# 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: to define new concise graph representations for which verification methods are adapted; to reduce graphs while preserving observed properties. We propose a new representation of a reachability graph where nodes are particular Petri nets, occurrence nets, characterizing state space parts. 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 formulas are verified.

## 1. INTRODUCTION

Temporal logic model checking is a useful method for verifying properties of finite concurrent systems. Many algorithms have been developed depending on property representation (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 are adapted verification methods. For instance, we can mention methods based on Binary Decision Diagram [Burch90], symmetries [Chiola91][ES95] and unfoldings [Nielsen80][McM92][Esp93]. 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][Mil89][KV92].

In this paper, we propose a new representation of a reachability graph for safe Petri nets where nodes are particular nets, occurrence nets, characterizing state space parts. This class of net has been already used for unfolding safe Petri nets [Nielsen80][McM92][Esp93]. In opposition to the unfolding, we represent the reachability graph as a graph of occurrence nets, sub-nets of the original one, rather than a single occurrence net where elements are duplicated. Our method takes entirely advantage of efficient algorithms dedicated to invariant properties verification for occurrence nets.

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 next-time-less linear temporal logic. Briefly, given an occurrence net graph, checking a formula is done 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. A safe Petri net modelling a round based mutual exclusion system illustrates the concept introduced in the various sections and can be found in Appendix.

Proofs of theorems and algorithms are technical and use intensively the elementary occurrence net properties. 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 labeling 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).

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

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 $\overset{\infty}{\Diamond}$ «infinitely often in the proper future». In other respects the definitions of our linear logic are standard and we will refer to it as $LTL^{\infty}$.

$LTL^{\infty}$ formulae are built from atomic propositions $f_1, f_2, \ldots$ on states, boolean connectives ($\wedge$, $\neg$) and temporal operators (*U* «until», $\overset{\infty}{\Diamond}$ «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, $\overset{\infty}{\Diamond}$ 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 $\alpha = w_1 \ldots w_i \ldots$ be a sequence from $w_0$ of a transition system H = (S, X, L, $s_0$, *terminal*). For a formula f, $\alpha \models f$ means that f is satisfied by $\alpha$. We have

- $\alpha \models f$ iff $f \in L(w_0)$, 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 $\exists$ $1 \leq i \leq |\alpha|$ such that ($\forall j \leq i$, $\alpha^{(j)} \models f$) and ($\alpha^{(i)} \models g$)
- $\alpha \models \overset{\infty}{\Diamond} f$ 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, $tr(w_1) = L(w_1)$
- if n>1 and $L(w_{n-1}) = L(w_n)$, $tr(\alpha) = tr(w_1 \ldots w_{n-1})$
- if n>1 and $L(w_{n-1}) \neq L(w_n)$, $tr(\alpha) = tr(w_1 \ldots w_{n-1}).L(w_n)$

The trace of an infinite sequence is the limit of the traces of its prefixes $tr(w_1 \ldots w_n)$.

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 nets are considered.
- R(N) is the reachability set 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,
- $(M,V) = (M \cup V^\bullet \cup {}^\bullet V, V, (VxP \cup PxV) \cap F, M)$ is the sub-net of N which represent the firing sequence from M using only transitions in V.

*Example :* Figure *8.1* (in the appendix) shows a safe Petri net modelling a round based mutual exclusion system and Figure *8.2* gives examples of sub-nets. On this safe Petri net, atomic proposition set $AP = \{c_l, c_r, s_l, s_r\}$ 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 properties used 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.

**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

By definition (Definition *3.1* point 3), there is a reflexive partial order on transitions and places. For a set of places and transitions U, $C_U(N)$ contains all the transitions of U and the ones which have a descendant in U. As notes K.L. McMillan in [McM92], «The most important theoretical notion regarding occurrence nets is that of a *configuration*». Configurations represent possible firing sequences of the net. They simply characterize reachable markings and quasi-live transitions. It is based on the reflexive partial order and on the notion of transition conflict.

**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 $g_0$ of the figure *8.2*, $C_{echo_r}(g_0)=\{apply_l,act_l,echo_r\}$ 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 $g_0$.

**Proposition 3.3 :** Let N be an occurrence net. A marking M is reachable iff there exists a configuration S such that $M=(M_0 \cup S\bullet)\backslash\bullet S$. 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 [McM92]. 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 $\bullet C_A(N) \cap A = \emptyset$. If $R_A(N)$ is non empty, the sub-net of N, ($min_A(N)$, $T_A(N)$), where:

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

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

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 $g_0$ in the figure *8.2*, we have $\mathrm{min}_{\bullet\mathrm{enter}_l}(g_0)$ = {$\mathrm{send}_l$,$\mathrm{granted}_l$,$\mathrm{local}_r$} and $T_{\bullet\mathrm{enter}_l}(g_0)$ = {$\mathrm{apply}_r$}.

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 non empty, the sub-net of N, ($\mathrm{min}_K^I(N)$ , $T_K^I(N)$ ), where:

$\mathrm{min}_K^I(N) = (M_0 \cup C_I(N)\bullet)\backslash\bullet C_I(N)$

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

is a quasi-live occurrence net such that $R((\mathrm{min}_K^I(N), T_K^I(N))) = R_K^I(N)$. Moreover, $R(N) = \bigcup_{I;I\subseteq K} R_K^I(N)$ and then the sub-nets ($\mathrm{min}_K^I(N)$ , $T_K^I(N)$ ), for which $R_K^I(N)$ is non 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 non 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 $g_0$ (in figure *8.2*) with respect to K={$\mathrm{act}_l$,$\mathrm{act}_r$} gives:

$\mathrm{min}_K^{\varnothing}(g_0) = M_0(g_0)$ , $T_K^{\varnothing}(g_0) = \{\mathrm{apply}_l, \mathrm{apply}_r\}$

$\mathrm{min}_K^{\mathrm{act}_l}(g_0) = \{\mathrm{sent}_l, \mathrm{requested}_l, \mathrm{quiet}_r, \mathrm{local}_r\}$ , $T_K^{\mathrm{act}_l}(g_0) = \{\mathrm{echo}_r\}$

$\mathrm{min}_K^{\mathrm{act}_r}(g_0) = \{\mathrm{sent}_r, \mathrm{requested}_r, \mathrm{quiet}_l, \mathrm{local}_l\}$ , $T_K^{\mathrm{act}_r}(g_0) = \{\mathrm{echo}_l\}$

$\mathrm{min}_K^K(g_0) = \{\mathrm{requested}_r, \mathrm{requested}_l\}$ , $T_K^K(g_0) = \varnothing$

## 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 - preservation of reachable states, deadlock characterization, and 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, 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 assume the stability of our representation, every firing of a node reachable state 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 $t \in T(g)$ or-else $\bullet t \not\subset P(g)$ or-else $R_{\bullet t}(g) = \emptyset$ then $\forall g' \in G, (g,t,g') \notin X$ |
| | else $\exists! g' \in G, (g,t,g') \in X$ |
| *(stability property)* | $\forall g,g' \in G, t \in T: (g, t, g') \in X, \min_{\bullet t}(g)[t\rangle M_0(g')$ |
| *(reachability property)* | $\forall g \in G$, there exist a path from $g_0$ to g |

*Example :* The Figures *8.1* and *8.2* present a safe Petri net and its occurrence net graph.

**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 labeling an output arc is enabled. For a computation point of view, the deadlock search can be done locally to a node. A node which 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)

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

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. On the other hand, we add missing arcs to assume that every firing sequence in the original net defines a symbolic sequence.

**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 ($X \subseteq \text{close(NPG)} \subseteq G \times T \times G$) which fulfills the following properties:

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

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

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 labeling 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 labeling function L iff:

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

We denote as $NPG_L$, 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 labeling 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 is used to collect 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 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 which appends new transitions to the node and computes its successor transitions. For each successor transitions, 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)
var t  : Transition;
    I  : TransitionSet:=T\T(g);
    Pg: PlaceSet:=M0∪T(g)•;

Ext:=∅;
while (∃ t∈I: •t ⊆ Pg) do
{ I := I \ t;
  if R•a(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
var G : 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 5.1 :** 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 exist 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 could be internal or external. The choice is done by the function ExtChoice. The resulting occurrence net graph strongly depends on this function. As an example, to obtain compatible graph, the transitions which potentially can change the truth value of atomic proposition have to be external. As a heuristic to reduce the graph size, transition with successor of its minimal state being an existing node initial marking is external.

For each arc (g,t,g'), the closure algorithm identifies nodes which 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 include 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 complete. 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'')$\in$X, notice that $M_0(g'') = min_{t'}^{t'}(gI)$), the other states $(min_{t'}^{\varnothing}(gI)\ ,\ T_{t'}^{\varnothing}(gI)\ )$ 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'^t'(M_0(g'),I));
        I := T_t'^∅(M_0(g'),I);
      }
    }
  }
}
return X';
```

**Algorithm 5.2 :** Closure Algorithm

## 6. MODEL CHECKING

The occurrence net graph preserves the reachability set and then any problem posed in term of coverability can be checked in a very efficient way (the complexity is O(|G|x|T|)). On the other hand, checking deadlock in an occurrence net is NP-complete. However, K.L. McMillan proposes, in [McM92], a nice algorithm based on a characterization of the configurations which lead eventually to deadlocks. He obtains, by this way, a very efficient procedure in practice which can be done in a linear space complexity. If we are concerned about linear temporal logic properties, only the existence of infinite sequences can easily be done on the occurrence net graph.

The key point of the $LTL^{\infty}$ model checker is the use of the closure. Given a set of atomic propositions, we proceed with 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 formulas 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 forcing the cutup transitions to be external. Another way consists in computing it from an occurrence net graph already done. Briefly, this algorithm achieves 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 nodes 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 in 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 withdraw. 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.

```
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);
```

```
CutUpConstructGraph(G,X,g0,K): Graph
var Gc: 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>;
```

**Algorithm 6.1 :** Cutup graph construction

*Example :* Figure *8.3* gives the compatible occurrence net graph computed from the original graph (Figure *8.2*) 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.

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. CONCLUDING REMARKS

We have proposed a new representation of reachability graph as a graph of particular Petri nets for which is adapted a model checker. The efficiency of the method strongly depends on the size of the graph and on the complexity of node computation. We assumed that both these constraints are respected. Indeed, the number of nodes is bounded by the state number; 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 is well adapted to invariant properties verification.

The nexttime-less temporal logic is a powerful model to express safeness and liveness properties. Based on stuttering equivalence, we have defined a model checker. At the present time, it is quite difficult to evaluate the efficiency of our method to verify realistic systems. For this purpose, an implementation using Binary Decision Diagram is in progress.

## 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.

[Burch90] Burch J.R., Clarke E.M., McMillan K.L., Dill D.L. & Hwang J.: **«Symbolic Model Checking: $10^{20}$ States and beyond»**. Proc. of the 5th Annual Symposium on Logic in Computer Science, June 1990.

[Chiola91] Chiola G., Dutheillet C., Franceschinis G., Haddad S.: **«On Well Formed Colored Petri Nets and their Symbolic Reachability Graph»**, In: «High Level Petri Nets. Theory and Application», K.Jensen & G. Rozenberg (eds.), Springer-Verlag, pp.373-396, 1991.

[ES95] Emerson E.A. & Sistla A.P.: **«Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach»**. Proc. of the 7th Int. Conference on Computer-Aided Verification, Springer Verlag, LNCS vol 939, pp 309-324, Liege, Belgium, July 1995.

[Esp93] Esparza J.: **«Model Checking using Net Unfoldings»**. Proc. of TAPSOFT'93, Gaudel M.C. & Jouannaud J.P. (eds.), Springer Verlag, LNCS vol 668, pp 613-628, 1993.

[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.

[Mil89] Milner R.: **«Communication and Concurrency»**, Prentice Hall, 1989.

[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.

[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.

[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.


## 8. APPENDIX

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

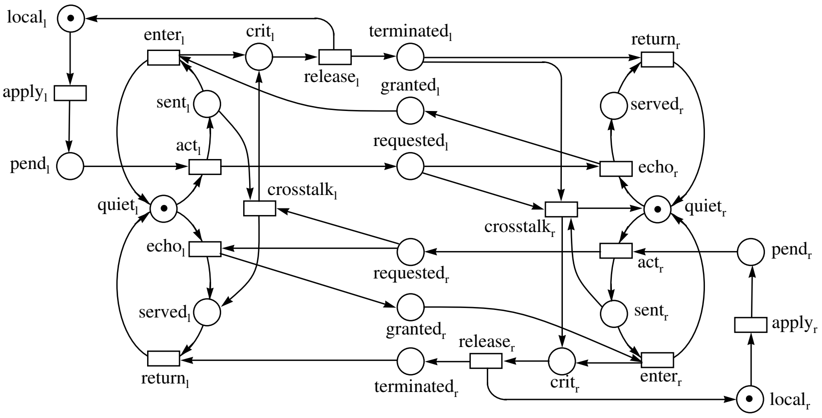


**Fig. 8.1 :** Round Based Mutual Exclusion

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.

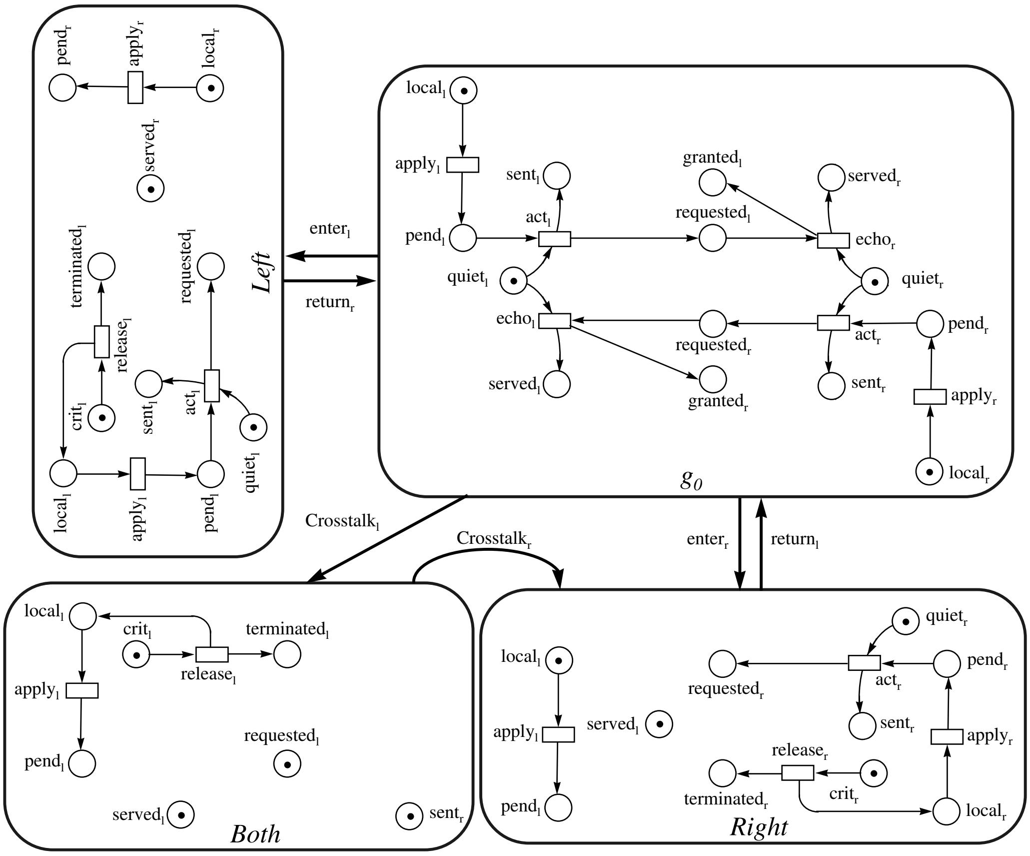

**Fig. 8.2 :** Occurrence net graph of the Round Based Mutual Exclusion Model

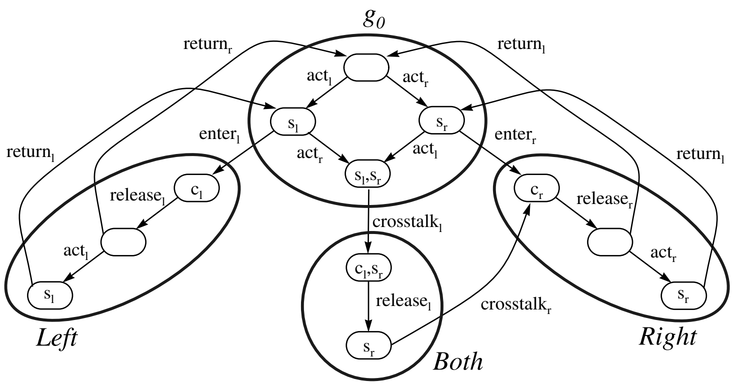

**Fig. 8.3 :** Occurrence net graph compatible with critical and sent