24

MODEL CHECKING BASED ON OCCURRENCE NET GRAPH

Jean-Michel Couvreur[†], Denis Poitrenaud[‡]

[†] Institut d'Informatique d'Entreprise, CEDRIC-IIE
18, Allée J. Rostand, 91025 Evry Cedex, FRANCE
couvreur@iie.cnam.fr

[‡] Laboratoire MASI, Institut Blaise Pascal, Université Paris VI
4, Place Jussieu, 75252 Paris Cedex 05, FRANCE
Denis.Poitrenaud@masi.ibp.fr

Abstract

The computation of a reachability graph is one of the most used method to check system properties. Its main drawback is the state explosion. Two different approaches are generally used to tackle this problem: by defining new concise graph representations for which verification methods are adapted; by reducing graphs while preserving observed properties. We propose a new representation of a reachability graph where nodes are particular Petri nets, occurrence nets, characterizing parts of the state space. Checking invariant properties can be done using efficient algorithms for occurrence nets. Moreover, our representation can be used to obtain a stuttering equivalent graph on which nexttime-less linear temporal formulae are verified.

Keywords

Verification, temporal logic, Petri nets

1. INTRODUCTION

Temporal logic model checking is a useful method for verifying properties of finite concurrent systems. Many algorithms have been developed that depend on the representation of properties (temporal logic formula or Büchi automaton). The reader can find a complete survey of this domain in [Wol89].

However, due to the state-explosion problem these methods tend to be impossible for the verification of realistic systems. Different approaches are generally used to combat this problem. One way is to define new concise graph representations for which there are adapted verification methods. For instance, we can mention methods based on Binary Decision Diagram [Clark94] [Past94] and unfoldings [Nielsen80][McM92][Esp96]. A complementary approach is to reduce reachability graphs while preserving observed properties. Among them, we can note Petri net reductions [Berth85], stubborn set method [Val91] and more generally semantic equivalences [Lam83][KV92].

In this paper, we propose a new representation of a reachability graph for safe Petri nets where nodes are particular nets, occurrence nets that characterize parts of the state space. This

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

class of net has been already used for unfolding Petri nets [Nielsen80]. In [McM92], unfoldings are used to check invariant properties and to dectect deadlocks. [Esp93] extends this results to a branching time temporal logic. The main shortcoming of this logic is the fact that liveness properties can not be expressed.

The main improvement of our method against the unfolding technique is its capability to preserve infinite sequence. Moreover, our representation can be used to obtain equivalent graphs according to the stuttering equivalence introduced in [Lam83][KV92]. This result induces a model checking for the nexttime-less linear temporal logic. Roughly speaking, given an occurrence net graph, checking a formula is performed in two steps: the computation of a stuttering equivalent graph with respect to the atomic propositions; the verification of the formula on the resulting graph.

This paper is organized as follows. In Section 2, basic notions on nexttime-less linear temporal logic and Petri nets are presented. Section 3 introduces occurrence nets and some specific properties. Section 4 contains the theoretical aspect of our model checking, the definition of occurrence net graphs and their key properties. Construction algorithms are given in Section 5. Section 6 concerns the model checking. Section 7 gives experimental results of our method followed by a comparison with other techniques and a discussion.

Proofs of theorems and algorithms are technical and use intensively the elementary properties of occurrence nets. Hence, they have been omitted for reason of space and clarity.

2. PRELIMINARIES

2.1. Nexttime-Less Linear Temporal Logic

We assume that there is a set AP called the set of atomic propositions. We denote a transition system H by (S, X, L, s_0, *terminal*) where S is the finite state set, $X \subseteq S \times S$ is the transition relation specifying the single step of a system, $L(s) \subseteq AP$ is the labelling of each state s with true propositions, $s_0 \in S$ is the initial state and *terminal* is a state proposition which is at least true for deadlock (i.e. state with no successor).

α is a finite sequence from w_0 of H iff $\alpha = s_0 s_1 \ldots s_{n-1}$ such that for all $0<i\leq n$, $(s_{i-1}, s_i) \in X$. α is an infinite sequence from s_0 of H iff every finite prefix of α is a finite sequence from s_0 of H. We denote by $\alpha^{(i)}$ the string obtained by leaving the first i elements of α.

We call computations infinite sequences from the initial state and finite sequence from the initial state to a *terminal* state.

We have chosen a linear temporal logic derived from one of the most common versions of propositional temporal logic appearing in the computer science literature and presented in [KV92]. Our logic does not accept the use of the temporal operator «nexttime» $\bigcirc$ but is augmented by an additional operator $\Diamond^\infty$ «infinitely often in the proper future». In other respects the definitions of our linear logic are standard and we will refer to it as LTL^∞.

LTL^∞ formulae are built from atomic propositions $f_1, f_2, \ldots$ on states, boolean connectives ($\wedge$, $\neg$) and temporal operators (U «until», $\Diamond^\infty$ «infinitely often in the proper future»). The formation rules are:

- atomic propositions are formulae;
- if f and g are formulae, so are $f \wedge g$, $\neg f$, $f \, U \, g$, $\Diamond^\infty f$.

We use $\Diamond$f («eventually») as an abbreviation for (*true* U f) and $\Box$f («always») as an abbreviation for $\neg\Diamond\neg$f. We also use $\vee$ and $\Rightarrow$ as the usual abbreviations.

The formal semantic of LTL$^{\infty}$ is given as follows. Let $\alpha = s_0 \ldots s_i \ldots$ be a finite or infinite sequence from s_0 of a transition system H = (S, X, L, s_0, *terminal*). For a formula f, $\alpha \models$ f means that f is satisfied by α. We have

- $\alpha \models$ f iff f $\in L(s_0)$, for f an atomic proposition
- $\alpha \models f \wedge g$ iff ($\alpha \models$ f) and ($\alpha \models$ g)
- $\alpha \models \neg$f iff not $\alpha \models$ f
- $\alpha \models f\ U\ g$ iff $\exists$ i such that ($\forall j \leq i$, $\alpha^{(j)} \models$ f) and ($\alpha^{(i)} \models$ g)
- $\alpha \models \Diamond^{\infty}$ f iff there are infinitely many $i \geq 0$ such that $\alpha^{(i)} \models$ f

For a transition system H, a LTL formula f is satisfied (denoted by H $\models$ f) iff each computation satisfies f.

The trace notion where only the truth labelling function modifications are visible, induces the stuttering equivalence between systems preserving our logic. Formally, a trace for a finite sequence $\alpha = w_1 \ldots w_n$ is defined by:

- if n=1, $tr(s_1) = L(s_1)$
- if n>1 and $L(s_{n-1}) = L(s_n)$, $tr(\alpha) = tr(s_1 \ldots s_{n-1})$
- if n>1 and $L(s_{n-1}) \neq L(s_n)$, $tr(\alpha) = tr(s_1 \ldots s_{n-1}).L(s_n)$

The trace of an infinite sequence is the limit of the traces of its prefixes $tr(s_1 \ldots s_n)$.

Three sets characterize the stuttering equivalence: dtr(H) is the finite computation trace set; inftr(H) is the infinite trace set of infinite computations and divtr(L) is the finite trace set of infinite computations. Two systems are said stuttering equivalent iff they have the same characteristic trace sets. L. Lamport [Lam83] and R. Kaivola et al. [KV92] establish the relation between the stuttering equivalence and LTL$^{\infty}$.

Theorem 2.1 : Let H and H' be transition systems. H and H' are stuttering equivalent iff for any LTL$^{\infty}$ formula f, H $\models$ f $\Leftrightarrow$ H' $\models$ f.

2.2. Petri Nets

We assume that the reader is familiar with the basic notions and notations of Petri nets, as given for instance in [Mur89]. We use finite safe Petri nets as system models. We denote a Petri net by N = (P, T, F, M_0) where (P, T, F) is a net and M_0 its initial marking. A place of a safe Petri net contains at most one token and then markings are represented as place sets.

In the sequel, we use the following particular notations:

- N = (P(N), T(N), F(N), M_0(N)) is used to denote the different net components when several Petri nets are considered.
- R(N) is the reachability set of N.
- Deadlock(N) is is a state proposition which is true for deadlock state of N.
- $R_A(N) = \{M \in R(N): A \subseteq M\}$ is the coverability set for the place set A.
- M[t> is the resulting marking of M by t when t is firable.
- $(M,V) = (M \cup V^{\bullet} \cup {}^{\bullet}V, V, (V \times P \cup P \times V) \cap F, M)$ is the sub-net of N that represents the firing sequence from M using only transitions in V.
- N_L = (R(N), [>, L, M_0(N), Deadlock(N)) is the transition system of a Petri net N with

respect to a state labelling function L ([> denotes the transition relation of the Petri net N).

Example : The following figure shows a safe Petri net previously presented in [Reisig95]. It models a mutual exclusion between two processes *l* and *r*.

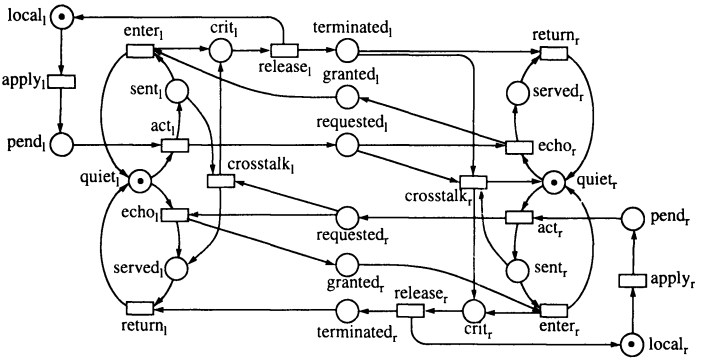

Figure 1 Round based mutual exclusion.

Site *l* may spontaneously *apply* for getting critical, by moving from $local_l$ to $pend_l$. Then *l* *acts*, i.e. sends a *request* and remains $sent_l$ until site *r* either *grants l* to *enter* $critical_l$ or site *r* also *requests* to go critical. In this case *l* goes *critical* along $crosstalk_l$. A second component of site *l* in state $quiet_l$ may *echo* a *request* from site *r*, *granting r* to go *critical*.

The reachability graph of this net contains 32 nodes and 59 arcs. On this safe Petri net, the set of atomic propositions AP = $\{c_l, c_r, s_l, s_r\}$ induces a linear temporal logic with $c_l \in L(M) \Leftrightarrow crit_l \in M$, $c_r \in L(M) \Leftrightarrow crit_r \in M$, $s_l \in L(M) \Leftrightarrow sent_l \in M$ and $s_r \in L(M) \Leftrightarrow sent_r \in M$. For instance, the formula $\Box(\neg c_l \vee \neg c_r)$ expresses the safeness property to access to the critical section.

3. Occurrence Nets

Occurrence nets are sub-class of safe Petri nets introduced in [Nielsen80] for unfoldings. In this section, we recall its definition, the notion of configuration and basic properties. We point out two essential problems which are relevant for the occurrence net graph construction:

1) the *coverability problem*: to characterize the reachable markings for which given places are marked,

2) the *cutup problem*: to represent the reachable marking set by the union of reachable markings of occurrence sub-nets in which given transitions do not appear.

More informations about occurrence nets can be founded in [Nielsen80][McM92][Esp96].

Definition 3.1 : An occurrence net is a marked Petri net N = (P, T, F, M_0) where:

1) $\forall\, p \in P,\ |{\bullet p}| \leq 1$ and $M_0 = \{p \in P: |{\bullet p}| = 0\}$

2) $\forall\, t \in T,\ |{\bullet t}| \geq 1$

3) The Petri net N is an acyclic graph

Example : The following figure shows a subnet of the net of Figure 1. This subnet is an occurrence net.

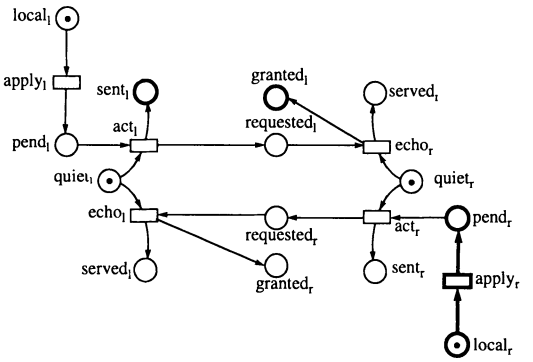

Figure 2 Occurrence net.

As K.L. McMillan notes in [McM92], «The most important theoretical notion regarding occurrence nets is that of a *configuration*». Configurations represent possible firing sequences of the net. It is based on the reflexive partial order induced by the graph which is acyclic and on the notion of transition conflict.

For a set of places and transitions U, we denote by $C_U(N)$ the set containing all the transitions of U and the ones that have a descendant in U.

Definition 3.2 : Let N be an occurrence net. A set of transitions S is a configuration iff:

(no conflict in S) $\forall\, t, t' \in S: {}^\bullet t \cap {}^\bullet t' \neq \emptyset \Rightarrow t=t'$

(S is downward closed) $S = C_S(N)$

Example : For the occurrence net of the Figure 2, $C_{echo_r}(g_0)=\{apply_l,act_l,echo_r\}$ is a configuration and then defines firing sequences. On the other hand, $C_{\{served_l,\, served_r\}}(g_0)$ is not a configuration because it contains, in particular, the conflict transitions $echo_r$ and act_r and then places $served_l$ and $served_r$ are in mutual exclusion in g_0.

Configurations simply characterize reachable markings and quasi-live transitions (transitions fired at least once).

Proposition 3.3 : Let N be an occurrence net. A marking M is reachable iff there exists a configuration S such that $M=(M_0 \cup S^\bullet)\setminus {}^\bullet S$. When these properties hold, we have $C_M(N) \subseteq S$ and if any transition in T has at least one output place then $C_M(N) = S$.

Proposition 3.4 : Let N be an occurrence net. Let t be a transition. t is quasi-live iff $C_t(N)$ is a configuration.

When any transition in T has at least one output place, the reachability algorithm associated to Proposition 3.3 is based on a graph traversal and its complexity is O(|F|). Otherwise the problem is NP-complete. The quasi-liveness algorithm for a transition has a complexity in O(|F|).

The coverability problem is «Does there exist a reachable marking including a place set A?». For our method, we need to represent the covering marking set $R_A(N)$. The following theorem characterizes it as the reachability set of an occurrence sub-net of N.

Theorem 3.5 : Let N be an occurrence net. Let A be a place set. We denote the covering marking set $R_A(N) = \{M \in R(N): A \subseteq M\}$. $R_A(N)$ is not empty iff $C_A(N)$ is a configuration and $\bullet C_A(N) \cap A = \varnothing$. If $R_A(N)$ is not empty, the sub-net of N, $(\min_A(N), T_A(N))$, where:

$\min_A(N) = (M_0 \cup C_A(N)\bullet)\backslash\bullet C_A(N)$

$T_A(N) = \{t \in T \mid C_A(N) \cup C_t(N)$ is a configuration $\wedge \bullet C_t(N) \cap A = \varnothing\}$

is a quasi-live occurrence net such that $R((\min_A(N), T_A(N))) = R_A(N)$.

It can be shown straightforwardly enough that the algorithm associated to coverability problem has a complexity in O(|F|).

Example : For the occurrence net g in the Figure 2, we have $\min_{\bullet enter_l}(g) = \{send_l, granted_l, local_r\}$ and $T_{\bullet enter_l}(g) = \{apply_r\}$. The subnet $(\min_{\bullet enter_l}(g), T_{\bullet enter_l}(g))$ is the bolded one in the Figure 2.

The cutup problem with respect to a transition set K, is to decompose the occurrence net in a set of occurrence sub-nets where:

1) the transitions in K do not appear in the sub-nets,

2) the reachable marking set of the original net is the union of the reachable marking sets of the sub-nets.

Theorem 3.6 : Let N be an occurrence net. Let I and K be transition sets with $I \subseteq K$. We denote the cutup marking set $R_K^I(N) = \{M \in R(N), \exists$ S a configuration: $M=(M_0 \cup S\bullet)\backslash\bullet S \wedge S \cap K = I\}$. $R_K^I(N)$ is not empty iff $C_I(N)$ is a configuration and $C_I(N) \cap K = I$. If $R_K^I(N)$ is not empty, the sub-net of N, $(\min_K^I(N), T_K^I(N))$, where:

$\min_K^I(N) = (M_0 \cup C_I(N)\bullet)\backslash\bullet C_I(N)$

$T_K^I(N) = \{t \in T \mid C_I(N) \cup C_t(N)$ is a configuration $\wedge C_t(N) \cap K = I\}$

is a quasi-live occurrence net such that $R((\min_K^I(N), T_K^I(N))) = R_K^I(N)$. Moreover, $R(N) = \bigcup_{I:I \subseteq K} R_K^I(N)$ and then the sub-nets $(\min_K^I(N), T_K^I(N))$, for which $R_K^I(N)$ is not empty, define a cutup of N with respect of K.

In general the complexity of this problem is exponential because of the number of subsets I in K for which $R_K^I(N)$ is not empty. However, in our construction, we often use the cutup with respect to a single transition, and its complexity is reduced to O(|F|).

Example : The cutup of the net g (in Figure 2) with respect to $K=\{act_l, act_r\}$ gives:

1) $\min_K^{\varnothing}(g) = M_0(g_0)$, $T_K^{\varnothing}(g) = \{apply_l, apply_r\}$

2) $\min_K^{act_l}(g) = \{sent_l, requested_l, quiet_r, local_r\}$, $T_K^{act_l}(g) = \{echo_r\}$

3) $\min_K^{act_r}(g) = \{sent_r, requested_r, quiet_l, local_l\}$, $T_K^{act_r}(g) = \{echo_l\}$

4) $\min_K^{K}(g) = \{requested_r, requested_l\}$, $T_K^{K}(g) = \varnothing$

These nets are presented in the Figure 3.

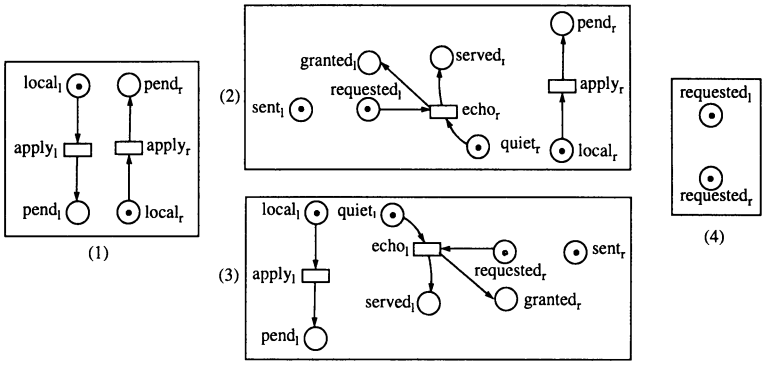

Figure 3 Cutup of a net.

4. Occurrence Net Graphs

This part contains the main results of this paper: the definition of occurrence net graph for safe Petri net and there basic properties - the preservation of reachable states, the deadlock characterization, and the stuttering equivalence. Briefly, an occurrence net graph is a representation of the reachability graph where nodes contain sub-graph and arcs link sub-graphs. Sub-graphs are expressed by occurrence nets that are sub-nets of the original one with distinct initial marking. The initial node has the same initial marking as the original net and then be a primary part of the reachability graph. To insure the stability of our representation, every firing from reachable states of nodes must be taken into account. Obviously, every firing of a transition belonging to the node is already represented. On the other hand, when a transition, not in the node, is enabled for some markings, the corresponding firing is represented by an arc in the graph. In this case, the covering marking set where the transition is enabled, is defined as an occurrence sub-net. To assume the stability in term of states, it is sufficient to take as the successor node, the one for which its initial marking is the firing result from the minimal state of the covering marking set. Indeed, the state set obtained by firing the transition from the covering marking set is represented for a part in the successor node and in the descendants of this node. However, for the LTL model checking point of view, it is essential to preserve in the representation of our graph all the firing sequences. To achieve this goal, complementary arcs are added to the nodes in which are distributed the considered markings.

First, are defined the occurrence net graphs which preserve reachable states.

Definition 4.1 : An occurrence net graph NPG = (G, X, g_0) of a safe net N is a graph where

(node property)	G is a set of quasi-live occurrence sub-nets of N
(transition relation)	$X \subseteq G \times T \times G$
(initial node)	$g_0 \in G$ and $M_0(g_0) = M_0(N)$
(unicity property)	$\forall\, g,g' \in G, M_0(g) = M_0(g') \Rightarrow g = g'$
(transition property)	if $t \notin T(g)$ and $(\bullet t \subseteq P(g)$ and-then $R_{\bullet t}(g) \neq \emptyset)$ then
	$\exists!\, g' \in G, (g,t,g') \in X$
	else $\forall\, g' \in G, (g,t,g') \notin X$

(stability property) $\forall\, g,g' \in G, t \in T: (g, t, g') \in X, \min_{\bullet t}(g)[t>M_0(g')$

(reachability property) $\forall\, g \in G$, there exists a path from g_0 to g

Example : The following figure presents an occurrence net graph of the safe net of Figure 1.

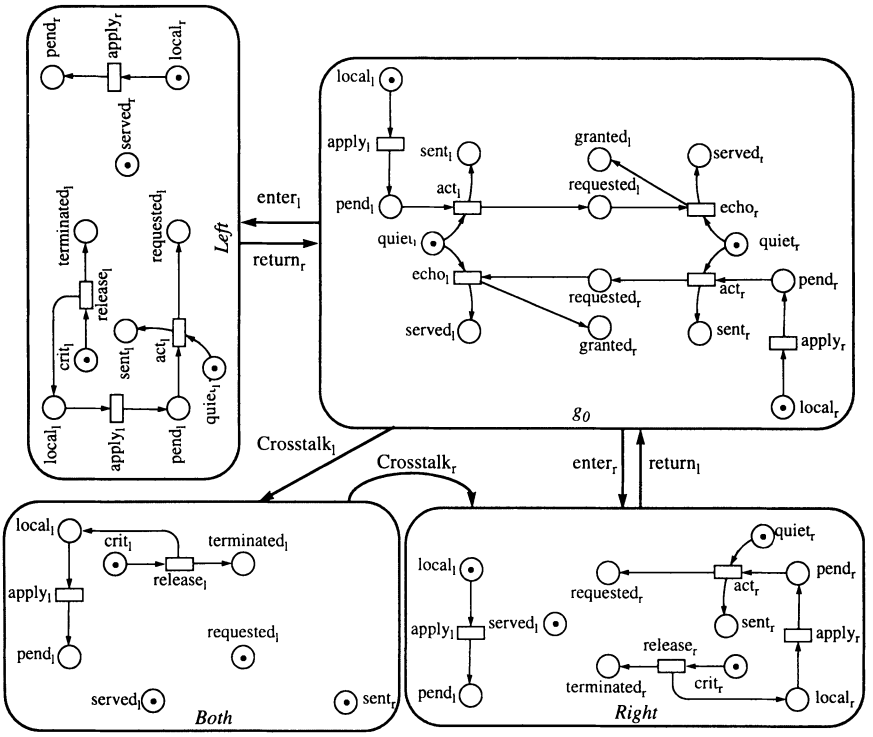

Figure 4 Occurrence net graph of the round based mutual exclusion model.

Theorem 4.2 : *(state preserving)* Let NPG be an occurrence net graph of N,

$$R(N) = \bigcup_{g \in G} R(g)$$

Deadlocks of N is a deadlock of a node for which no transition labelling an output arc is enabled. For a computation point of view, the deadlock search can be done locally to a node. A node that contains a deadlock is called terminal because symbolic sequences in the occurrence net graph can deadly stop in such a node.

Theorem 4.3 : Let NPG = (G, X, g_0) be an occurrence net graph of N, g be a net in G and M a reachable marking of g ($M \in R(g)$). M is a deadlock in N iff:

M is a dead marking in R(g)

$\forall\, t \in T, \forall\, g' \in G: (g,t,g') \in X, \neg M[t>$

If g contains a dead marking of N then g is defined as *terminal.*

Until now, we were concerned by state properties. However, every symbolic sequence in an occurrence net graph deduces a set of firing sequences in the original net. An exemplary firing sequence of them is the one which goes from node initial marking to node initial marking using local firing sequences and transitions labelling output arcs. If we want to preserve every firing sequence, we have to add missing arcs. Indeed, for an arc (g,t,g') and a marking M of g such that t is enabled, M[t> is not necessary represented in g'. That leads us to the definition of a closure of an occurrence net graph. The first condition assures that the firing of any transition is represented in the graph. In other words, every firing sequence corresponds to a symbolic sequence. The second one preserves the fact that every symbolic sequence in the closure deduces a firing sequence set in the original net.

Definition 4.4 : Let NPG = (G, X, g_0) be an occurrence net graph of a net N. close(NPG) is a transition relation including X (X $\subseteq$ close(NPG) $\subseteq$ G x T x G) which fulfills the following properties:

1) $(g,t,g') \in X \Rightarrow (g,t,g') \in close(NPG) \wedge R(min_{\bullet t}(g)[t>, T_{\bullet t}(g)) \subseteq \bigcup_{g'':(g,t,g'') \in close(NPG)} R(g'')$

2) $(g,t,g'') \in close(NPG) \Rightarrow \exists (g,t,g') \in X \wedge M_0(g'') \in R(min_{\bullet t}(g)[t>, T_{\bullet t}(g))$

We call closure of the occurrence net graph NPG the graph (G, close(NPG), g_0).

To preserve the stuttering equivalence between the closure of an occurrence net graph and the net, we impose that every node reachable states have the same label, considered as the label of the node. Such an occurrence net graph is said to be compatible with a labelling function of the net.

Definition 4.5 : Let NPG = (G, X, g_0) be an occurrence net graph of N. NPG is said to be compatible with a labelling function L iff:

$\forall g \in G, \forall m \in R(g), L(m) = L(M_0(g))$

We denote by NPG_L, the transition system defined by (G, close(NPG), L, g_0, *terminal*) where $L(g) = L(M_0(g))$.

Theorem 4.6 : Let NPG be an occurrence net graph of N compatible with a labelling function L. NPG_L and N_L are stuttering equivalent.

5. Occurrence Net Graph Construction

The occurrence net graph construction is based on the classical reachability graph algorithm. A stack collects nodes to be treated and initially contains the initial marking of the original net. The main loop picks a node in the stack (while it is not empty), computes its successors and, if they do not already exist, pushes them. At the contrary to the classical algorithm, a node picked in the stack needs to be completed. This is the goal of ConstructNode that appends new transitions to the node and computes its successor transitions. For each successor transition, the initial marking of the destination node is obtained. If this one already exists, a new arc is created. Otherwise, a new node is generated with the initial marking and transitions already deduced from the previous node. After that, the new node is pushed and the arc is added. Note that the detection of the non-respect of the safeness properties is done during the computation.

```
ConstructNode(g,Ext)
vart : Transition;
   I : TransitionSet:=T\T(g);
   Pg: PlaceSet:=M0∪T(g)•;

Ext:=∅;
while (∃ t∈I: •t ⊆ Pg) do
{ I := I \ t;
  if R•t(g) ≠ ∅ then
  { if (t•∩Pg ≠ ∅) or-else
          ExtChoice(g,t) then
      Ext := Ext ∪ t;
    else
    { T(g) := T(g) ∪t;
      Pg := Pg ∪t•;
    }
  }
}
```

```
ConstructGraph(): Graph
varG : NodeSet;
   X : ArcSet:=∅;
   g0 : Node:=<M0,∅>;
   Ext: TransitionSet;
   g,g': Node;
   S : Stack of Node;

G := {g0};
PushS(g0);
while (g := PopS) do
{ ConstructNode(g, Ext);
  forall t:t∈Ext do
  { if ((min•t(g)\•t)∪T•t(g)•)∩t•≠∅ then
      error("N is not safe")
    if not (∃g'∈G:M0(g')=min•t(g)[t>) then
    { g':= (min•t(g)[t>,T•t(g));
      G := G ∪ {g'};
      PushS(g');
    }
    X := X ∪ {(g,t,g')};
  }
}
return <G,X,g0>;
```

Algorithm 1 Occurrence net and occurrence net graph construction.

To complement a node g, potentially enabled transitions ($\bullet t \subseteq P(g)$) are taken into account. The enabled test is a coverability problem on the transition input places. In case of unsuccess, the transition will never be taken into account (there exists a conflict in $C_{\bullet t}(g)$ and this conflict will remain forever). Otherwise, the transition is defined as internal (new transition in g) or external (new arc in X). If appending this transition to the net causes it not to be an occurrence net ($t\bullet \cap P(g) \neq \emptyset$), then the transition is external. Else, the transition is either internal or external. The choice is made by the function ExtChoice. The resulting occurrence net graph strongly depends on this function. As an example, to obtain a compatible graph, the transitions which potentially can change the truth value of an atomic proposition have to be external. As a heuristic to reduce the graph size, transition is made external if the successor of its minimal state is the initial marking of an existing node.

For each arc (g,t,g'), the closure algorithm identifies nodes that contains the successor states of g by t. These states are defined by the occurrence net ($\min_{\bullet t}(g)[t>, T_{\bullet t}(g)$). The stack contains couples (g',gI) of occurrence nets with the same initial marking. g' is a node of the graph and gI is the occurrence to be included in g' and its descendants. For our implementation, a couple (g', gI) is coded by (g',T(gI)). If gI is a sub-net of g', obviously, the computation is already completed. Else, an external transition t' of g' exists for which both configurations in g' and gI are identical. Cutting up gI with respect to the transition t', the state in ($\min_{t'}^{t'}(gI)$, $T_{t'}^{t'}(gI)$) will be treated in the successor node of g' (g'':(g',t',g'')∈X, noticed that $M_0(g'') = \min_{t'}^{t'}(gI)$), the other states ($\min_{t'}^{\emptyset}(gI)$, $T_{t'}^{\emptyset}(gI)$) will be treated in g'. The first step of the computation is to push (g',$T_{\bullet t}(g)$). As and when the algorithm go through nodes, arcs are created.

```
Close(G,X): ArcSet
var  t        : Transition;
     I        : TransitionSet;
     X'       : ArcSet;
     g,g',g'' : Node;
     S        : Stack of <Node, TransitionSet>;

X':= ∅;
forall (g,t,g') ∈ X do
{ X' = X' ∪ (g,t,g');
  Push_S(g',T_*t(g));
  while ((g',I) := Pop_S) do
  { forall (t',g''):(g',t',g'') ∈ X do
    { if C_*t'(g')⊆ I then
      { X' = X' ∪ (g,t,g'');
        PushS(g'',T_t'^∀(M_0(g'),I));
        I := T_t'^∅(M_0(g'),I);
      }
    }
  }
}
return X';
```

Algorithm 2 Closure algorithm.

6. MODEL CHECKING

The occurrence net graph preserves the reachability set and then any property expressed in term of coverability can be checked in a very efficient way (the complexity is O(|G|x|T|)). On the other hand, deciding if a node is terminal (i.e. contains a deadlock of the original net) is NP-complete. However, K.L. McMillan proposes, in [McM92], a clever algorithm, based on a characterization of the configurations that lead eventually to deadlocks. This algorithm can be directly used in our work. The occurrence net graph preserves also the existence of infinite sequence and this property can be checked in O(|X|).

The key point of the LTL$^{\infty}$ model checker is the use of the closure. Given a set of atomic propositions, we proceed the analysis in five stages:

- Identify a cutup set (transition which may change the truth value of atomic propositions).
- Compute a compatible occurrence net graph.
- Identify terminal nodes (using Mc Millan's algorithm).
- Compute the closure of the graph.
- Check formulae on the stuttering equivalent graph, the closure of a compatible occurrence net graph.

Computing a compatible occurrence net graph can be done in several ways. The simplest one is to compute this graph directly from the original net by forcing the cutup transitions to be external. Another way consists in computing it from an occurrence net graph already done. Briefly, this algorithm makes a cutup on each node. Because compatible nodes are computed locally from original nodes, the node construction algorithm is more efficient (quasi-liveness tests can be omitted). Before the construction, an upper-bound of the node number can be estimated as the number of cutup nets of original nodes.

The CutUpConstructGraph algorithm is almost the same as ConstructGraph. The main difference is in the information pushed on the stack in order to compute local compatible nodes. In the stack information (gc, g, C), gc is the local compatible node to be treated; g its node localization and C the configuration in g which leads to the initial marking of gc. The first step is to complete gc. Notice that in CutUpConstructNode the quasi-liveness tests have been withdrawn. Three kinds of external transitions have to be managed:

- Transition which leads to already computed node: its computation is self-evident.
- Transition of the original node: we push the successor node, the same original node and the new configuration.
- Transition external to the original node: the function Find is used to identify the original node containing the wanted state and its associated configuration.

```
CutUpConstructGraph(G,X,g0,K): Graph
varGc: NodeSet;
   Xc : ArcSet:=∅;
   gc0 : Node:=<M0,∅>;
   Ext,C: TransitionSet;
   g,g',gc,gc': Node;
   S: Stack of <Node,Node,TransitionSet>;

Gc := {gc0};
PushS(gc0,g0,∅);
while ((gc,g,C) := PopS) do
{ CutUpConstructNode(X,g,K,gc,Ext);
  forall t:t∈Ext do
  { if not(∃gc'∈Gc:M0(gc')=min*t(gc)[t>) then
    { gc':= (min*t(gc)[t>,T*t(gc));
      Gc := Gc ∪ {gc'};
      if (t∈T(g)) then
        PushS(gc',g,C ∪ Ct(g));
      else
      { g': (g,t,g')∈X;
        PushS(gc',find(X,g',C\C*t(g)∪t));
      }
    }
    Xc := Xc ∪ {(gc,t,gc')};
  }
}
return <Gc,Xc,gc0>;
```

```
CutUpConstructNode(X,g,K,gc,Ext)
var  t : Transition;
     I : TransitionSet:=T(g);
     Pg: PlaceSet:=M0(gc);

Ext:=∅;
while (∃ t∈I: •t ⊆ Pg) do
{ I := I \ t;
  if (t∈K) then
     Ext := Ext ∪ t;
  else
  { T(gc) := T(gc) ∪t;
    Pg := Pg ∪t•;
  }
}
Ext:=Ext∪{t∈T:∃(g,t,g')∈X ∧ •t⊆Pg};
```

```
Find(X,g,C): <Node, TransitionSet>
var  t : Transition;
     g': Node;

while not (C ⊆ T(g)) do
{ (g',t):(g,t,g')∈X∧C*t(g)∪{t}⊆C;
  C := C\(C*t(g)∪{t});
  g := g';
}
return (g,C);
```

Algorithm 3 Cutup graph construction.

Example : Figure 5 gives the compatible occurrence net graph computed from the original graph (Figure 4) which is already closed. The atomic propositions are «waiting for a critical section access» (places $send_l$ and $send_r$, denoted by s_l and s_r) and «accessing to the critical section» (places $crit_l$ and $crit_r$, denoted by c_l and c_r). The cutup set is defined by the input and the output transitions of these places. The size of this graph was foreseeable and correspond to the cutup of each node.

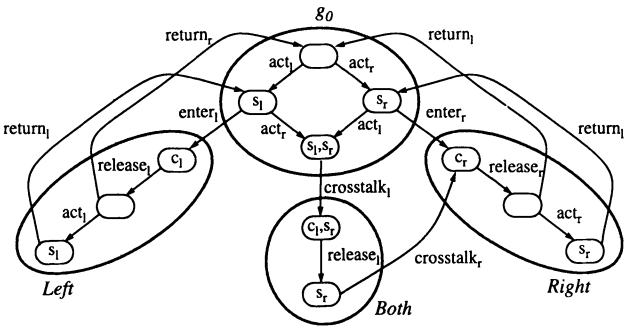

Figure 5 Occurrence net graph compatible with critical and sent.

The fairness property for guaranteed accessibility to the critical section ($\Box(s_l U c_l)$, $\Box(s_r U c_r)$) are easily verified in the compatible occurrence net graph, as well as the priority of the process l over r ($\Box((s_l \wedge s_r) U c_l)$). The safeness of the critical section ($\Box(\neg c_l \vee \neg c_r)$) is obvious on the graph, however, it can be checked on the original graph as a coverability problem.

7. Experimental Results

The efficiency of our approach is evaluated in comparison to the following techniques: the Petri net traversal algorithm based on BDD [Past94], The stubborn set method [Val91] and the construction of a complete finite prefix of an unfolding. For this last technique, we use an improvement of the original method [McM92] due to J. Esparza, S. Römer and W. Vogler [Esp96]. Used tools are Prod [Var92] for the stubborn set method, Pep [GB96] for the unfolding and tools developed in our laboratory for the Petri net traversal algorithm and our approach.

The results for three examples are presented in this paper: the dining philosophers [Past94], the Peterson's mutex algorithm for *n* processes [Cou94] and the distributed database [Jens86]. All examples are scalable, in such a way that the number of system states is exponential with respect to a given parameter. Figure 6 presents a run-time comparison, with a semi-logarithmic scale. Measurements have been done on a Sparc 5 workstation. Table 1 gives the size of the initial Petri net, the number of states in the reachability graph, the number of nodes for the final BDD representation of the state set, the number of states in the reduced reachability graph, the size of the unfolding and the number of nodes of the occurrence net graph.

The unfolding technique is very closed to our method and a comparison seems to be natural. BDD representation and stubborn set reduction are the bases of some efficient LTL model checkers (respectively [Clarke94] and [GW94]). The key points of these approaches are their capabilities to capture the partial order of a system and to give a concise state space representation.

Example				BDD		Stubborn sets		Esparza's unfolding		Occurrence net graph	
	n	size (pl/tr)	nb.of states	nb. of nodes	cpu (sec.)	nb. of states	cpu	size (pl/tr)	cpu	nb. of nodes	cpu
dining philosophers	20	140/100	2×10^{13}	771	357.37	25426	93.30	200/100	0.23	1	0.13
	50	350/250	2×10^{33}	1886	4447.5	-	-	500/250	0.91	1	0.55
	100	700/500	-	-	-	-	-	1000/500	4.71	1	2.08
Peterson's mutex algorithm	2	18/18	50	78	0.65	32	0.14	51/31	0.08	9	0.03
	3	45/57	1065	436	64.26	645	2.18	758/447	0.68	119	0.65
	4	84/132	25636	4041	1487.9	15600	122.1	12132/6804	85.9	2183	31.54
distributed database	2	15/8	7	51	0.19	7	0.06	23/8	0.09	3	0.07
	4	61/32	109	392	8.06	29	0.14	101/32	0.13	44	0.18
	6	139/72	1459	1148	93.24	67	0.32	235/72	0.24	432	2.9
	8	249/128	17497	2848	606.68	121	0.59	425/128	0.49	3264	72.77

Table 1 Experimental results.

For the dining philosopher example, unfolding and occurrence net graph methods give the best results. Both techniques construct almost the original net and then lead to a perfect concise representation of the state space. On the other hand, the stubborn set reduction captures in an efficient way the partial order but the lack of concise representation leads to low performances. The good BDD representation of the state space can not avoid the state explosion.

For the Peterson's mutex algorithm, the results given by the stubborn set method show that the system is highly synchronized (the reduction rate is less than two). Efficiency of the different approaches depends mainly on the state representation. Size of the result given by the BDD based technique is satisfactory. However a memory high peak increases the computation time. For the unfolding method, one can notice that the size of the resulting net becomes large. The more the size of the unfolding net grows during the construction, the more the cost of adding a new transition increases. Indeed, this cost depends on the size of the net and especially on the rate of duplicated elements. This phenomenon can be observed on the run-time comparison diagram. In the occurrence net graph, a node construction only depends on the size of the original net. Therefore, good performances can be obtained.

For the distributed database, the efficiency of the stubborn set and unfolding techniques is remarkable. This is due to the fact that this model is largely asynchronous. At the contrary, the occurrence net graph approach fails. Indeed, our method does not capture the full partial order of the system. As an example, for a system composed by n completly independant subsystems, the occurrence net graph of the system is the cartesian product of the occurrence net graph of each subsystems. In each node the partial order is well captured. However, this order is ignored at the graph level.

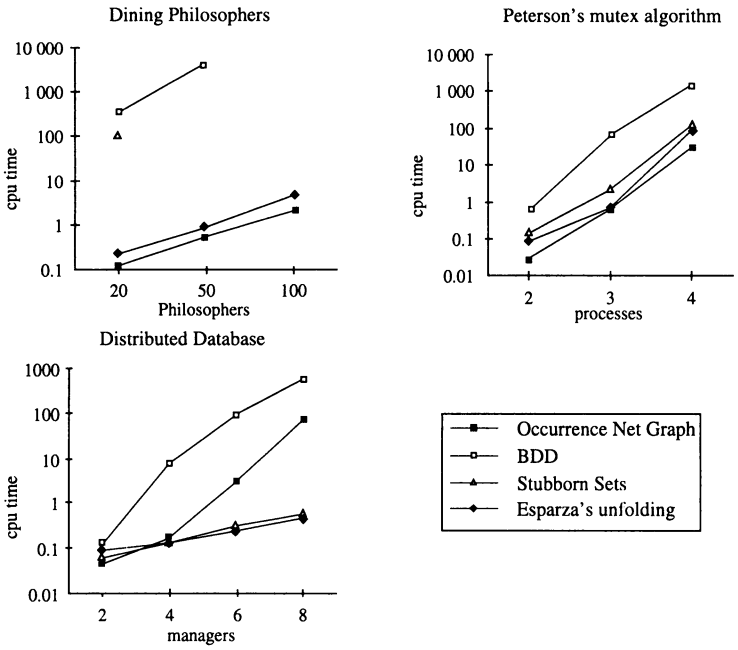

Figure 6 Run-time comparison.

8. CONCLUDING REMARKS

We have proposed a new representation of reachability graph as a graph of particular Petri nets for which a model checker is adapted. The efficiency of the method strongly depends on the size of the graph and on the complexity of node computation. However, the number of nodes is bounded by the number of states. The node construction uses efficient occurrence net algorithms.

Many problems can be stated in term of coverability or deadlock. The occurrence net graph representation provides adequate algorithms to check them on each node and then fits the verification of invariant properties. Based on stuttering equivalence, we have defined a nexttime-less temporal logic model checker.

The primary experimental results have demonstrated that our method is competitive against other ones and very efficient in some cases. The improvement under consideration is to take into account the partial order at the graph level by an approach based on the stubborn set theory. An other objective is to adapt our construction algorithm to verify temporal logic formulae in an on-the-fly way.

9. REFERENCES

[Berth85] Berthelot, G.: **«Checking Properties of Nets Using Transformations»**. Advances in Petri Nets 1985, Rozenberg, G. (ed.), Springer Verlag, LNCS vol 222, pp 19-40, 1986.

[Clarke94] Clarke E., Grumberg O. & Long D.: **«Verification tools for finite-state concurrent systems»**, In: A Decade of Concurrency - Reflections and Perspectives, LNCS vol 803, 1994.

[Cou94] Couvreur J.M. & Paviot-Adet E.: **«New Structural Invariants for Petri Nets Analysis»**. 15th Int. Conference on Application and Theory of Petri Nets, Valette R. (ed), Springer Verlag, LNCS vol 815, pp 199-218, Zaragoza, Spain, June 1994.

[Esp93] Esparza J.: **«Model Checking Using Net Unfoldings»**. Science of Computer Programming, vol 23, pp 151-195, 1994.

[Esp96] Esparza J., Römer S. & Vogler W.: **«An Improvement of McMillan's Unfolding Algorithm»**. Proc. of 2nd Int. Workshop TACAS'96, Springer Verlag, LNCS vol 1055, pp 87-106, Passau, Germany, March 1996.

[GB96] Grahlmann B. & Best E.: **«PEP - More than a Petri Net Tool»**. Proc. of 2nd Int. Workshop TACAS'96, Springer Verlag, LNCS vol 1055, pp 397-401, Passau, Germany, March 1996.

[GW94] Godefroid P. & Wolper P.: **«A Partial Approach to Model Checking»**. Information and Computation, vol 110, No 2, pp 305-326, May 1994.

[Jens86] Jensen K.: **«Coloured Petri Nets»**. In Petri Nets: Central Model and their Properties, Advances in Petri Nets 1986, Part 1, Brauer W., Reisig W. & Rozenberg G. (eds), Springer Verlag, LNCS vol 254, pp 248-299, Bad Honnef, Germany, September 1986.

[KV92] Kaivola R. & Valmari A.: **«The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic»**. Proc. of the 3th Int. Conference on Concurrency Theory, Cleaveland W.R. (ed.), Springer Verlag, LNCS vol 630, pp 207-221, Stony Brook, NY, USA, August 1992.

[Lam83] Lamport L.: **«What Good is Temporal Logic?»**, Proc. of the IFIP 9th World Computer Congress, pp 657-668, 1983.

[McM92] McMillan K.L.: **«Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits»**. Proc. of the 4th Int. conference on Computer-Aided Verification, Springer Verlag, LNCS vol 663, pp 164-175, Montreal, Canada, June 1992.

[Mur89] Murata T.: **«Petri Nets: Properties, Analysis and Applications»**. Proc. of the IEEE, vol 77, No 4, pp 541-580, April 1989.

[Nielsen80] Nielsen M., Plotkin G. & Winskel G.: **«Events Structures and Domains»**. Theoretical computer Science, vol 13, No 1, pp 85-108, 1980.

[Past94] Pastor E., Roig O., Cortadella J. & Badia R.M.: **«Petri Net Analysis using Boolean Manipulation»**. 15th Int. Conference on Application and Theory of Petri Nets, Valette R. (ed), Springer Verlag, LNCS vol 815, pp 416-435, Zaragoza, Spain, June 1994.

[Reisig95] Reisig W.: **«Petri Net Models of Distributed Algorithms»**. In Computer Science Today: Recent Trends and Developments, Jan van Leeuven (ed.), Springer Verlag, LNCS vol 1000, 1995.

[Val91] Valmari A.: **«Stubborn Sets for Reduced State Space Generation»**. Advances Petri Nets 90, Springer verlag, LNCS vol 483, pp 491-515, 1991.

[Var92] Varpaaniemi K. & Rauhamaa M.: **«The Stubborn Set Method in Practice»**. 13th Int. Conference on Application and Theory of Petri Nets, Jensen K. (ed), Springer Verlag, LNCS vol 616, pp 389-393, Sheffield, UK, June 1992.

[Wol89] Wolper P.: **«On the relation of Programs and Computations to Models of Temporal Logic»**. Proc. of Temporal Logic in Specification, Banieqbal B., Barringer H. & Pnueli A. (eds.), Springer Verlag, LNCS vol 398, pp 75-123, 1989.