Structural Methods to Improve the Symbolic Analysis of Petri Nets

Enric Pastor[1], Jordi Cortadella[2], and Marco A. Peña[1]

[1] Department of Computer Architecture
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
enric,marcoa@ac.upc.es
http://www.ac.upc.es/recerca/VLSI/
[2] Department of Software
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
jordic@lsi.upc.es

Abstract. Symbolic techniques based on BDDs (Binary Decision Diagrams) have emerged as an efficient strategy for the analysis of Petri nets. The existing techniques for the symbolic encoding of each marking use a fixed set of variables per place, leading to encoding schemes with very low density. This drawback has been previously mitigated by using Zero-Suppressed BDDs, that provide a typical reduction of BDD sizes by a factor of two.

Structural Petri net theory provides P-invariants that help to derive more efficient encoding schemes for the BDD representations of markings. P-invariants also provide a mechanism to identify conservative upper bounds for the reachable markings. The unreachable markings determined by the upper bound can be used to alleviate both the calculation of the exact reachability set and the scrutiny of properties. Such approach allows to drastically decrease the number of variables for marking encoding and reduce memory and CPU requirements significantly.

1 Introduction

Petri nets (PNs) are a graph-based mathematical formalism that allows to describe systems modeling causality, concurrency and conflict relations among its events [16, 7]. In particular, PNs play an important role in the synthesis and verification of concurrent systems. PNs are applied, for example, to the synthesis and verification of digital asynchronous circuits, to model heterogeneous systems in hardware/software codesign frameworks, and to verify concurrent systems [5, 19].

Symbolic analysis of PNs suffer from the state explosion problem [18, 19]. The number of reachable markings grows exponentially with the size of the PN description. Temporal logic analysis, hazard verification or circuit synthesis, need to express conditions in terms of sets of markings or sequences of events.

Therefore, the size of the representation of the overall state space is critical and limits the efficiency of formal methods on large practical examples. The major goal of the ongoing research on symbolic analysis of PNs is to increase the size of the systems that can be analyzed.

Along the last decades PNs have been deeply investigated with a large number of theoretical results available in the literature. Specially, structural theory connects the dynamic behavior of PNs with its underlying structure [10, 15, 13]. Until recently none of these well-known results has been used in order to alleviate the BDD-based symbolic analysis of PNs.

This work discusses several techniques for the symbolic analysis of PNs. We will show how structural and symbolic techniques can be efficiently combined in the same framework. Sets of P-invariants that will be retrieved from the PN can be applied to ease the symbolic analysis. Previous analysis techniques already make use of binary vector representations of markings [11] and P-invariants to reduce the number of bits in the vector representations [6]. However, they did not exploit the fact that the combination of P-invariants and BDDs already provide information about the reachable markings in the PN.

The proposed algorithms can be classified in two groups according to their application to the computation and representation of the reachability set.

- The first set improves the symbolic BDD representation of the reachability set, reducing the number of required Boolean variables and BDD nodes. Encoding algorithms will be proposed both for the subclass of safe PNs and for general bounded PNs.
- The second set provides enlarged approximations of the reachability set that can be efficiently computed. This approximations can be applied to conservative verification techniques or to provide approximations of the sets of unreachable markings to further reduce the BDD representations.

The outline of the paper is the following. In Section 2 we introduce some basic notions on PNs and Boolean functions. Section 3 describes by means of an example how symbolic BDD techniques currently encode PNs and demonstrates the existence of room for further improvement. An algorithm to encode safe PNs is presented in Section 4. The algorithm is based on sets of one-token SM-Components, assigning to each place in an SM-Component a unique Boolean function. Section 5 extends the encoding methodology to any bounded PN by using general P-invariants. Each potential token configuration in a P-invariant is assigned a unique Boolean function. Additionally, P-invariants allow to determine a set of Potentially Reachable Markings. In Section 6 we show that computing a conservative set of unreachable markings may help to further improve the analysis of PNs. Finally, Section 7 presents experiments that demonstrate the efficiency of the proposed encoding techniques. Section 8 concludes the paper and introduces some future research directions.

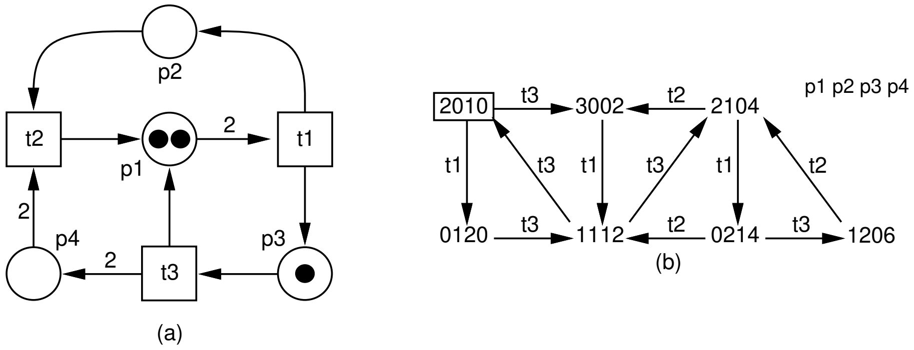

Fig. 1. (a) A bounded PN and (b) its reachability graph.

2 Basic Notations

2.1 Petri nets

A *Petri net* (PN) [16, 7] is a four-tuple $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}, M_0 \rangle$, where $\mathcal{P}$ and $\mathcal{T}$ are sets of places and transitions respectively. $\mathcal{W} : (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}) \rightarrow \mathbb{N}$ defines the *weighted flow relation*. If $\mathcal{W}(u, v) > 0$ then there is an arc from u to v with *weight* $\mathcal{W}(u, v)$. The function $M : \mathcal{P} \rightarrow \mathbb{N}$ is called a *marking*; that is, an assignment of a nonnegative integer to each place. If k is assigned to place p in a marking M, we will say that p is marked with k tokens ($M(p) = k$). M_0 is the initial marking of the PN.

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 circumscribed into the places. Fig. 1(a) depicts a PN with initial marking $M_0 = \{M_0(p_1) = 2, M_0(p_2) = 0, M_0(p_3) = 1, M_0(p_4) = 0\}$.

The set of markings that can be reached from the initial marking M_0 by repeatedly firing the transitions of the net is called the *reachability set* (denoted RS). Fig. 1(b) shows the reachable markings corresponding to the PN example in Fig. 1(a).

A place $p \in \mathcal{P}$ is called *k-bounded* ($k \in \mathbb{N}$) iff at any reachable marking it does not contain more than k tokens. A PN is *bounded* iff every place is k-bounded for some value k. A PN is *safe* if all places are 1-bounded. Fig. 2 depicts a safe PN describing two competing philosophers.

PNs can be symbolically manipulated by means of BDDs [18, 17, 19]. Each place in the PN is considered as an integer variable, being represented by a number of Boolean variables. The RS can be obtained by computing the least fix point of the following recurrence [5]:

$$
\begin{aligned}
S_o &= M_0 \\
S_{i+1} &= S_i \cup Image(PN, S_i)
\end{aligned}
$$

where *Image* is a function that returns the markings reachable from S_i in one step. In the PN example of Fig. 1(a), $\text{Image}(PN, [2010]) = \{[3002], [0120]\}$.

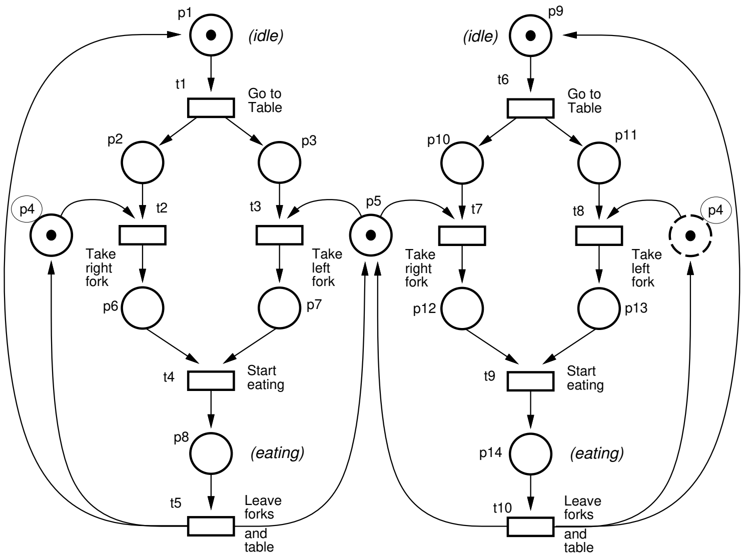

Fig. 2. PN for two dining philosophers (two instances of p_4 are depicted for clarity).

From now on, we will assume that the reader to be familiar with both BDDs and Algebraic Decision Diagrams (ADDs) [2, 4, 1].

2.2 Place-invariants and State Machines Components

The structure of a PN can be represented by its *incidence matrix* [13, 15], a $\mathcal{P} \times \mathcal{T}$ integer matrix given by $C(p_i, t_j) = \mathcal{W}(t_j, p_i) - \mathcal{W}(p_i, t_j)$. The incidence matrix of the PN depicted in Fig. 1(a) is the following:

$$C = \begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{pmatrix}$$

The places of a PN have an associated token conservation equation usually written in the matrix form $M = M_0 + C \cdot \boldsymbol{\sigma}$, where $\boldsymbol{\sigma}$ is called the *Parikh vector* of a sequence of transitions σ.

Every solution $X \in \mathbb{Q}^{|\mathcal{P}|}$ of the equation $X^T \cdot C = 0$ is said to be a P-invariant [16]. A P-invariant I is called *semi-positive* if $I \geq 0$ and $I \neq 0$. The support of a semi-positive P-invariant I, denoted by $\langle I \rangle$, is the set of places p satisfying $I(p) > 0$. A semi-positive P-invariant I is *minimal* if no other semi-positive P-invariant J satisfies $\langle J \rangle \subset \langle I \rangle$. In the sequel, for sake of simplicity we will refer to P-invariants as *invariants*.

Given an invariant I, any reachable marking M must agree with the initial marking M_0; that is, $I \cdot M_0 = I \cdot M$. Therefore, invariants can be used to prove that a marking M is not reachable if M and M_0 do not agree on an invariant.

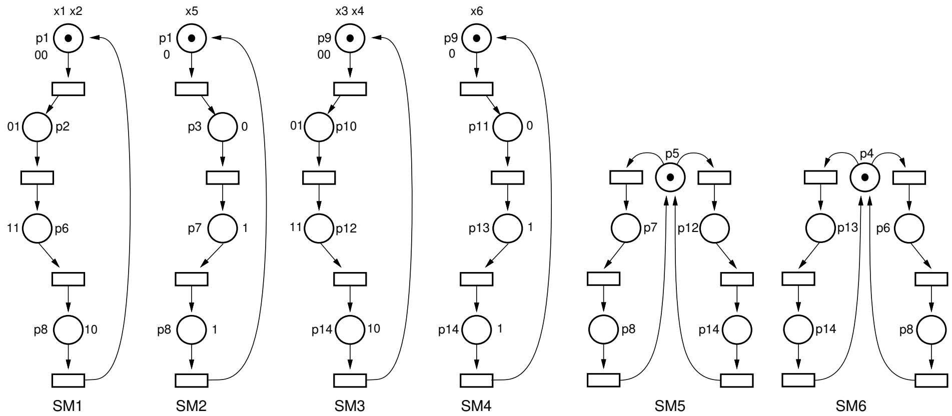

Fig. 3. SM decompositions for the dining philosophers example.

The PN N_i generated by a subset of places is said to be a *State Machine Component* (SM) of N if N_i is a strongly connected State Machine. A key result for the contribution of this work is the following [7]: *Let* $N_i = \langle \mathcal{P}_i, \mathcal{T}_i, \mathcal{W}_i, M_{0i} \rangle$ *be a SM-Component of a Petri Net* N*. Then* N_i *is a minimal semi-positive P-invariant of* N*.*

The *Smith Normal Form* [12] provides an efficient method to derive invariants for bounded PNs. This technique has been introduced by Desel et al. [8] and generates a basis of all possible invariants (not necessarily minimal or semi-positive). A basis of invariants for the PN in Fig. 1(a) is:

$$\begin{array}{l} I_1 : 2p_1 + 4p_2 - p_4 = 4 \\ I_2 : p_1 + p_2 + p_3 = 3 \ . \end{array} \tag{1}$$

For safe PNs we can compute SM-Components by posing a set of linear equations [15, 13]. A minimal one-token semi-positive invariant I_{P}, including a place p_i, can be computed by solving the linear system of equations:

$$\begin{array}{lr} min & \sum_p I_{\mathrm{P}}(p) \geq 0 \\ s.t. & I_{\mathrm{P}} \cdot C = 0 \\ & \sum_p I_{\mathrm{P}}(p) \cdot M_0(p) = 1 \\ & I_{\mathrm{P}}(p_i) \geq 1 \end{array}$$

Figure 3 shows six SM-Components generated from the PN of Fig. 2. For example, SM1 has been generated from the invariant $p_1 + p_2 + p_6 + p_8 = 1$ and SM5 from $p_5 + p_7 + p_8 + p_{12} + p_{14} = 1$.

2.3 Logic Functions

Now we briefly sketch some basic theory on Boolean algebras. Most of the concepts presented here have been extracted from [3].

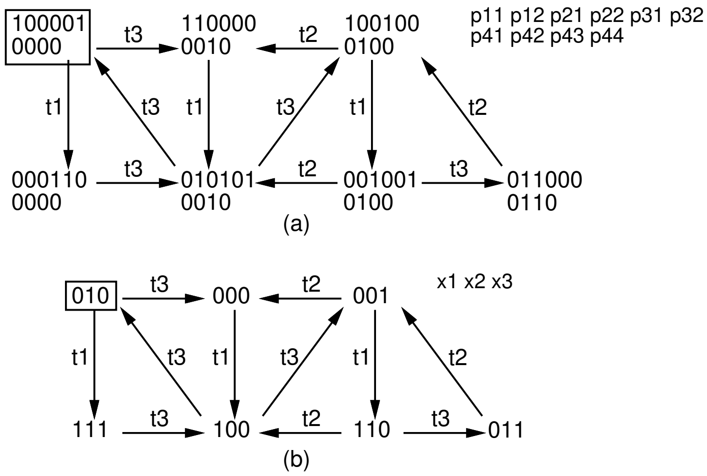

Fig. 4. Encoded reachability graph: (a) sparse and (b) optimal, for the PN in Fig. 1.

A *Boolean algebra* is a quintuple $(\mathrm{B}, +, \cdot, 0, 1)$ where B is a set called the carrier, $+$ and $\cdot$ are binary operations on B, and 0 and 1 are elements of B. The system $(\mathrm{B} = \{0, 1\}, +, \cdot, 0, 1)$, with $+$ and $\cdot$ defined as the *logic OR* and *logic AND* operations respectively, is a Boolean algebra.

An n-variable *logic function* is a mapping $f : \mathrm{B}^n \to \mathrm{B}$. Let $\mathcal{F}_n$ be the set of n-variable logic functions on B. Then the system $(\mathcal{F}_n, +, \cdot, 0, 1)$ is also a Boolean algebra. Let us also define $\mathcal{M}_n$ as the subset of n-variable logic functions such that one and only one combination of inputs evaluates to 1 (i.e. that only contain a minterm).

3 A Motivating Example

The symbolic representation of the *RS* of a PN requires an encoding mechanism to map each marking in a unique binary code inside a Boolean algebra. Traditionally, the encoding has been created by assigning a number of Boolean variables to each place in the PN. The number of variables should be enough to represent the maximum number of tokens that can be located in each place. This encoding technique is known as an *sparse* encoding scheme [17].

Sparse encoding schemes are extremely inefficient for PNs because they assume that all possible combinations of tokens inside places are possible. However, in most cases, places are causally related or in conflict and therefore not all combinations of tokens exist.

In order to compare the efficiency of different encoding schemes we introduce an *encoding density* function $\mathcal{D}$. Given a PN, $\mathcal{D}_{PN}$ is calculated as the optimum number of variables required to encode the RS, divided by the actual number of variables that are used, *i.e.*

$$\mathcal{D}_{PN} = \frac{\lceil \log_2(|[M_0\rangle|) \rceil}{\#\text{ of variables}}$$

An encoding is optimal if $\mathcal{D}_{PN} = 1$. This optimality implies that the Boolean space is fully used and no binary code is left unassigned.

The bounded PN in Fig. 1(a) has four different places that may contain a certain number of tokens. The maximum number of tokens that can be located in each place will determine how many variables are required for sparse encoding. A conservative upper bound for each place p_i can be derived from a basis B of invariants, i.e.

$$\begin{array}{ll} max & M(p_i) \\ s.t. & B^t M = B^t M_0 \end{array}$$

For that example, after solving the Linear Programming Problem we obtain that the maximum number of tokens are $\mathsf{max}(p_1) = \mathsf{max}(p_2) = 3$, $\mathsf{max}(p_3) = 2$ and $\mathsf{max}(p_4) = 8$. That implies that to encode places p_1, p_2, p_3 we need 2 variables for each of them (because their values are between 0 and 3); while place p_4 requires 4 variables (because its values are between 0 and 8). This sparse scheme leads to a Boolean algebra with 10 variables, representing up to 2^{10} different markings. However, it is known that the PN has only 7 reachable markings! (see Fig. 4(a).)

An *optimal* encoding should use a logarithmic number of variables with respect to the number of reachable markings ($\lceil \log_2 |[M_0\rangle| \rceil$). In the previous example, $\lceil log_2\ 7 \rceil = 3$ is the optimal number of variables (see Fig. 4(b)).

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

It is well known that the number of BDD variables does not always have a direct impact on the number of BDD nodes required to represent a set of markings. For a fixed set and number of variables, the number of BDD nodes may vary from polynomial to exponential depending on the variable ordering in the BDD. However, experiments show that a reduction in the number of variables combined with an accurate assignment of binary codes to markings provide significant improvements both in the number of BDD nodes and their computation times.

Finding out relations among places that restrict their simultaneous marking may help to reduce the number of Boolean variables required to encode the same RS [17]. Relations among places not only provide an encoding mechanism, but additionally restrict the set of potentially reachable markings. Some markings will be determined not to be in the RS even before starting any symbolic traversal.

In this work we will propose encoding schemes based on the information known a priori from the PN structure —its invariants. These invariants allow to discard sets of unreachable markings and find more efficient encodings for those that are still potentially reachable. The method proposed for the dense encoding of the reachable markings of a PNs is structured as:

1. A basis of invariants of the PN is calculated. Algebraic and linear programming techniques will be used for ordinary PNs, while the SNF can be used for bounded PNs.
2. The PN must be bounded and the upper bounds must be known, either derived from the invariants or provided by the user.
3. A dense encoding is derived for the places covered by invariants. The rest of places are encoded by using the sparse scheme. Efficient encoding techniques are used for one-token SM-Components, while more elaborated mechanisms are required for general invariants.
4. Assign binary codes to the places in each invariant, in such a way that BDD size is minimized.
5. Given the selected encoding, calculate the transition relation of the PN and the RS by using symbolic traversal techniques.

4 Encoding Safe Petri Nets

This section proposes an encoding scheme that is based on the fact that safe PNs can be decomposed into one-token SM-Components. The places in each SM can be encoded separately using a logarithmic encoding technique. After combining the variables in each invariant, the result is a reduced number of Boolean variables compared to the conventional sparse techniques.

To illustrate the proposed encoding scheme we use the PN in Fig. 2. This PN can be decomposed into six SMs that, in this particular case, cover all places (see Fig. 3). The sparse encoding scheme requires 14 Boolean variables to encode each place, resulting into a density of $\mathcal{D}_{PN} = \lceil \log_2(22) \rceil / 14 = 0.36$.

First, we show how an SM-Component can be encoded by using an optimal number of variables. Then we determine the set of invariants that allows to encode the PN while minimizing the total number of variables. Two methodologies are proposed to select the set of invariants: a simple method that does not consider the interrelations between invariants, and a more elaborated that takes into account those interactions.

4.1 Encoding a single SM

Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the subset of places covered by a one-token SM-Component I_i. Since one and only one place in $\mathcal{P}_i$ is marked at each marking, a logarithmic encoding can be found for those places. Thus, any injective encoding function $\mathcal{E}_{I_i} : \mathcal{P}_i \rightarrow \mathcal{M}_n$ ($n = \lceil \log_2 |\mathcal{P}_i| \rceil$) is appropriate. Each place must be assigned a unique minterm to uniquely identify the location of the token in I_i, *i.e.*

$$\forall p_j, p_k \in \mathcal{P}_i,\ j \neq k\ :\ \mathcal{E}_{I_i}(p_j) \cdot \mathcal{E}_{I_i}(p_k) = 0\ . \tag{2}$$

4.2 Selecting SMs

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

Let SMC $= \{I_i\}$ be a set of SMs that (totally or partially) cover the places of the PN. The problem of finding an optimal subset of SMC to encode the PN can be formulated as a *Unate Covering Problem*[14] as follows:

1. Take SMC $\cup \mathcal{P}$ as the set of covering objects and $\mathcal{P}$ as the set of covered objects. Each invariant I_i covers a subset of places $\mathcal{P}_i \subseteq \mathcal{P}$. Each place $p_i \in \mathcal{P}$ covers itself.
2. For each $I_i \in$ SMC, define $\mathsf{cost}(I_i) = \lceil log_2(|\mathcal{P}_i|) \rceil$.
3. For each $p_i \in \mathcal{P}$, define $\mathsf{cost}(p_i) = 1$.
4. Find a minimum cost cover of SMs and places.

In practice heuristics are used, e.g. a Fiduccia&Mattheyses algorithm that iteratively takes or rejects invariants for encoding [9]. Obviously, the quality of the final encoding depends on the initial selection of invariants and the order in which they are processed.

The final encoding of each place can be computed as the conjunction of the encoding function used in each particular SM; that is,

$$\mathcal{E}(p_j) = \prod_{I_i : p_j \in \mathcal{P}_i} \mathcal{E}_{I_i}(p_j) \ .$$

The following minimum cost encoding using 10 variables (with density $\mathcal{D}_{PN} = 5/10 = 0.5$) 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).

4.3 Combining SMs for a Denser Encoding

The encoding scheme presented in the previous section can be further improved by taking into account that places may be covered by more than one invariant. In that case, a 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 SMs.

A denser encoding scheme can be implemented as follows. Let us assume that a subset of SMs, $\{I_1, \ldots, I_{i-1}\}$ is already used to encode some places of the PN. Let us include now an additional SM I_i covering the places $\mathcal{P}_i$. We can partition $\mathcal{P}_i$ into two subsets $\mathcal{P}_i = \mathcal{P}_i^{cov} \cup \mathcal{P}_i^{new}$. $\mathcal{P}_i^{cov}$ contains all those places already covered by $\{I_1, \ldots, I_{i-1}\}$, whereas $\mathcal{P}_i^{new}$ contains the places only covered by I_i.

Given that places in $\mathcal{P}_i^{cov}$ have been already encoded in other SMs, we only need additional variables to encode the places remaining in $\mathcal{P}_i^{new}$; that is, $\lceil \log_2(|\mathcal{P}_i^{new}|) \rceil$ variables. Since most of the SMs of a PN overlap each other, encoding the places in $\mathcal{P}_i^{new}$ rather than the whole set $\mathcal{P}_i$ should lead to much dense encodings.

Once we have determined the number of variables, we need to define the conditions under which binary codes have to be assigned to encode each place. A valid encoding for I_i would be any function $\mathcal{E}_{I_i} : \mathcal{P}_i \rightarrow \mathcal{M}_n$ ($n = \lceil \log_2(|\mathcal{P}_i^{new}|) \rceil$) such that assigns a unique minterm to each place in $\mathcal{P}_i^{new}$; i.e.

$$\forall p_j, p_k \in \mathcal{P}_i^{new}, p_j \neq p_k \ : \ \mathcal{E}_{I_i}(p_j) \cdot \mathcal{E}_{I_i}(p_k) = 0 \ . \tag{3}$$

Equation (3) imposes looser conditions than (2), because no encoding restriction is defined over places already covered in $\mathcal{P}_i^{cov}$. This encoding scheme will use the new Boolean variables to both encode the places in $\mathcal{P}_i^{new}$ and $\mathcal{P}_i^{cov}$. In that way, a certain degree of over-encoding is introduced for places in $\mathcal{P}_i^{cov}$.

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

$$\mathcal{P}_p = \{p' \in \mathcal{P}_i^{cov} \mid \mathcal{E}_{I_i}(p) \cdot \mathcal{E}_{I_i}(p') \neq \emptyset\} \ .$$

This ambiguity is only apparent since the marking of p can be indirectly determined by the marking of the other SMs encoding the places of $\mathcal{P}_p$. In the extreme case of having a single place in $\mathcal{P}_i^{new}$ no additional variables are required because the value of p can be determined by using the places in $\mathcal{P}_i^{cov}$, i.e. p will be marked iff no other place in $\mathcal{P}_i^{cov}$ is marked.

The number of variables required to encode the PN depicted in Fig. 2 can be reduced by using the improved encoding technique. A minimum cost encoding using 6 variables (with density $\mathcal{D}_{PN} = 5/6 = 0.84$) can be found. To derive this encoding all SMs available in Fig. 3 have been used, but only a subset of places in each SM is covered:

- SM_1 covering places $\{p_1, p_2, p_6, p_8\}$ (2 variables).
- SM_2 covering places $\{p_3, p_7\}$ (1 variable).
- SM_3 covering places $\{p_9, p_{10}, p_{12}, p_{14}\}$ (2 variables).
- SM_4 covering places $\{p_{11}, p_{13}\}$ (1 variable).
- SM_5 covering place $\{p_5\}$ (0 variables).
- SM_6 covering place $\{p_4\}$ (0 variables).

Figure 3 shows all SMs of the PN with the suggested codes to be assigned to each place. The encoding described in Table 1 can be derived for the places of the PN maintaining the one-to-one relation between markings and binary codes.

4.4 Characteristic Functions for Places

In general, every place p can be covered by several SM-Components. By using the improved encoding approach presented in the previous section, only one of

Table 1. Encoding for the dining philosophers example in Fig. 2.

SM / place variables	SM1 $x_1 x_2$	SM3 $x_3 x_4$	SM2 x_5	SM4 x_6	SM5	SM6
Encoding	$\boxed{p_1 = \overline{x_1}\,\overline{x_2}}$	$\boxed{p_9 = \overline{x_3}\,\overline{x_4}}$	$p_1 = \overline{x_5}$	$p_9 = \overline{x_6}$	$\boxed{p_5 = 1}$	$\boxed{p_4 = 1}$
	$\boxed{p_2 = \overline{x_1}\,x_2}$	$\boxed{p_{10} = \overline{x_3}\,x_4}$	$\boxed{p_3 = \overline{x_5}}$	$\boxed{p_{11} = \overline{x_6}}$		
	$\boxed{p_6 = x_1\,x_2}$	$\boxed{p_{12} = x_3\,x_4}$	$\boxed{p_7 = x_5}$	$\boxed{p_{13} = x_6}$		
	$\boxed{p_8 = x_1\,\overline{x_2}}$	$\boxed{p_{14} = x_3\,\overline{x_4}}$	$p_8 = x_5$	$p_{14} = x_6$		

Table 2. Characteristic functions for the places according to Table 1.

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

the SMs will be used to encode p, whereas the other SMs will merely assign p a code already used for other places.

Let us call I_p the SM used to encode place p. The characteristic function of place p ($\chi[p]$ markings with p marked) is the following:

$$\chi[p] = \mathcal{E}_{I_p}(p) \cdot \bigwedge_{p' \neq p:\ \mathcal{E}_{I_p}(p) \cdot \mathcal{E}_{I_p}(p') \neq \emptyset} \overline{\mathcal{E}_{I_{p'}}(p')} \tag{4}$$

The characteristic function for each place in Fig. 2 is shown in Table 2.

5 Bounded PN Encoding

This section will show how invariants can be used to efficiently encode any bounded PN. The goal is to characterize the number of tokens in each place by using the information available in a given invariant. Each invariant describes the distribution of tokens in its places for any reachable marking. However, the analysis of token configurations inside a general invariant is more complex that in a simple one-token SM-Component.

To illustrate the proposed encoding scheme we will use the PN depicted in Fig. 1. This PN can be decomposed into the invariants in (1). A sparse scheme requires 10 Boolean variables to encode all places, resulting in a density of $D_{PN} = \lceil \log_2(7) \rceil / 10 = 0.36$.

First, we will analyze which are the reachable markings characterized by each invariant. A number of variables should be assigned to encode each invariant. However, once an invariant has been encoded, less variables are required to encode the remaining invariants. We introduce a greedy methodology to select which invariants should be encoded first, based on the variable reductions obtained compared to the sparse scheme.

5.1 Token Configurations in Invariants

Let us define a *token configuration* $\mathcal{C}_i$ as an integer assignment to places of an invariant. A token configuration can be *total* or *partial*. A total token configuration defines the token count for all places in the invariant, while a partial token configuration only defines the count for a subset of places. Given the invariant I_1 for the PN in Fig. 1, $\{\{p_1,1\},\{p_2,1\},\{p_4,2\}\}$ is a total token configuration, while $\{\{p_1,1\},\{p_2,0\}\}$ is a partial token configuration.

The exhaustive analysis of each invariant provides all possible token configurations that may correspond to potentially reachable markings. The generation of all potential token configurations can be represented as a tree (see Fig. 5), where each node corresponds to a place and the arc to each child is labeled with possible token assignments. Each leaf in the tree represents a total token configuration. In general, we may generate the token configurations of an invariant that has been partially encoded (e.g. see Section 4.3 for safe nets). The subset of the invariant that has been already encoded will be depicted in a rectangular root node in which each outgoing arc to its child is labeled with the number of tokens already assigned to places (in Fig. 5 no place has been encoded, hence 0 is assigned to the root node and its arc). For both invariants I_1 and I_2 in (1) there exists 10 and 9 total token configurations respectively, as shown in Fig. 5.

In order to characterize the token configurations that may lead to potential markings, we define the *potential marking* function for an invariant I_i as:

$$PM_{I_i} \;:\; 2^{\mathcal{P}_i \times \mathbb{N}} \rightarrow \{0,1\} \;;$$

where $\mathcal{P}_i$ is the set of places in I_i. The PM function characterizes the token configurations $\mathcal{C}_j \in 2^{\mathcal{P}_i \times \mathbb{N}}$ that satisfy $(PM_{I_i}(\mathcal{C}_j) = 1)$ or not $(PM_{I_i}(\mathcal{C}_j) = 0)$ the invariant, e.g. $PM_{I_1}(\{p_1,1\}\{p_2,0\}) = 0$ and $PM_{I_1}(\{p_1,1\}\{p_2,1\}\{p_4,2\}) = 1$ (see Fig. 5). Let $\mathcal{C}_{I_i}$ be the set of potentially reachable total token configurations in I_i.

The combination of information from several invariants further improves the characterization of the potentially reachable markings. Basically, it is known that any reachable marking must agree with all the invariants in the PN. Therefore, if a token configuration does not exists in one invariant then it can not be valid for any other invariant of the PN.

In Fig. 5, $PM_{I_1}(\{p_1,2\},\{p_2,2\}) = 1$ but $PM_{I_2}(\{p_1,2\},\{p_2,2\}) = 0$; therefore the token configuration $\{\{p_1,2\},\{p_2,2\},\{p_4,8\}\}$ is not valid for I_1 and we can update the PM function with $PM_{I_1}(\{p_1,2\},\{p_2,2\},\{p_4,8\}) = 0$. Similarly, the token configurations between the invariants in Fig. 5 indicates that, in fact, no marking with $\{\{p_1,2\},\{p_2,2\}\}$, $\{\{p_1,3\},\{p_2,1\}\}$ or $\{\{p_1,1\},\{p_2,0\}\}$ could exist. The corresponding arcs in the solution trees are eliminated (denoted by shadowed configurations in Fig. 5). We can conclude that each invariant has 8 possible token configurations.

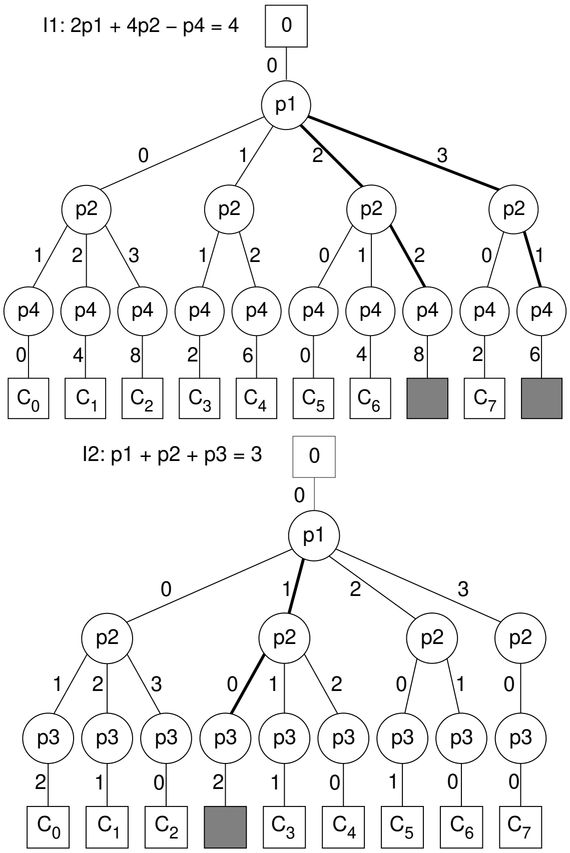

Fig. 5. Token configurations for the invariants of the example in Fig. 1.

Once we have determined the potential token configurations in each invariant we can assign Boolean variables to encode each combination of tokens. The number of variables required to encode the invariant is $V_{I_i} = \lceil \log_2(|\mathcal{C}_{I_i}|) \rceil$. Then, any injective encoding function $\mathcal{E}_{I_i} : \mathcal{C}_{I_i} \rightarrow \mathcal{M}_n$ $(n = V_{I_i})$ is appropriate to encode the invariant. Each different total token configuration must be assigned a unique minterm, *i.e.*

$$\forall \mathcal{C}_j, \mathcal{C}_k \in \mathcal{C}_{I_i},\ j \neq k\ :\ \mathcal{E}_{I_i}(\mathcal{C}_j) \cdot \mathcal{E}_{I_i}(\mathcal{C}_k) = 0\ . \tag{5}$$

For the invariants in (1) we have to encode 8 different token configurations, therefore $\lceil log_2(8) \rceil = 3$ variables are required for each invariant.

Fig. 6 describes a Decision Diagram with one possible encoding of invariant I_1 using three Boolean variables (denoted $x_1 \ldots x_3$). Each one of the 8 total token configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_7)$ is encoded by a different assignment to variables $x_1 \ldots x_3$ (a different minterm described by each branch of the tree). For example,

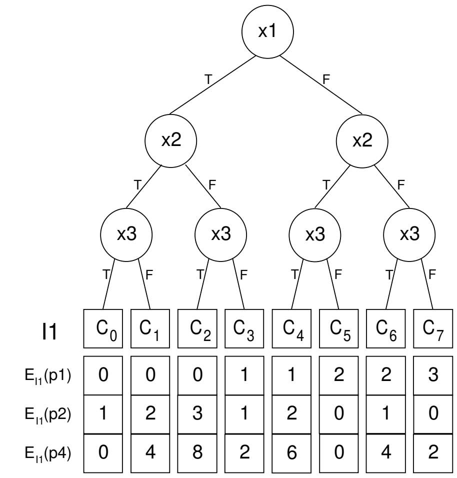

Fig. 6. DDs for the encoding of token configurations in I_1.

the token configuration $\mathcal{C}_4 = \{\{p_1, 1\}, \{p_2, 2\}, \{p_4, 6\}\}$ is encoded as $\mathcal{E}_{I_1}(\mathcal{C}_4) = \overline{x_1}x_2x_3$.

5.2 Invariant Selection for Dense Encoding

Similarly to the techniques used for safe PNs, places that appear in different invariants do not need to be encoded multiple times. Each place must be encoded only at one invariant. The invariant selection process can be formulated as a *Covering Problem* in which each place can be covered by an invariant or left uncovered (using sparse encoding). The goal is to select a number of invariants that minimizes the total number of variables in the encoding.

To avoid the inherent complexity of covering problems, a heuristic algorithm has been derived to select in which invariant a place should be encoded. Basically, we choose the invariant that requires less variables compared to the sparse encoding technique and that has less token configurations to have better control of the minterm assignment process. Given the PN in Fig. 1(a), sparse encoding requires 8 variables for invariant I_1 and 6 variables for I_2. Using the proposed dense encoding only 3 variables are required for each invariant. Invariant I_1 will be encoded first because we obtain an improvement of 5 variables with respect to the sparse technique.

When each potential marking has been encoded it is possible to derive the *encoding function* $\mathcal{E}_{I_i} : \mathcal{P} \times \mathbb{N} \rightarrow \mathcal{F}_n$ that characterizes when a place holds a number k of tokens ($n = V_{I_i}$). This function is the union of total token configurations $\mathcal{C}$ that satisfy $\{p, k\} \in \mathcal{C}$, i.e.

$$\mathcal{E}_{I_i}(p_j, k) = \bigvee_{\mathcal{C}_l \in \mathcal{C}_{I_i} \,:\, \{p_j, k\} \in \mathcal{C}_l} \mathcal{E}_{I_i}(\mathcal{C}_l) \ .$$

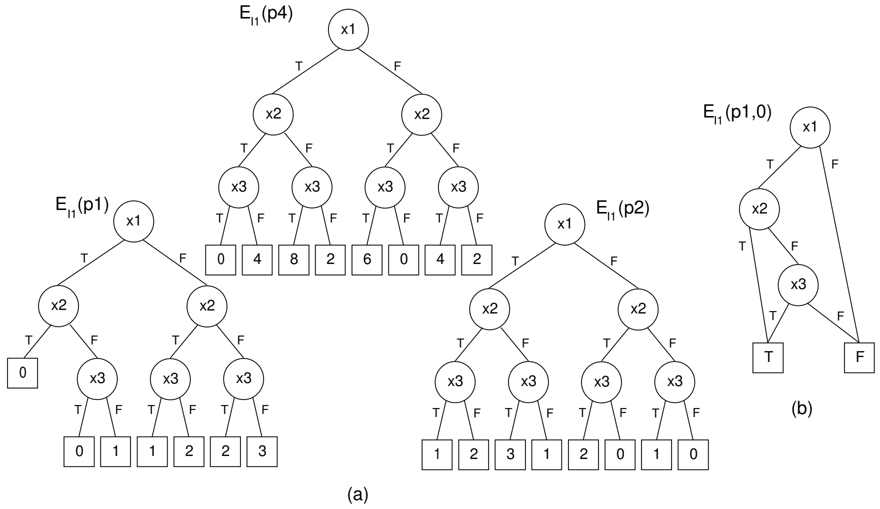

Fig. 7. DDs for the encoding of places in I_1.

A multi-valued encoding function $\mathcal{E}_{I_i} : \mathcal{P} \to \mathbb{N} \times \mathcal{F}_n$ is defined to characterize all token assignments for each place, i.e.

$$\mathcal{E}_{I_i}(p_j) = \bigvee_{\forall k} [k \times \mathcal{E}_{I_i}(p_j, k)] \quad .$$

The token assignments in $\mathcal{E}_{I_i}(p_j)$ can be efficiently represented by means of ADDs. Each branch of an ADD describes a set of binary codes that are assigned to a certain integer value (the token count). Fig. 7(a) explicitly depicts the ADDs for the characteristic function of places in I_1, e.g.

$$\begin{aligned}\mathcal{E}_{I_1}(p_1) = {} & 0 \times (x_1 x_2 + x_1 \overline{x_2} x_3) + 1 \times (x_1 \overline{x_2}\,\overline{x_3} + \overline{x_1} x_2 x_3) + \\ & 2 \times (\overline{x_1} x_2 \overline{x_3} + \overline{x_1}\,\overline{x_2} x_3) + 3 \times (\overline{x_1}\,\overline{x_2}\,\overline{x_3}) \quad . \\ \mathcal{E}_{I_1}(p_2) = {} & 0 \times (\overline{x_1} x_2 \overline{x_3} + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) + 1 \times (x_1 x_2 x_3 + x_1 \overline{x_2}\,\overline{x_3} + \overline{x_1}\,\overline{x_2} x_3) + \\ & 2 \times (x_1 x_2 \overline{x_3} + \overline{x_1} x_2 x_3) + 3 \times (x_1 \overline{x_2} x_3) \quad .\end{aligned}$$

On the other side, BDDs are used to represent the subset of markings in which places have a particular token count. Each branch of a BDD describes a set of binary codes that either belongs to the set (if the leaf node is TRUE) or not (the leaf is FALSE). Fig. 7(a) depicts the BDD for the the characteristic function of $\mathcal{E}_{I_1}(p_1, 0) = x_1 x_2 + x_1 \overline{x_2} x_3$.

Once an invariant has been encoded the rest of invariants may need fewer variables because some places have been already encoded. In the example, places p_1 and p_2 have been already encoded by I_1 and therefore fewer token configurations need to be described when considering I_2. The number of tokens accumulated in $p_1 + p_2$ can be easily computed by operating the ADDs corresponding to $\mathcal{E}_{I_1}(p_1)$ and $\mathcal{E}_{I_1}(p_2)$ [1], i.e.

$$\mathcal{E}_{I_1}(p_1) + \mathcal{E}_{I_1}(p_2) = 1 \times (x_1 x_2 x_3) + 2 \times (x_1 \overline{x_2}\,\overline{x_3} + x_1 x_2 \overline{x_3} + \overline{x_1} x_2 \overline{x_3}) +$$

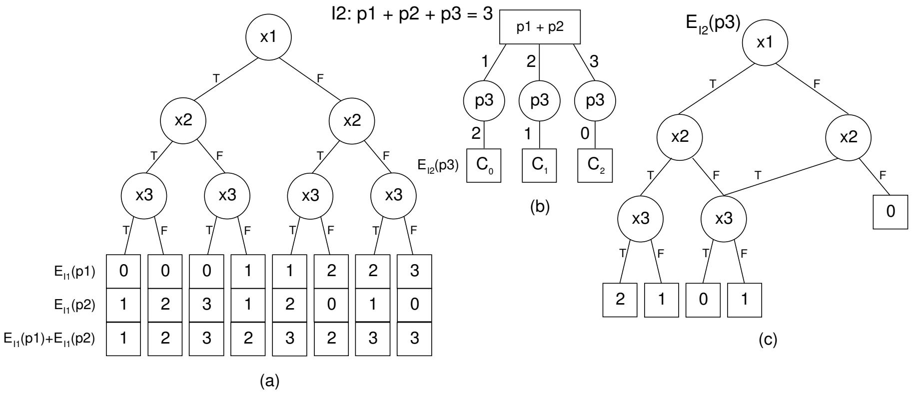

Fig. 8. DDs that characterize the encoding of invariant I_2 after encoding I_1.

$$3 \times (x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) \ .$$

The result shows that only three token configurations exists for the addition of both places, corresponding to $p_1 + p_2 = \{1, 2, 3\}$ (see Fig. 8(a)). Now, it is clear that the value of $\mathcal{E}_{I_2}(p_3)$ can be implicitly derived according to invariant $I_2 : p_1 + p_2 + p_3 = 3$ (see Fig. 8(b)).

The root node (p_1+p_2) of the token configuration tree (see Fig. 8(b)) holds an implicit encoding due to the binary codes previously assigned to other invariants. We denote this encoding function as *implicit encoding function* $\mathcal{E}^i_{I_i} : \mathcal{C}_{I_i} \to \mathcal{F}_m$ because assigns to each token configuration a function that depends on all the m Boolean variables already assigned in previously considered invariants. In Fig. 8(a) we have that $\mathcal{E}^i_{I_i}(\mathcal{C}_0) = x_1x_2x_3$, $\mathcal{E}^i_{I_i}(\mathcal{C}_1) = x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3}$ and $\mathcal{E}^i_{I_i}(\mathcal{C}_2) = x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}$.

Given the root encoding function, the remaining part of the invariants may need fewer variables because the conditions in (5) for the encoding function $\mathcal{E}_{I_i}$ can be relaxed to:

$$\forall \mathcal{C}_j, \mathcal{C}_k \in \mathcal{C}_{I_i},\ j \neq k \ : \ \mathcal{E}^i_{I_i}(\mathcal{C}_j)\mathcal{E}_{I_i}(\mathcal{C}_j) \cdot \mathcal{E}^i_{I_i}(\mathcal{C}_k)\mathcal{E}_{I_i}(\mathcal{C}_k) = 0 \ . \tag{6}$$

Only those token configurations with encoding functions that may intersect should be assigned a unique code (the implicit encoding already prevents some intersections). Hence, the number of variables for encoding is reduced to:

$$V_{I_i} = \left\lceil \log_2 \left| \{ \mathcal{C}_i \ : \ \exists\, \mathcal{C}_j, i \neq j \ s.t.\ \mathcal{E}^i_{I_i}(\mathcal{C}_i) \cdot \mathcal{E}^i_{I_i}(\mathcal{C}_j) \neq 0 \} \right| \right\rceil \ .$$

Finally, if I_p is the invariant used to encode place p, the multi-valued characteristic function $\chi[p]$ of place p must combine the codes assigned in I_p with the implicit information assigned from other invariants, i.e.

$$\chi[p] = \bigvee_{\forall k} \left[k \times \bigvee_{\mathcal{C}_i \in \mathcal{C}_{I_p} \ : \ \{p,k\} \in \mathcal{C}_i} \mathcal{E}^i_{I_p}(\mathcal{C}_i)\mathcal{E}_{I_p}(\mathcal{C}_i) \right] \tag{7}$$

In that case no additional variables are required to encode p_3. The encoding for $\chi(p_3)$ can be created as $\chi(p_3) = 3 - (\mathcal{E}_{I_1}(p_1) + \mathcal{E}_{I_1}(p_2))$ [1], i.e.

$$\begin{aligned}\chi(p_3) = 0 \times (&x_1\overline{x_2}\,x_3 + \overline{x_1}x_2x_3 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}) \\ &1 \times (x_1\overline{x_2}\,\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3}) + 2 \times (x_1x_2x_3) \ .\end{aligned}$$

The overall encoding process can be described as follows:

1. Compute the potentially token configurations for each invariant.
2. Encode the invariant that provides the maximum variable decrease with respect sparse encoding and minimal number of token configurations.
3. Eliminate invariants with all places already encoded.
4. Update the token configuration trees for the remaining invariants.
5. Repeat from 2 until all places have been encoded.

The encoded reachability graph for the PN in Fig. 1(a) is shown in Fig. 4(b).

6 Computation of Potentially Reachable Markings

Invariants not only provide an efficient mechanism to encode places in a PN, but offer an initial approximation of the RS. Any reachable marking must agree with the initial marking at any invariant of a PN. Therefore every invariant can be used to divide the Boolean space into a set of potentially reachable markings and a set of unreachable markings.

The general situation that we consider arises whenever binary codes are left unallocated to any potential token configuration. Given a general invariant I_i : $a_1p_1 + \ldots + a_mp_m = N$, the characteristic function $\chi[I_i]$ of all markings that satisfy that equation is constructed by:

1. Operating the characteristic function $\chi(p_i)$ of all places in the invariant, i.e. $\sum_{p_j \in \mathcal{P}_i} a_i \cdot \chi[p_j]$;
2. In the resulting function, all leaf nodes that are equal to N correspond to markings that satisfy the invariant ($\chi[I_i] = [N - \sum_{p_j \in \mathcal{P}_i} a_i \cdot \chi[p_j] = 0]$).

Since any reachable marking has to satisfy all the invariants, the upper bound of the RS is obtained as the conjunction of the characteristic functions for all invariants.

The characteristic function for one token SM-Components can be easily computed by operating the characteristic function of each place. Given an invariant I_i, when a place $p_j \in \mathcal{P}_i$ is marked ($\chi[p_j] = 1$) the rest of places cannot be marked; hence, the characteristic function is computed as:

$$\chi[I_i] = \sum_{p_j \in \mathcal{P}_i} \chi[p_j] \cdot \overline{[\sum_{p_k \in \mathcal{P}_i, k \neq j} \chi[p_k]]} \ .$$

Approximations of the RS computed by using structural information improves the symbolic analysis of the PN in two ways:

Table 3. Comparison between sparse and dense encoding schemes for safe PNs.

PN				Sparse encoding				Dense encoding						
name	P	T	RS	V	nTR	nRS	CPU	Ninv	Nnodes	V	nTR	nRTR	nRS	CPU
muller10	40	20	4.2×10^2	40	180	770	1	10	40	20	140	123	189	1
muller20	80	40	2.5×10^5	80	360	3188	9	20	80	40	280	241	668	3
muller30	120	60	6.0×10^7	120	540	6694	51	30	120	60	480	426	1390	13
phil5	65	50	8.5×10^4	65	330	639	2	15	125	25	644	459	158	2
phil10	130	100	7.4×10^9	130	660	7805	40	30	250	50	1284	914	433	24
phil15	195	150	6.4×10^{14}	195	990	87419	700	45	375	75	1924	1369	708	124
slot5	50	50	1.7×10^6	50	330	673	14	10	50	25	325	283	129	5
slot10	100	100	3.8×10^{11}	100	660	2516	1006	20	100	50	650	581	460	309

Table 4. Comparison between sparse and dense encoding schemes for bounded PNs.

PN				Sparse encoding				Dense encoding					
name	P	T	RS	V	nTR	nRS	CPU	Ninv	Nnodes	V	nTR	nRS	CPU
proc1	8	8	7	11	107	29	0	4	41	5	131	12	0
robot1	17	8	1.6×10^2	28	208	389	1	11	222	12	99	58	1
robot2	15	6	4.8×10^1	24	149	243	1	10	69	6	817	9	1
robot12	24	14	1.3×10^3	40	358	1330	2	13	9465	18	647	141	7

- A set of markings that is known to be unreachable offers a number of binary codes to be used as *don't care* set. The BDD representation of functions involved in the symbolic analysis can be simplified by using this *don't care* set. In particular, the size of the transition relation and the RS of the PN can be reduced.
- The potential RS approximations may already provide enough information to determine if the properties under analysis are satisfied in a positive or negative way without requiring the symbolic traversal of the PN.

7 Experimental Results

The efficiency of the proposed encoding technique is measured in terms of reductions achieved for number of variables, BDD nodes to represent the transition relation and the RS of the PN, and CPU computation times.

Table 3 compares the results of symbolic traverse after both sparse and dense encoding of several safe PNs based on the general invariant-based algorithm. Scalable examples have been used. *Muller* describes a Muller pipeline with n-stages, *Phil* describes n competing philosophers, *Slot* a model for the slotted-ring protocol with n stages. We have analyzed the results obtained by using a sparse encoding (labeled Sparse) and a dense encoding with set of minimal invariants computed with algebraic techniques (labeled Dense). For both cases we provide the number of Boolean variables required by the encoding (V), the number of BDD nodes to represent the transition relations (nTR) and the RS (nRS), and the computation times (CPU). Additionally, for the dense encoding we provide the number of invariants that have been used ($Ninv$) and the total number of token assignments generated along the encoding process ($Nnodes$). When using the potentially reached markings to simplify the TR of the PN the number of BDD nodes is also presented ($nRTR$).

The experiments show that 50% variable reductions or better can be obtained. The influence of these results is evident on the number of BDD nodes to represent the RS (70% to 90% are obtained) and on the computation times (40% to 80% speedups are achieved). Conversely, the number of BDD nodes to represent the transition relations may even increase due to the complexity of the encoding assignments. The computation of the potentially reachable markings also help to further reduce the size of the TRs between $10 - -30\%$.

Table 4 compares the results of symbolic traverse after both sparse and dense encoding of a few bounded PNs. The examples describe several robot control automatons. Again 50% variable reductions can be obtained. The influence of these results is also quite significant on the number of BDD nodes to represent the RS. However, the increase in the number of nodes to represent the transition relations may reduce the computation speed-ups. Further research is needed in that direction. From the $robot12$ example it can also be seen that in same cases the number of token configurations may be even bigger that the reachable states. Heuristics must be derived to avoid exploring invariants with large number of configurations.

8 Conclusions

This paper has presented encoding techniques that improve the efficiency of symbolic methods for the analysis of PNs. Structural PN theory provides sets of P-invariants to identify interrelations among places, which allows to immediately identify sets of unreachable markings. These techniques alleviate the complexity of the existing symbolic techniques for the calculation of the exact reachability set.

The structural theory of PNs goes beyond P-invariants. Although the structure is not enough for the exact analysis of a PN, it provides information that can be efficiently combined with symbolic analysis. Future work intends to derive a general framework to combine the efficiency of the structural PN theory with the accuracy of the symbolic techniques.

Acknowledgments

This work has been partially funded by CICYT TIC 98-0410 and ACiD-WG (ESPRIT 21949). We thank anonymous reviewers for their useful comments.

References

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proc. ICCAD*, pages 188–191, November 1993.

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

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

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

[5] 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.

[6] G. Chiola Compiling Techniques for the Analysis of Stochastic Petri Nets. In *4th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 11–24, September 1989.

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

[8] J. Desel, K.P. Neuendorf, and M.D. Radola. Proving nonreachability by modulo-invariants. *Theoretical Computer Science*, (153):49–64, 1996.

[9] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. DAC*, 1982.

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

[11] G. J. Holzmann. An Improved Protocol Reachability Analysis Technique *Software Practice and Experience*, 18(2):137–161, 1988.

[12] R. Kannan and A. Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM J. Comput.*, 4(8):499–577, 1979.

[13] 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.

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

[15] 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.

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

[17] E. Pastor and J. Cortadella. Efficient encoding schemes for symbolic analysis of petri nets. In *Proc. Design, Automation and Test in Europe*, pages 790–795, Paris (France), February 1998.

[18] 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.

[19] 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.