minimization of boolean functions. *Bell Syst. Technical Journal*, (35):1417–1444, November 1956.

[14] G. Memmi and G. Roucairol. Linear algebra in net theory. In W. Brauer, editor, *Net Theory and Applications*, volume 84 of *LNCS*, pages 213–223. Springer-Verlag, 1980.

[15] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.

[16] E. Pastor, O. Roig, J. Cortadella, and R.M. Badia. Petri net analysis using boolean manipulation. In *15th Int. Conf. on Application and Theory of Petri Nets*, volume 815 of *LNCS*, pages 416–435. Springer-Verlag, June 1994.

[17] O. Roig, J. Cortadella, and E. Pastor. Verification of asynchronous circuits by BDD-based model checking of Petri nets. In *16th Int. Conf. on Application and Theory of Petri Nets*, volume 935 of *LNCS*, pages 374–391, Torino, June 1995. Springer-Verlag.

[18] T. Yoneda, H. Hatori, A. Takahara, and S. Minato. BDDs vs. Zero-Suppressed BDDs: for CTL symbolic model checking of petri nets. In *Proc. of Formal Methods in Computer-Aided Design*, volume 1166 of *LNCS*, pages 435–449. Springer-Verlag, 1996.