Optimized Colored Nets Unfolding

Fabrice Kordon, Alban Linard, and Emmanuel Paviot-Adet

Université P. & M. Curie - Paris 6, CNRS UMR 7606 - LIP6/MoVe
4, place Jussieu, F-75252 Paris CEDEX 05, France
Fabrice.Kordon@lip6.fr, Alban.Linard@lip6.fr, Emmanuel.Paviot-Adet@lip6.fr

Abstract. As some structural properties, like generative families of positive P-invariants, can only be computed in P/T nets, unfolding of Colored Petri Nets is of interest. However, it may generate huge nets that cannot be stored concretely in memory. In some cases, removing the dead parts of the unfolded net can dramatically reduce its size, but this operation requires the unfolded net to be represented anyway. This paper presents a symbolic representation of unfolded nets using Data Decision Diagrams. This technique allows to store very large models and manipulate them for optimization purpose.

1 Introduction

Colored Petri nets, introduced by K. Jensen in 1981 [8] are very convenient for modeling complex systems. However, basic structural properties of P/T nets [11] remain difficult to extend to Colored Petri Nets: a generative family of positive invariants can only be computed under restrictive conditions [5] and structural bounds are generally not available.

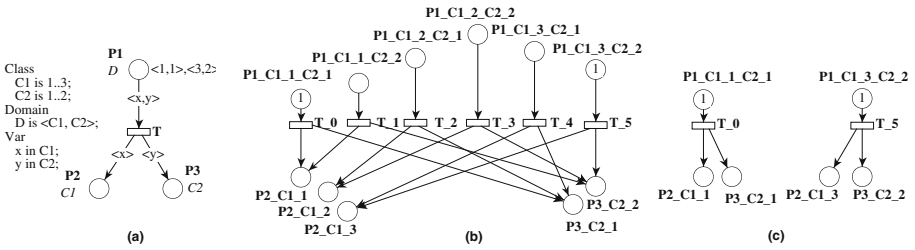


Fig. 1. Example of Net unfolding

To overcome this problem, modelers may transform the colored Petri net into an equivalent P/T net. This operation, called *unfolding*, generates for each original place, numerous P/T places instances according to its color domain. Moreover, one P/T transition is generated for any possible binding of each colored transition. This leads to huge unfolded net. As an example, the colored model (a in Figure 1) is unfolded into a P/T net (b in the figure). Hopefully, in many cases, simplifications can lead to a smaller unfolded net according to:

E. Najm et al. (Eds.): FORTE 2006, LNCS 4229, pp. 340–356, 2006.

1. the initial marking of places (here, place P1 in the colored model lacks some marking and thus, some of the corresponding P/T places are zero-marked),
2. false guards on transitions.

Model c in Figure 1 shows the unfolded Petri net after optimization.

Unfolding Colored Petri nets raises several problems when they become large because the result cannot be stored in memory. This is typically the case for Petri nets derived from higher level specification (such as UML as suggested in [1]). Optimization of the unfolded net, when possible, requires the unfolded net to be represented anyway before optimization.

In this paper, after a brief remainder of Well-Formed Petri Nets [3] in Section 2 and a presentation of Data Decision Diagrams (DDDs) [4] a shared structure we use to store huge P/T nets in Section 3, we present in Section 4 how P/T places and transitions are symbolically stored. P/T arcs, however, are not represented, they are computed on the fly from the colored description when needed. Section 5 describes the main contribution of this work: implementation of the optimization algorithms in this symbolic context. We end with experimentation on various types of specifications to assess when our technique is efficient in Section 6.

2 Well-Formed Colored Petri Nets

In this section we describe Well-Formed Petri Nets (WN) [3], the input formalism for our work. Originally designed to ease structural verification algorithms expression, this formalism is a subset of Colored Petri nets with a simple and rigorous syntax and is also used to exploit model symmetries. Definitions in this section are adapted from [12].

Classes and domains define the color structures used in WN models, allowing tokens to be identified from one another.

Definition 1 (Classes, domains and variables). *A* class *is a cyclicly ordered finite set of elements with successor (resp. predecessor) function defined. A class is an interval over the positive naturals $k_1..k_2$, where $k_1 < k_2$, the successor of k_2 is k_1 (resp. the predecessor of k_1 is k_2).*

A domain *is a Cartesian product of classes.*

Variables *are defined over classes, but not over domains.*

The next definition introduces basic expressions over variables. Those basic expressions are used in predicates and domain functions.

Definition 2 (Basic color functions). *A* basic color function *$E(x)$ is the identity function (noted x) or x++n or x--n the $n^{\underline{th}}$ successor and predecessor of variable x respectively or the constant function (denoted by the correponding class member).*

Definition 3 (Standard Predicates). *A* standard predicate *is a Boolean expression of* basic predicates*. The allowed basic predicates are: $E(x) = E(y)$,*

$E(x) \neq E(y)$, $E(x) < E(y)$ where x and y are variables of same class and $E(x)$ and $E(y)$ are basic color functions.

Definition 4 (Domain functions). *Let $D = \langle C_1, \dots C_n \rangle$ be a domain where C_i are classes. Let x_i be a variable defined over the class C_i.*

A basic domain function *BF of D is : $BF : \langle C_1, \dots C_n \rangle \rightarrow Bag(\langle C_1, \dots C_n \rangle)$ where BF is a function of the form $BF = k.\langle E(x_1), \dots, E(x_n) \rangle$.*

A Domain function *F of D is: $F : \langle C_1, \dots C_n \rangle \rightarrow Bag(\langle C_1, \dots C_n \rangle)$ where F is a function of the form $F = BF_1 + \dots + BF_n$ where BF_i is a basic domain function.*

Definition 5 (Well-Formed Colored Petri Nets). *A Well-Formed Net is a twelve-tuple $N = \langle P, T, Pre, Post, C, D, V, VDom, TVar, Dom, gd, M_0 \rangle$ where:*

- *P and T are disjoint finite non empty sets (respectively the places and transitions of N),*
- *V is a set of variables,*
- *C and D are respectively sets of classes and domains,*
- *$VDom$ is a function that maps variables to classes,*
- *$TVar$ is a function that maps transitions to subsets of V,*
- *Dom is a function defining the color domain of each place and transition, the domain of transition t is a Cartesian product of the classes $VDom(v)$ for each v in $TVar(t)$,*
- *gd is a function defining for each transition t its predicate (called guard), variables used in basic predicates belong to $Tvar(t)$,*
- *$Pre[p,t]$ (resp. $Post[p,t]$) is the pre-incidence (resp. post-incidence) function: a domain function over $Dom(p)$, variables used in each basic color functions belong to the corresponding class in the domain,*
- *$M_0 : M_0(p) \in Bag(Dom(p))$ is the initial marking of place p.*

An equivalent P/T net can be assiociated to each WN. To a colored place P are associated $|Dom(P)|$ ordinay places. Equivalently,to a colored transition t are associated $|VDom(x_1)| * \cdots * |VDom(x_n)|$ ordinary transitions, where x_i belongs to $TVar(t)$. Ordinary arcs link places to transitions according to Pre and $Post$.

3 Data Decision Diagrams

Data Decision Diagrams represent assignment sequence sets of the form $e_1 = x_1; e_2 = x_2; \cdots e_n = x_n$ where e_i are variables and x_i are values. Like in Binary Decision Diagrams, common parts are shared at the beginning and the end of the sequences. No fixed order is needed over the assignments sequences and multiple reassignments are allowed. Moreover, no assumptions are done on the variables domains.

DDDs use three terminal nodes: `0`, `1` and $\top$. As usual, a sequence ending with `1` is part of the set described by the DDD, a sequence ending with `0` is not part of this set and a sequence ending with $\top$ means that an error occurred in a previous operation. In the following, E denotes a set of variables, and for any $e \in E$, Dom(e) represents the domain of e. For more detailed information see [4].

Definition 6 (Data Decision Diagram). *The set* $\mathbb{D}$ *of DDDs is defined by* $d \in \mathbb{D}$ *if:*

- $d \in \{\mathtt{0}, \mathtt{1}, \top\}$ *or*
- $d = (e, \alpha)$ *with:*
  - $e \in E$
  - $\alpha : \mathrm{Dom}(e) \to \mathbb{D}$, *such that* $\{x \in \mathrm{Dom}(e) \mid \alpha(x) \neq 0\}$ *is finite.*

We denote $e \xrightarrow{a} d$, *the DDD* (e, α) *with* $\alpha(a) = d$ *and for all* $x \neq a$, $\alpha(x) = 0$.

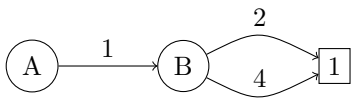
We denote $e \xrightarrow{a..b} d$, *the DDD* (e, α) *with* $\alpha(a) = \ldots = \alpha(b) = d$ *and for all* $x \notin \{a, \ldots b\}$, $\alpha(x) = 0$.

This definition allows multiple DDD representations of the empty set, therefore, each DDD can have multiple representations. An equivalence relation over the DDDs is thus needed. `0` denotes the empty set and each node equivalent to the empty set is replaced by `0`. This induces a canonical representation.

Since DDDs represent sets, sets operators are defined over DDDs: *union* $+$, *intersection* $*$ and *difference* $\setminus$. DDDs also represent sets of sequences, the *concatenation* operator . is also defined: if d_1 and d_2 are two DDDs, then $d_1.d_2$ is composed of all possible sequences beginning with a sequence of d_1 while the remainder is a sequence of d_2.

The main feature of the DDDs that is attractive for our work is the notion of homomorphisms: an homomorphism Φ is a mapping on DDDs that maps the empty set to itself ($\Phi(\mathtt{0}) = \mathtt{0}$) and that is linear with respect to the union ($\Phi(d_1 + d_2) = \Phi(d_1) + \Phi(d_2)$). The identity mapping Id ($Id(d) = d$) is the easiest homomorphism one can define. Basic homomorphisms can be composed to create new homomorphisms: if Φ_1 and Φ_2 are homomorphisms, then $\Phi_1 \circ \Phi_2$ is a new homomorphism.

Another simple homomorphism is the one that takes a couple (variable, value) as parameters $Construct(e, x)$ and returns the DDD composed of a node labeled e and an arc leading to terminal node `1`, labeled x. Using predefined operators, DDDs can be created: $Construct(A, 1).(Construct(B, 2) + Construct(B, 4))$ returns the DDD $A \xrightarrow{1} B \xrightarrow{\{2,4\}} \mathtt{1}$. This DDD is depicted in Figure 2



. **Fig. 2.** Basic DDD creation

Many operations on DDDs, like variable reordering or value modification, cannot be obtained via predefined operators only. A special set of homomorphisms is introduced: the *inductive homomorphisms*. Those homomorphisms associate a DDD to terminal node `1` and apply defined homomorphisms for each sub-DDD to each couple (variable, value).

Definition 7 (Inductive homomorphism). *Let* I *be an index set. Let* $(d_i)_{i \in I}$ *be a family of DDDs. Let* $(\Phi_i(e, x))_{i \in I}$ *be a family of homomorphisms.*

Then the recursive definition of mappings $(\Phi_i)_{i \in I}$ *in Figure 3 defines a family of homomorphisms called inductive homomorphisms.*

$$\forall d \in \mathbb{D}, \Phi_i(d) = \begin{cases} \mathtt{0} & \text{if } d = \mathtt{0} \\ d_i & \text{if } d = \mathtt{1} \\ \top & \text{if } d = \top \\ \sum_{x \in \mathrm{Dom}(e)} \Phi_i(e,x)(\alpha(x)) & \text{if } d = (e, \alpha) \end{cases}$$

Fig. 3. Definition of inductive homomorphisms

4 Shared Structure

In this work, optimization algorithms are designed on P/T nets stored via decision diagrams. Since we chose to cut the unfolded net representation into several short decision diagrams, the symbolic structure must allow variable reordering locally to each set in order to compute P/T arcs. OBDDs [2] cannot, therefore, be used to store the unfolded net.

For this reason, we have chosen Data Decision Diagrams (DDDs) [4]: a structure that is not bound to any order, allows variable repetition and offers a large toolset to handle the structure. After a quick overview of the problem, a detailed description of the symbolic representation is given.

4.1 Partial Unfolding

Unfolding P'_n, if n is a place (or T'_n if n is a transition), of a node n with $Dom(n) = C_0 \times \ldots \times C_m$ is a set of $|C_0| * \ldots * |C_m|$ nodes N'_n composed of unfolded nodes for all possible bindings for each component of $Dom(n)$. It results in a P/T net where all color classes have disappeared. Partial unfolding of the same node for color class C_u is a set of $\prod_{c \in Dom(n) | c = C_u} |c|$ nodes, composed of all possible bindings for each component of $Dom(n)$ of color class C_u.

Full unfolding is a special case of partial unfolding, obtained by the successive partial unfoldings, in any given order, over all color classes. A colored net N is fully unfolded to a net $N' = unfold(N) = \circ_{c \in C} unfold_c(N)$. Differences between full and partial unfolding are presented in algorithms of this paper.

4.2 Optimized Unfolding

The number of unfolded nodes in a net can be huge, especially when color classes contain a lot of elements, or when domains contain a lot of color classes. Unfolding of the Train [6] model generates more than 10^9 transitions. Such a net cannot have a concrete representation in memory and is almost useless if its size is not reduced.

Optimizations can be applied to unfolded nets to reduce their size, but our previous unfolder needed a concrete representation of the unfolded net to perform these optimizations. When no concrete representation of the unfolded net was possible because of its size, the optimizations needed to reduce its size could not be applied.

The new unfolder described here offers a solution to this problem by the substitution of the concrete representation of the unfolded nets with a symbolic one

for their places and transitions, and an implicit one for arcs. Optimizations can then be applied to huge unfolded nets and, in some cases, their size after optimization can be small enough for a concrete representation generation, usable by other tools.

4.3 Symbolic Unfolded Net

Symbolic representation of the unfolded net must be compact and allow fast operations. These two goals being more likely to be achieved using short decision diagrams, the representation chosen uses several decision diagrams, one for each colored place and one for each colored transition.

Symbolic representation of sets for unfolded places and transitions is done using DDD. This structure is well adapted to the representation of sets of integer vectors, and thus sets of vectors of color class values. Each sequence, in the representation of unfolded nodes from a node, is interpreted as one unfolded node. Integer vectors represent then, for an unfolded place or transition, the values in color classes associated to this particular object.

A **symbolic unfolded net** is defined above a Well-Formed Net by adding:

- $SVP : P \times (C \times \mathbb{N}) \rightarrow SV$ and $SVT : T \times V \rightarrow SV$: functions returning for each couple (*place* $\times$ *domain component*) (the domain component being the color class and its occurrence number in the domain), resp. (*transition* $\times$ *valuation variable*), the DDD variable in the symbolic representation,
- $SP : P \rightarrow \mathbb{DD}$ and $ST : T \rightarrow \mathbb{DD}$: for each $p \in P$ (resp. $t \in T$), its unfolded places (resp. transitions) represented using a DDD,
- $M'_0 : P \rightarrow \mathbb{DD}$: for each $p \in P$, the DDD of initially marked unfolded places.

Let us note that arcs are not explicitly represented in a symbolic unfolded net. Therefore, there is no direct encoding of Pre and $Post$ functions as usually: arcs are computed on the fly. To do so, four functions are defined to get pre-conditions or post-conditions of a node :

$$Pre_T(t) = \{p \in P | Pre[p,t] \neq \emptyset\} \qquad Post_T(t) = \{p \in P | Post[p,t] \neq \emptyset\}$$
$$Pre_P(p) = \{t \in T | Post[p,t] \neq \emptyset\} \qquad Post_P(p) = \{t \in T | Pre[p,t] \neq \emptyset\}$$

Partial unfolding is supported by this definition as the symbolic representation only gives an information about the presence of a node, still colored or not, in the unfolded net. Algorithms are presented for a full unfolding, because notations for partial unfolding can be less readable, but they also work for partial or are easily extended to achieve this goal.

We now introduce the notation $DDD_p = SP(p)$ (resp. $DDD_t = ST(t)$) for the symbolic representation of P'_p (resp. T'_t) the unfolded places (resp. transitions) from p (resp. t). A shorter notation is used for $SVP(p, c)$: it is the DDD variable for the component c, color class and occurrence number of it in $Dom(p)$, of place p.

The chosen representation only describes the presence of unfolded places or transitions and the structure of their DDD representation, and thus avoids a

representation of unfolded arcs, because the operations on the net, *Pre* and *Post* matrices or guards, are represented using homomorphisms. As these homomorphisms are operations suited to only particular purposes, for example interpreting some guards or getting places pre-condition of several transitions, they are not part of the definition of the symbolic unfolded net.

The unfolder is basically divided in three parts : the translation from a Well-Formed Net to a symbolic unfolded net and the definition of operations on this symbolic unfolded net, the application of the operations on the symbolic representation, and the translation from a symbolic unfolded net to a Well-Formed or P/T Net.

4.4 Construction of Symbolic Representation

The symbolic representation DDD_p of unfolded places P'_p from a place p is built recursively as shown in Algorithm 1. Unfolded places are initially represented with the DDD `1`, meaning that an already non-colored place unfolded for some color class or a colored place unfolded for no color class is unfolded as itself. Then, for each component c_i of $Dom(p)$ to unfold, its corresponding DDD variable $SVP(p, c_i)$ is added on top of place DDD, linked with one arc for each value in the color class of c_i to previously built place DDD. The same applies to construction of unfolded transitions from a transition t, but each transition variable $v \in TVar(t)$ gives a corresponding DDD variable with $SVT(t, v)$.

This algorithm defines an order in the DDD, but this order has no influence on algorithms presented in this paper as only DDD variables are used in operations. However, the implementation deals with this problem by enabling other orders.

Algorithm 1. Symbolic unfolded net construction

Require: $P \subset N$
Ensure: $\forall p \in P, DDD_p$
  for all $p \in P$ **do**
    DDD_p := `1`
    for all $c \in Dom(p)$ **do**
      if $is\_unfolded(c)$ **then**
        DDD_p := $\sum_{x \in c} SVP(p, c) \xrightarrow{x} DDD_p$
      end if
    end for
  end for

Require: $T \subset N$
Ensure: $\forall t \in T, DDD_t$
  for all $t \in T$ **do**
    DDD_t := `1`
    for all $v \in TVar(t)$ **do**
      if $is\_unfolded(VDom(v))$ **then**
        DDD_t := $\sum_{x \in VDom(v)} SVT(t, v) \xrightarrow{x} DDD_t$
      end if
    end for
  end for

4.5 Reconstruction of Explicit Representation

Construction of an explicit unfolded net from a symbolic representation is done, as shown in Algorithm 2, by creating a place for each path in the decision diagrams of unfolded places, and creating a transition for each path in the DDD

of unfolded transitions. The `0` DDD means no unfolded place or transition exists, `1` means the colored (or not) place or transition is kept identical in unfolded net, and other symbolic representations, except $\top$ which should never happen, mean unfolded places or transitions have to be created. The algorithm presented is valid only for full unfolding, but can be easily extended for partial unfolding. The algorithm is the same for transitions as for places, and the same for post arcs than for pre ones.

Algorithm 2. Concrete unfolded net construction

Require: N, `the symbolic unfolded Net`
Ensure: $N' = \langle P', T', Pre', Post', M0' \rangle$, the concrete unfolded net

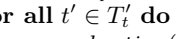
Unfolded places

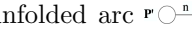
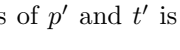
for all $p \in P$ **do**
  for all $path \in DDD_p$ **do**
    P'_p := $P' \cup \{path\}$
  end for
  P' := $P' \cup P'_p$
end for

Unfolded initial marking

for all $p \in P$ **do**
  for all $p' \in P'_p$ **do**
    if $M_0(p)(p') \neq 0$ **then**
      M'_0 := $M0' \cup \{(p', M_0(p)(p'))\}$
    end if
  end for
end for

Unfolded Pre arcs

for all $arc =$  $\in Pre$ **do**
  for all $p' \in P'_p$ **do**
    for all $t' \in T'_t$ **do**
      n := $valuation(p', t')$
      if $n \neq 0$ **then**
        Pre' := $Pre' \cup \{$  $\}$
      end if
    end for
  end for
end for

For each colored arc  $\in Pre$, an unfolded arc  is created if the evaluation of *valuation* for unfolded values of p' and t' is non-zero. The same applies for colored arcs $\in Post$.

For each place $p \in P$, its initial marking has a symbolic representation DDD_p^{marked} in the same way as DDD_p. For each unfolded place $p' \in P'_p$, its initial marking is created if a path in DDD_p^{marked} corresponding to p' is found.

5 Optimizations on the Unfolded P/T Net

Optimizations applied to the symbolic unfolded net are described in the next subsections. First of them is a simple one, the removal of `false` guarded transitions. Then, a more powerful but complicated one is presented, the removal of maximal unmarked syphon. A last optimization, the removal of marked orphaned places, is not presented in this paper as it uses the same operations as the maximal unmarked syphon.

Homomorphisms are not described in this paper because of lack of space, but are available to the reader on request to the authors.

5.1 Optimization Order

Definition 8 (Partial order over optimizations). *$\prec$ is a partial order operator over optimizations defined by $o_i \prec o_j$ iff o_j applied on an unfolded net previously optimized by o_i has a different result from o_j applied on an unoptimized unfolded net.*

Guard $\prec$ *Syphon*, as maximal unmarked syphon reduces the unfolded net because some transitions can never be fired, either because one of their input places cannot be marked or because the bindings of the transition variables lead to a `false` guard.

Guard $\prec$ *Orphan* and *Syphon* $\prec$ *Orphan*, as removal of orphaned marked places has no effect until some transitions have been removed.

For each optimization *Opt*, the best result is obtained when the length of the path $Opt_0 \prec \ldots \prec Opt$ is maximal. Using this constraint and this partial order, the order for optimization application is : removal of `false` guarded transitions, removal of maximal unmarked syphon and removal of orphaned marked places.

5.2 Removal of `false` Guarded Transitions

A first optimization is to remove, from the unfolded net, the transitions that are `false` guarded. Guards are used in Well-Formed Nets to enable or disable unfolded transitions using unfold bindings of transition variables. Each unfolded transition has a unique set of bindings that can be used for guard evaluation to remove bindings leading to a `false` guard.

A guard is an expression tree, where nodes have different arities. Nodes and leaves do not cover all the range of syntactic tokens expressing guards, for example $x > y$ or $x \leq y$, as a reduced set of operators described in Figure 4 is sufficient.

Name	Type	Arity	Meaning	
`OR`	node	2	$ltree \vee rtree$	where $ltree, rtree$ are subtrees
`AND`	node	2	$ltree \wedge rtree$	where $ltree, rtree$ are subtrees
`NOT`	node	1	$\neg tree$	where $tree$ is a subtree
`EQ`	leaf	-	$lvar = rvar$	where $lvar, rvar$ are variables
`LT`	leaf	-	$lvar < rvar$	where $lvar, rvar$ are variables
`IN`	leaf	-	$var \in vals$	where var is a variable and $vals$ a set of values
`TRUE`	leaf	-	$true$	
`FALSE`	leaf	-	$false$	

Fig. 4. Nodes and leaves in a guard tree

Application of a guard is an evaluation of the guard expression tree using bindings for all transition variables. The symbolic operation follows the same algorithm with one more indirection level, by creating for each guard an homomorphism to select sequences of bindings that evaluate to `true`. The same homomorphism is easily extended to select only non-`false` guards in partial unfolding, where some variables are not bound.

Creation of the homomorphism follows the guard expression tree, each node or leaf being translated to an homomorphism as described in the following paragraphs. One node, `NOT` raises a problem in its definition because its meaning is to keep all paths except selected ones. It thus translates to the difference between two symbolic representations, the full representation of all possible bindings and the representation of selected ones. We chose to avoid the difference operation by pushing negation down the tree using De Morgan's laws.

`AND` nodes are translated to the $\circ$ homomorphism operator. As the meaning of $\circ$ used here is $\wedge$, evaluation of subtrees can be done in any order. `OR` nodes are as easy to translate, by the $+$ homomorphism operator. Whereas two leaves, `TRUE` and `FALSE` have a direct translation by Id and $Constant(0)$, which always returns the DDD `0`, other leaves of the form $v_l{+}{+}s_l = v_r{+}{+}s_r$, $v_l{+}{+}s_l < v_r{+}{+}s_r$ and $v{+}{+}s \in C$ have no predefined homomorphisms or operations corresponding and are thus translated to inductive homomorphisms.

In partial unfolding, a guard tree leaf applied on at least one unbound variable is considered always non-`false`, as later binding of the variable can still lead to true of false. The Id homomorphism is used in this case instead of real leaf operation to keep the paths concerned with these special leaves.

5.3 Removal of Maximal Unmarked Syphon

The maximal unmarked syphon is a subset of the places of the unfolded net that cannot structurally be marked, and thus can be safely removed. Transitions post-condition of these places may be removed simultaneously because they can never be fired. Depending on the colored net, this syphon can be negligible, or almost cover the whole unfolded net.

Algorithm for P/T Nets. The algorithm for removal of the maximal unmarked syphon is based on an iterative construction, by removing places that can be marked from a superset S of the syphon, until stability. It is composed of three steps, described below.

1. Initially, the considered set is composed of all the unfolded places except the initially marked ones.
   $S = \cup_{p \in P}(p' \in P'_p | M'_0(p') = \emptyset)$.
2. Until stability, reduction is applied on S. For each unfolded transition $t' \in T'_{t \in T}$, if all its input places are outside the syphon, $Pre_T(t') \cap S = \emptyset$, then the transition can structurally be fired and all its output places are removed from the syphon, $S \leftarrow S \setminus Post_T(t')$. If no place is removed from the syphon in one iteration, then stability has been reached and the algorithm comes to its ending step.
   $\forall t \in T, \forall t' \in T'_t, (Pre_T(t') \cap S = \emptyset) \implies S \leftarrow S \setminus Post_T(t')$
3. The algorithm is ended by removing the syphon from the unfolded net. All transitions post-condition of places in syphon are removed.
   $\forall p \in P, P'_p \leftarrow P'_t \setminus S$
   $\forall t \in T, T'_t \leftarrow T'_t \setminus \{t' \in T'_t | Pre_T(t') \cap S \neq \emptyset \vee Post_T(t') \cap S \neq \emptyset\}$

Algorithm for Data Decision Diagrams. The algorithm of the removal of maximal unmarked syphon can be easily extended to the manipulation of sets of nodes, and sets represented by DDDs. In the following paragraphs, we use the notation DDD^i_n for the symbolic representation at step i of unfolded node n, meaning that $SP(n) = DDD^i_n$ (for a place) in the symbolic unfolded net at this step. The three steps of the algorithm become :

1. $\forall p \in P, DDD^0_p \leftarrow DDD_p \setminus M'_0(p)$
2. until stability :
   $\forall p \in P, DDD^i_p \leftarrow DDD^{i-1}_p \setminus \cup_{t \in T} HPost^T(DDD_t \setminus \cup_{p' \in P} HPost^P(DDD^{i-1}_{p'}))$
3. $\forall p \in P, DDD^n_p \leftarrow DDD_p \setminus DDD^{n-1}_p$
   $\forall t \in T, DDD^n_t \leftarrow DDD_t \setminus \cup_{p \in P}(HPre^P(DDD^{n-1}_p) \cup HPost^p(DDD^{n-1}_p))$

$HPre^P$, $HPre^T$, $HPost^P$ and $HPost^T$ are homomorphisms defined to apply respectively Pre_P, Pre_T, $Post_P$ and $Post_T$ on sets of places or transitions. These homomorphisms can be divided in two groups : $HPre^P$, $HPost^P$ and $HPre^T$, $HPost^T$, based on their input and output types, these group are described in the following paragraphs.

To ease reading of this paragraph, we take the example of $HPre^P$, but the same applies for the four homomorphisms. $HPre^P$ applies on all unfolded places P' of the net. It is defined using sub-homomorphisms for each unfolded place set $P'_p | p \in P$ by the homomorphism $HPre^P_p$, which returns the set of unfolded transitions pre-condition of places unfolded from p. If not applied on places in P'_p, it returns `0`. $HPre^P_p$ is itself divided into sub-homomorphisms for each arc a from a transition t to p, using a $HPre^P_a$ homomorphism. As the valuation of an arc is a sum of terms, $valuation = \sum_i v_i$, the latter homomorphism can be cut into smaller ones noted $HPre^P_v$, considering each valuation term as an arc. Operation for combination of sub-homomorphism is either $\circ$ or $\cup$, depending on the searched result.

$HPre^P = \cup_{p \in P} \cup_{t \in T | Post[t,p] \neq \emptyset} \cup_{v \in Post[t,p]} HPre^P_v$ is the homomorphism returning the set of unfolded transitions pre-condition of a set of unfolded places. If $Post[t, p] = \emptyset$, the unfolded arc does not exist and no set is returned, using the DDD `0`, implicitly represented in the $\cup$ operators.

Removal of Initially Marked Places. For each colored place p, a DDD representing only its marked unfolded places DDD^{marked}_p can be built, using the same symbolic representation as DDD_p. The operation to remove marked unfolded places is $DDD^0_p = DDD_p \setminus DDD^{marked}_p$.

A colored mark is a tuple, $mark(p) = \langle c_1, \ldots, c_n \rangle$, with one c_i value for each component of $Dom(p)$. In partial unfolding, if c_i is of a non unfolded color class C_j, it is not used in the symbolic representation of unfolded places and thus not used in the marking representation.

For a place where no color class has to be unfolded, the unfolded marking is `0` for no marking, `1` for a marking.

***HPre*$_v^P$ and *HPost*$_v^P$ Homomorphisms.** A colored arc is translated to an homomorphism $HPost_v^P$ if applied from DDD_p^i to a subset of DDD_t, to get the transitions post-condition of places in DDD_p^i. Almost the same homomorphism $HPre_v^P$ is created for the reverse arc, to get transitions pre-condition of places, and only the small difference with $HPost_v^P$ is given.

These two operations are a composition of six homomorphisms, each one being a step in the transformation. Some steps can be grouped to improve efficiency by reducing the number of operations, but these optimizations are not presented here. The steps can be divided into two main groups : the selection of places that are valid inputs of the valuation (*1, 2*) and the conversion of these places to the symbolic representation of output transitions (*3, 4, 5, 6*).

Each step is itself composed of several homomorphisms, each one defined to be applied for only one color class, not the whole domain. The composition for several color classes is : $HPre_v^P = \circ_{term \in v} HPre_{term}^P$. We consider a colored arc from transition t to place p (Pre_v^P), or from place p to transition t ($Post_v^P$), valued with term $term$.

Select valid places for valuation constants. If $Dom(p) = c_0 \times ... \times c_n$, for each v_i of $term = \langle v_0, \ldots, v_n \rangle$, if c_i is an unfolded color class and v_i is a constant, then the value of v_i is compared to the unfold values of P'_p for component c_i, assignment values of $SVP(p, c_i)$ DDD variable, to keep only the unfolded places matching the constant.

DDD variables involved in this step are useless in symbolic representation after the selection, because their values have no relation with values of valuation variables of t, and are thus removed.

Select places consistent with valuation variables appearing several times. A single variable can be referenced several times in one term. For each couple $c_i, c_j \in Dom(p)$ where $v_i = var{+}{+}s_i$ and $v_j = var{+}{+}s_j$ refer the same unfolded valuation variable var, only the unfolded places where $SVP(p, c_i)$ has the same value as $SVP(p, c_j)$ shifted with $s_i - s_j$ are kept in symbolic representation.

As values of $SVP(p, c_i)$ and $SVP(p, c_j)$ are linked, only one occurrence is necessary for the next steps, the other is removed.

Remove successor ranks. Successor ranks have been used in previous step to check consistency, but valuation variables do not use them. As each variable left in symbolic representation corresponds to a component $v_i = var{+}{+}s_i$ of the term, an homomorphism transforms each value of $SVP(p, c_i)$ to its s_i predecessor for the $HPost_{term}^P$ version, and to its s_i successor for the $HPre_{term}^P$ one, to keep only the value without its successor rank.

Transform place DDD to transition DDD. In the previous steps, we considered the symbolic representation of DDD_p. From now one, we consider the one of DDD_t. The transformation begins in this step, by renaming for each remaining $v_i = var$ the variable $SVP(p, c_i)$ to $SVT(t, var)$.

Reorder DDD variables to fit transition DDD order. All transition variables in the valuation are in the symbolic representation, but their order differs of the order of transition variables in symbolic representation of the unfolded transitions. A reordering is thus done in this step to fit the order of the transition.

Complete transition DDD. For each variable $v \in TVar(t)|VDom(v)$ is unfolded, missing in symbolic representation, $SVT(t,v) \xrightarrow{x \in VDom(v)} \texttt{1}$ is added in its right place in the symbolic representation.

$HPre_v^T$ and $HPost_v^T$ Homomorphisms. A colored arc is translated to an homomorphism $HPost_v^T$ if applied from DDD_t to a subset of DDD_p^i, to get the places post-condition of transitions in DDD_t. Almost the same homomorphism $HPre_v^T$ is created for the reverse arc, to get places pre-condition of transitions.

As for $HPre_v^P$ and $HPost_v^P$, $HPost_v^T$ is a composition of four steps and divided in subhomomophisms for each term of v. We consider a colored arc from transition t to place p ($Post_v^T$), or from place p to transition t (Pre_v^T), valued with term $term$.

Copy transition DDD variables to their bound place DDD variable. For each unfolded transition variable $var \in TVar(t)|VDom(var)$ is unfolded, if $Dom(p) = c_0 \times \ldots \times c_n$, and $term = \langle v_0, \ldots, v_n \rangle$, for each $v_i \in term$, if $VDom(var)$ is c_i then the DDD values for $SVT(t,var)$ are copied to the DDD variable $SVP(p,c_i)$.

Remove unused DDD variables. All valuation variables represented in DDD have been copied, if used in the term, to place DDD variables, and are thus useless. This step removes them from symbolic representation. For each $var \in TVar(t)|VDom(var)$ is unfolded, $SVT(t,var)$ is removed from DDD.

Add successor ranks. Successor ranks have not been used in previous steps. As each variable in symbolic representation corresponds to a component $v_i = var{+}{+}s_i$ of the term, an homomorphism transforms each value of $SVP(p,c_i)$ to its s_i predecessor for the $HPre_{term}^T$ version, and to its s_i successor for the $HPost_{term}^T$ one.

Insert constants of valuation. As valuation can contain constants, new DDD variables are created to insert the constants in the symbolic representation. For each constant v_i in the term, $SVP(p,c_i) \xrightarrow{v_i}$ is inserted in its right place.

6 Experimentation and Results

Implementation. Optimized unfolder has been implemented as a library and an executable built on top of this library. Unfolded net follows a pipe of optimizations, like those seen in sections 5.2 and 5.3. The symbolic to explicit net transformation presented in section 4.5 is also a component of this pipe.

Experimentation. We selected several models to validate our strategy:

- *PolyORB* : it comes from a cooperation with Telecom Paris to build PolyORB, a formally verified Middleware [7]. In our test, the model is parameterized according to the number of threads that are allocated in the system.
- *Peterson* : it models the Peterson's mutual exclusion algorithm for N processes [10]. This model does not include any redundant information to test processes level. This fact explains the high number of transitions. In our test, the model is parameterized according to the number of processes.
- *Train* : It comes from a joint studies on the San Francisco area BART case study, that was presented in [6]. This model is not parametrized.

Model	Initial		Guards		Syphon		Orphans		Optimized			
	Places	Transitions	% P	% T	% P	% T	% P	% T	Time	Memory	Places	Transitions
PolyORB_t20	4,964	3,392,800	0	100	0	0	0	0	< 1	11	4,964	3,392,040
PolyORB_t40	9,344	6,786,800	0	100	0	0	0	0	< 1	20	9,344	6,784,860
PolyORB_t60	13,724	10,182,400	0	100	0	0	0	0	1	29	13,724	10,178,480
PolyORB_t80	18,104	13,579,600	0	100	0	0	0	0	1	41	18,104	13,572,900
Peterson_05	1,150	58,850	0	55	100	45	ε	0	< 1	10	470	7,810
Peterson_10	9,100	1,835,400	0	59	100	41	ε	0	3	73	3,890	313,220
Peterson_15	30,600	13,838,400	0	61	100	39	ε	0	10	194	13,260	2,576,730
Train	10,620	1.4072e+09	0	61	99	39	1	0	5	111	343	202

Fig. 5. Execution results from our benchmark

Figure 5 summarizes the results we got on a 3.6 GHz Pentium-4 CPU with 2 GBytes of RAM. We provide the maximum number of places and transitions (columns *Initial Places* and *Initial Transitions*), the final number of places and transitions after optimizations (columns *Optimized Places* and *Optimized Transitions*). The explicit unfolded net is not generated, as for all examples but `Train` execution time would be hours or days. We also measure the execution time (in seconds) and the maximum amount of memory required to process the unfolding (in Mbytes). We also provide the ratio of deleted places and transitions for every listed optimizations (for example, 100% of deleted transition in one column means that all suppressed transitions are detected by this optimization).

Our first test (PolyORB) shows the capacity of our approach to take care of numerous objects in a very short time. This Petri net is very symmetric (all threads in the system can exchange their role) and that explains the very poor performances of the implemented optimizations (only the guard optimization is activated for a very small number of transitions). These symmetries also explain the very low amount of memory required to store that many Petri net objects (this is a typical advantage of decision diagram based representations).

Our second model (`Peterson`) only has two colored places (one for processes, another one for the level variables that store the last process to enter in each level). States of the processes are modelled as a class. Therefore, all classes of the domain place depicting the processes are not always useful (e.g. used to scan all processes), leading to never marked places in the unfolded net. For that types of specifications, optimization factor is excellent since up to 80% of dead transitions are eliminated.

Our last model (`Train`) provides spectacular performances on the optimization rate (99,99999985% of transitions as well as 99,995% of places are deleted). This is due to the presence of $<$ and $>$ in transition guards as well as the use of a modeling technique to express discretized non linear function (e.g. the braking distance according to a vehicle speed). A function $y = f(x_1, ..., x_N)$ is modeled with a place having a $N+1$ product color class. This very large product, as well as that color domains usually are large, generates numerous P/T corresponding places for which only a few are marked (the initial marking only has a few of the possible values). As shown here, the corresponding P/T net model, before optimization, contains billions of places and transitions that do not remain after optimization.

The removal of orphaned marked places has poor results. It only removes a few places in the `Peterson` and `Train` models, and none in PolyORB. This optimization is only intended to clean the unfolded net after removal of the maximal unmarked syphon, and to inform the modeller about unused marked places, as these places might be a modelling error.

Comparison with Maria. We now compare our work to the unfolder included in Maria [9]. It is based on different principles : unfolding is viewed as related to the colored transition enabling test. Compact structures as BDDs or DDDs are not used : a unique explicit colored marking is built using the following rules:

- a place color is or is not in the marking, no cardinality is taken into account,
- all places colors in the initial marking are in the unique marking,
- if a transition is enabled with colors in the unique marking (here again, cardinalities are not taken into account), then all post-condition places colors are added to the unique marking.

Place colors in the unique marking are translated into ordinary places and corresponding transitions and arcs are computed. It can easily be shown that place colors not translated into ordinary places corresponds to our maximal unmarked syphon.

Performance differences between our symbolic unfolder and Maria have several explanations. Firstly, colored transition enabling test is more efficient than our symbolic manipulations : the latter one uses heavy DDD variable reordering. Secondly, our unfolder can use a lot of memory for some models, whereas Maria always keeps a very low memory usage, but maria is not able to extract information from huge optimized unfolded nets, like PolyORB, because Maria's strategy, in that case, leads to and explicit representation of the unfolded net.

The main difference between our unfolder and Maria is a consequence of the different chosen approaches : symbolic for our tool and explicit for maria. As expected, explicit approach has better results on small unfolded nets. For the `Train` model, Maria ends its execution after 20 seconds using less than 1 MBytes. This is equivalent in time (since Maria generates the output, which takes time) and much lower from the memory use than our symbolic approach. However, for the other examples, whereas no other operation can be done until the full

unfolded net is generated for Maria, which can take hours, our symbolic unfolded net enables some operations without generation of its explicit representation. These new operations are to be defined in the future.

7 Conclusion

To complete some structural analysis of colored Petri nets, it is important to back-track to equivalent P/T nets and use structural techniques that are not yet available for colored Petri nets. This is important, especially when dealing with very large models deduced from higher specification languages. Our experiments show that it really supports very large unfolded models.

The technique presented in this paper relies on data decision diagram for a very compact internal representation. This is an original use of such techniques in a new application domain. It provides good perspectives in the handling of large system specifications.

Another advantage of our symbolic unfolding technique is that it is suitable for partial unfolding of a subset of the color domains. This is of interest to discard some useless color domains or increase the specification symmetry for analysis purpose. So far, no unfolder from High Level Petri Nets to P/T nets offers this possibility.

The presented technique is implemented and already available in CPN-AMI 3.0 as a beta version tool.

References

1. S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In *WOSP '02: Proceedings of the 3rd int. workshop on Software and performance*, pages 35–45. ACM Press, 2002.
2. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.
3. G. Chiola, C. Dutheillet, G. Franceschini, and S. Haddad. On Well-Formed Coloured Nets and their Symbolic Reachability Graph. *High-Level Petri Nets. Theory and Application, LNCS*, 1991.
4. J-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P-A. Wacrenier. Data decision diagrams for petri net analysis. In J. Esparza and C. Lakos, editors, *Applications and Theory of Petri Nets 2002*, number 2360 in LNCS, pages 101–120. Springer Verlag, June 2002.
5. J-M. Couvreur, S Haddad, and J. F. Peyre. Generative families of positive invariants in coloured nets sub-classes. In G. Rozenberg, editor, *Applications and Theory of Petri Nets*, LNCS, pages 51–70, 1991.
6. A. de Groot, J. Hooman, F. Kordon, E. Paviot-Adet, I. Vernier-Mounier, M. Lemoine, G. Gaudiere, V. Winter, and D. Kapur. A survey: Applying formal methods to a software intensive system. In *6th International Symposium on High-Assurance Systems Engineering*, pages 55–64. IEEE Computer Society, 2001.
7. J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *Proceedings of the 9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, volume ENTCS 133, pages 139 – 157. Elsevier, Sept. 2004.

8. K. Jensen. Coloured Petri nets and the invariant-method. *Theor. Comput. Sci.*, 14:317–336, 1981.
9. M. Mäkelä. Optimising enabling tests and unfoldings of algebraic system nets. In *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets*, number 2075 in LNCS, pages 283 – 302. Springer-Verlag, 2001.
10. Gary L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.
11. R. Valk. *Basic definitions*, chapter 4, pages 41–51. Springer Verlag, Petri nets and system engineering (Claude Girault and Rudiger Valk Eds), first edition, 2003.
12. C. Dutheillet Y. Thierry-Mieg and I. Mounier. Automatic Symmetry Detection in Well-Formed Nets. In W. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003, Proceedings*, number 2679 in LNCS, pages 82–101. Springer Verlag, June 2003.