Checking Linear Temporal Formulas on Sequential Recursive Petri Nets

Serge Haddad
LAMSADE - UPRESA 7024, Université Paris IX, Dauphine
Place du Maréchal De Lattre de Tassigny, 75775 Paris cedex 16

Denis Poitrenaud
LIP6 - UMR 7606, Université Paris VI, Jussieu
4, Place Jussieu, 75252 Paris cedex 05

Abstract

Recursive Petri nets (RPNs) have been introduced to model systems with dynamic structure. Whereas this model is a strict extension of Petri nets and context-free grammars (w.r.t. the language criterion), reachability in RPNs remains decidable. However the kind of model checking which is decidable for Petri nets becomes undecidable for RPNs. In this work, we introduce a submodel of RPNs called sequential recursive Petri nets (SRPNs) and we study the model checking of the action-based linear time logic on SRPNs. We prove that it is decidable for all its variants : finite sequences, finite maximal sequences, infinite sequences and divergent sequences. At the end, we analyze language aspects proving that the SRPN languages still strictly include the union of Petri nets and context-free languages and that the family of languages of SRPNs is closed under intersection with regular languages (unlike the one of RPNs).

1. Introduction

In the area of verification theory, a great attention has been recently paid on infinite state systems. In contrast to finite state systems where theoretical and practical developments mainly focus on complexity reduction, an essential topic in infinite state systems is to find a trade-off between expressivity of the models and decidability of verification. As the model checking of temporal logic formula is one of the most general approach for verification, it has been intensively studied in the framework of infinite-state systems.

Context-free grammars and stack automata have led to several works. In [10], it is shown that the model checking of branching time μ-calculus formula for pushdown processes is decidable. When restricting the logic to the linear time logic LTL, one obtains polynomial time algorithms [3].

In [2], model checking for Petri nets has been studied. The branching temporal logic as well as the state-based linear temporal logic are undecidable even for restricted logics. Fortunately, the model checking for action-based linear temporal logic is decidable. The case of infinite sequences may be straightforwardly reduced to the search of repetitive sequences studied in [11] and the case of finite sequences may be similarly reduced to the reachability problem [8]. It seems interesting to combine context-free grammars and Petri nets and to look for decidable properties. Indeed, for two such models - the process rewrite systems [9] and the recursive Petri nets (RPNs) [5] - the reachability problem is decidable. However, for both these two models, the model checking of action-based temporal logic becomes undecidable. It remains undecidable even for restricted models such as those presented in [1]. So for any previously existing model strictly including Petri nets and context-free grammars, the action-based linear time model checking was undecidable.

In this work, we present a submodel of RPNs called sequential recursive Petri nets (SRPNs) and we give some decision procedures including the model checking. Roughly speaking, in RPNs some transitions emulate concurrent procedure calls by initiating a new to-

ken game in the net. The return mechanism is ensured by reachability conditions. A state of a RPN is then a tree of "token games". In a SRPN, a procedure call freezes the current token game and the activity goes on, in the last initiated token game. A state of a SRPN is then a stack of "token games". At first, we illustrate the increase of modelling power of SRPNs w.r.t. the Petri nets one. Indeed, a SRPN can model an infinite in-degree transition system whereas it is not the case with Petri nets (or with process algebras). Moreover, from a practical point of view the use of SRPNs often leads to a great simplification in the design process.

We then focus on the model checking problem for an action-based linear time logic. We handle the case of finite (and maximal) sequences relying on a *product SRPN* construction which emulates the synchronised product of a SRPN and an automaton and then reducing the problem to a reachability problem (which is decidable from [5]). The case of infinite (and divergent) sequences is more tricky and requires to distinguish such sequences w.r.t. the asymptotic behavior of the depth of token games. As the decision procedure is partly based on a reachability decision algorithm for Petri nets, it is not primitive recursive. Nevertheless in a modeling, the places of a SRPN are very often k-bounded with the bound k given *a priori* (e.g. computed by linear algebraic techniques). In such situations, taking as inputs the SRPN and the bound, we can show that our procedure is in PSPACE. We emphasize that even in this case we deal with infinite-state systems.

At last, we study the language family of SRPNs and we show that this family strictly includes the union of Petri nets and context-free languages. Moreover, unlike RPNs, this family is closed under intersection with regular languages. Finally we will discuss about complexity features and give some perspectives to this work. The complete proofs can be found in a research report [6].

2. Sequential Recursive Petri Nets

2.1. Presentation

A sequential recursive Petri net has the same structure as an ordinary one except that the transitions are partitioned into two categories: *elementary transitions* and *abstract transitions*. Moreover a *starting marking* is associated to each abstract transition and an effectively semilinear set of *final markings* is defined. The semantics of such a net may be informally explained as follows. In an ordinary net, a thread plays the token game by firing a transition and updating the current marking (its internal state). In a SRPN there is a stack of threads (denoting the fatherhood relation) where all the threads, except the one on the top of the stack, are suspended. We call this thread, *the current thread*. The step of a SRPN is thus a step of the current thread. If the thread fires an elementary transition, then it updates its current marking using the ordinary firing rule. If the thread fires an abstract transition, it consumes the input tokens of the transition and creates a new thread on the top of the stack (the new current thread) which begins its token game with the starting marking of the transition. If the thread reaches a final marking, it may terminate its token game producing (in the token game of its father) the output tokens of the abstract transition which gave birth to him. In case of a single thread in the stack, one obtains an empty stack.

Definition 2.1 (Sequential Recursive Petri nets) *A* sequential recursive Petri net *is defined by a tuple* $N = \langle P, T, W^-, W^+, \Omega, \Upsilon \rangle$ *where*

- P *is a finite set of places,* T *is a finite set of transitions.*
- *A transition of* T *can be either elementary or abstract. The sets of elementary and abstract transitions are respectively denoted by* T_{el} *and* T_{ab} *(with* $T = T_{el} \uplus T_{ab}$ *where* $\uplus$ *denotes the disjoint union).*
- W^- *and* W^+ *are the pre and post flow functions defined from* $P \times T$ *to* $\mathbb{N}$.
- Ω *is a labeling function which associates to each abstract transition an ordinary marking (i.e. an element of* $\mathbb{N}^P$*) called the starting marking of* t.
- Υ *is an effective representation of semilinear set of final markings.*

A semilinear set of markings is a finite union of linear sets of markings. A linear set L is defined

by a marking m_0 and a finite family of markings $\{m_1, \ldots, m_k\}$ such that $L = \{m \mid \exists \lambda_1, \ldots, \lambda_k \in \mathbb{N}^k,\ m = m_0 + \sum_{i=1,\ldots,k} \lambda_i.m_i\}$. An effective representation is any representation which can be reduced (by an algorithm) to this standard representation. For instance, any system of linear (in)equations on the places marking is an effective representation.

Definition 2.2 (Extended marking) *An* extended marking tr *of a sequential recursive Petri net* $N = \langle P, T, W^-, W^+, \Omega, \Upsilon \rangle$ *is a labeled list* $tr = \langle V, M, E, A \rangle$ *where*

- V *is the set of nodes. If* V *is not empty then* V *contains a* bottom node v_B *and a* top node v_T. *These nodes are identical* iff $|V| = 1$.
- M *is a mapping* $V \to \mathbb{N}^P$,
- $E \subseteq (V \setminus \{v_T\}) \times (V \setminus \{v_B\})$ *is the set of edges with:*
  $\forall v \in V \setminus \{v_B\}$ *there is only one node called* $pred(v)$ *such that* $(pred(v), v) \in E$
  $\forall v \in V \setminus \{v_T\}$ *there is only one node called* $succ(v)$ *such that* $(v, succ(v)) \in E$
- A *is a mapping* $E \to T_{ab}$.

A marked sequential recursive Petri net $\langle N, tr_0 \rangle$ *is a sequential recursive Petri net* N *associated to an initial extended marking* tr_0. *For sake of simplicity (w.l.o.g.), we will require that there is only one node in any initial extending marking.*

When we will deal with different extended markings, we will denote the items of an extending marking tr as a function of the extending marking (e.g. $v_B(tr)$). The empty list is denoted by $\perp$. Any ordinary marking m can be seen as an extended marking, denoted by $\lceil m \rceil$, consisting of a single node. An *elementary step* of a SRPN may be either a firing of a transition or an ending of a token game (called a *cut step* and denoted by τ).

Definition 2.3 *A transition* t *is* enabled *in an extended marking* $tr \neq \perp$ *(denoted by* $tr \xrightarrow{t}$*) if* $\forall p \in P, M(v_T)(p) \geq W^-(p,t)$ *and a cut step is* enabled *(denoted by* $tr \xrightarrow{\tau}$*) if* $M(v_T) \in \Upsilon$

Definition 2.4 *The* firing *of an enabled elementary step* t *from an extended marking* $tr = \langle V, M, E, A \rangle$ *leads to the extended marking* $tr' = \langle V', M', E', A' \rangle$ *(denoted by* $tr \xrightarrow{t} tr'$*) depending on the type of* t.

- $t \in T_{el}$
  - $V' = V$, $E' = E$, $\forall e \in E, A'(e) = A(e)$, $\forall v' \in V \setminus \{v_T\}$, $M'(v') = M(v')$
  - $\forall p \in P, M'(v_T)(p) = M(v_T)(p) - W^-(p,t) + W^+(p,t)$
- $t \in T_{ab}$
  - $V' = V \cup \{v'\}$, $E' = E \cup \{(v_T, v')\}$, $\forall e \in E, A'(e) = A(e)$, $A'((v_T, v')) = t$
  - $\forall v'' \in V \setminus \{v_T\}, M'(v'') = M(v'')$,
  - $\forall p \in P$, $M'(v_T)(p) = M(v_T)(p) - W^-(p,t)$
  - $M'(v') = \Omega(t)$

  where v' *is a fresh identifier absent in* V *;* $v' = v_T(tr')$.
- $t = \tau$
  - $V' = V \setminus \{v_T\}$, $E' = E \cap (V' \times V')$, $\forall e \in E', A'(e) = A(e)$
  - $\forall v' \in V' \setminus \{pred(v_T)\}, M'(v') = M(v')$
  - $\forall p \in P$, $M'(pred(v_T))(p) = M(pred(v_T))(p) + W^+(p, A(pred(v_T), v))$

  Let us notice that if $|V| = 1$ *then the firing of* τ *leads to to empty list* $\perp$.

The depth of an extended marking is defined as $|V|$. For an extended marking tr, its depth is denoted by $depth(tr)$. A firing sequence is defined as usual: a sequence $\sigma = tr_0.t_0.tr_1.t_1.\ldots.t_{n-1}.tr_n$ is a firing sequence (denoted by $tr_0 \xrightarrow{\sigma} tr_n$) iff $tr_i \xrightarrow{t_i} tr_{i+1}$ for $i \in [0, n-1]$. We define the depth of σ as the maximal depth of tr_1, tr_2, ..., tr_n. In the sequel, for sake of simplicity, σ will be often denoted by $\sigma = t_0.t_1.\ldots.t_{n-1}$

2.2. An illustrating example

In order to analyze fault-tolerant systems, the engineer starts from a nominal system and then introduces the faulting behavior as well as the repairing mechanisms. We limit ourselves to an elementary system. The nominal system periodically records some measure of the environment (elementary transition t_{count}).

The number of measures is stored in place p_{count}. The complete system is obtained by adding the bordered part of the figure 1. The behavior of the SRPN can be described as follows. Initially and in the crash state, the extended marking consists in a single node. A token in the place p_{repair} indicates that one is repairing the system while a token in p_{start} indicates that the system is ready. When the abstract transition t_{start} is fired the correct behavior is "played" by the new thread. If this thread dies by a cut step, a crash state is reached. As the place p_{fault} is always marked in the correct system and from the very definition of Υ, the occurrence of a fault is always possible. With additional places and modifying Υ, we could model more complex fault occurrences (e.g. conditioned by software execution).

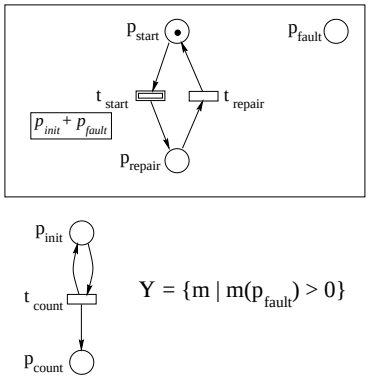

Figure 1. a basic fault-tolerant system

The SRPN switches between states with a single node and states with a bottom and a top node. However, the number of reachable markings in the top node is infinite (the place p_{count} is unbounded). In other words, the crash state can be reached from an infinite number of states which means that *the transition system associated to a SRPN may have some nodes with an infinite in-degree*. This capability is neither shared by Petri nets nor by process algebras. Consequently, such models cannot represent this kind of systems. More generally, any transition system where some node has an infinite in-degree can neither be modelled by Petri nets nor by process algebras. We emphasize that even in the case of finite state transition systems such modelisations are rather difficult and lead to very intricate Petri nets (or process algebras) whereas the same design is quite easy with SRPNs. In [7], the model of recursive Petri nets is illustrated by additional examples and compared to other similar models.

3. Model Checking

The model checking that we investigate applies on action based linear-time formulas represented by Büchi automaton. The usual verification method consists to check the *existence of a sequence* of the system fulfilling the negation of the formula. Depending on the kind of the sequence, different semantics have been defined. We will study the main ones: finite sequences, maximal finite sequences (leading to a deadlock), infinite sequences, divergent sequences (infinite sequences ended by a non observable subsequence).

Definition 3.1 (Büchi automaton) *A Büchi automaton is a tuple $A = \langle \Sigma, Q, \Delta, q_0, F \rangle$ where Σ is an alphabet, Q a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation, $q_0 \in Q$ an initial state and $F \subseteq Q$ a set of accepting states.*

As usual, we denote by $q \xrightarrow{a} q'$ that $(q, a, q') \in \Delta$. Moreover, the extension of $\longrightarrow$ to words over Σ is denoted by $\Longrightarrow$ and is defined as follows:

- $\forall q \in Q, q \stackrel{\lambda}{\Longrightarrow} q$
- $\forall q, q' \in Q, q \stackrel{\omega a}{\Longrightarrow} q' \Leftrightarrow \exists q'', q \stackrel{\omega}{\Longrightarrow} q'' \wedge q'' \xrightarrow{a} q'$

A run r of A on a finite word $\omega = a_1 \dots a_n$ over Σ is a finite sequence $q_0, \dots, q_n$ on Q such that $\forall j \in [1, n], q_{j-1} \xrightarrow{a_j} q_j$. A run r of A on a infinite word $\omega = a_1 \dots a_i \dots$ is an infinite sequence $q_0, \dots, q_i, \dots$ on Q such that $\forall j > 0, q_{j-1} \xrightarrow{a_j} q_j$. We now define how such an automaton recognizes and accepts finite and infinite words.

Definition 3.2 (Recognition and acceptance) *Let $A = \langle \Sigma, Q, \Delta, q_0, F \rangle$ be a Büchi automaton.*

- *Let ω be a finite word over Σ. Then ω is* recognized *by A if there is a run on ω. Moreover, if one of these runs is ended by a state of F then ω is* accepted *by A. $\mathcal{L}(A)$ denotes the set of finite words accepted by A.*

- *Let $\omega = a_1 \dots a_i \dots$ be an infinite word over Σ. Then ω is* recognized *by A if there is a run $r = q_1, \dots q_i, \dots$ on ω. Moreover, if one of these runs fulfills $|\{k \mid q_k \in F\}| = \infty$ then ω is* accepted *by A. $\mathcal{L}^\infty(A)$ denotes the set of infinite words accepted by A.*

The observable behaviors of the SRPN we will consider are defined via a labeling function. A *labeled marked sequential recursive Petri net* is a marked SRPN and a labeling function h defined from the transition set $T \cup \{\tau\}$ to an alphabet Σ plus λ (the empty word). As usual, h is extended to sequences.

Before the study of the model checking problem, we introduce A SRPN representing the synchronised product of a given SRPN and an automaton both labeled on a same alphabet. The product SRPN is constructed from the places of the original one by adding a place set Q which corresponds to the states of the automaton. As usual, the elementary transitions are synchronized with the ones of the automaton using these new places. For each extended arc $q \xRightarrow{a} q'$ (with $a \in \Sigma \cup \{\lambda\}$) of the automaton and for each elementary transition t such that $h(t) = a$, an elementary transition $t.q.q'$, having $W^-(t) + q$ as pre-condition and $W^+(t) + q'$ as post-condition, is added. When an abstract transition is fired a new node appears and, due to the SRPN definition, the token game is limited to this node. Then, we have to predict the state reached by the automaton when the new token game will be ended. The abstract transitions constructed in the product SRPN are denoted $t.q.q'.q''$ where the prefix $t.q.q'$ expresses the same conditions as for the elementary transitions (excepted that t is an abstract transition of the original net). For each state $q'' \in Q$ such an abstract transition is added (the prediction is non deterministic). To ensure that the predicted state is effectively reached when the cut step closing the token game is fired, a set of places $\overline{Q}$ (complementary to Q) is used. The firing of an abstract transition $t.q.q'.q''$ leads to the creation of a new node for which its starting marking has the place $\overline{q''}$ marked. Using these places, the effectively semilinear set of final markings is built in order to ensure that the predicted state is effectively reached. Let us notice that this composition corresponds to a weak synchronization as some transitions of the SRPN can be labeled by λ.

Definition 3.3 (Product SRPN) *Let $A = \langle \Sigma, Q, \Delta, q_0 \rangle$ be an automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN. The product SRPN of A and S is a labeled marked SRPN $\langle \langle N', \lceil m'_0 \rceil \rangle, \Sigma, h' \rangle$ defined by*

- $P' = P \cup Q \cup \overline{Q}$, $m'_0 = m_0 + q_0$
- $T'_{el} = \{t.q.q' \mid (t \in T_{el}) \wedge (q, q' \in Q) \wedge (q \xRightarrow{h(t)} q')\}$
- $\forall t.q.q' \in T'_{el}$,
  - $h'(t.q.q') = h(t)$,
  - $W'^-(t.q.q') = W^-(t) + q$,
    $W'^+(t.q.q') = W^+(t) + q'$
- $T'_{ab} = \{t.q.q'.q'' \mid (t \in T_{ab}) \wedge (q, q', q'' \in Q) \wedge (q \xRightarrow{h(t)} q')\}$
- $\forall t.q.q'.q'' \in T'_{ab}$,
  - $h'(t.q.q'.q'') = h(t)$
  - $W'^-(t.q.q'.q'') = W^-(t) + q$,
    $W'^+(t.q.q'.q'') = W^+(t) + q''$
  - $\Omega'(t.q.q'.q'') = \Omega(t) + q' + \overline{q''}$
- $\Upsilon' = \{m + q + \overline{q'} \mid (m \in \Upsilon) \wedge (q, q' \in Q) \wedge (q \xRightarrow{h(\tau)} q')\}$
- $h'(\tau) = h(\tau)$

The next proposition shows the soundness of the product SRPN construction. This SRPN simulates the synchronized product of the orginal net with the Büchi automaton w.r.t. the language criterion.

We denote by $\mathcal{L}(N, tr_0, Tr_f)$ (where Tr_f is a finite set of extended markings) the set of firing sequences (mapped on $(T \cup \tau)^*$) of N from tr_0 to an extended marking of Tr_f. This set is called the language of N. The language of a labeled marked SRPN $\langle \langle N, tr_0 \rangle, \Sigma, h \rangle$ for a finite extended marking set Tr_f is defined by $h(\mathcal{L}(N, tr_0, Tr_f))$ where h is extended to languages.

For sake of simplicity, we impose that the sets of terminal states are composed by extended marking limited to a single node. One can remark that this condition is not a theoretical restriction.

Proposition 3.4 (SRPN product property) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton, $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN and M_f a set of terminal markings. Let $\langle \langle N', \lceil m'_0 \rceil \rangle, \Sigma, h' \rangle$ be the product SRPN of $\langle \Sigma, Q, \delta, q_0 \rangle$ and S and $M'_f = \{\lceil m + q \rceil \mid \lceil m \rceil \in M_f \wedge q \in F\}$. The following equality holds*

$$h'(\mathcal{L}(N', \lceil m'_0 \rceil, M'_f)) = h(\mathcal{L}(N, \lceil m_0 \rceil, M_f)) \cap \mathcal{L}(A)$$

Sketch of proof:
The main part of the proof follows from the construction presented below. The critical point is that although the product SRPN allows "bad" sequences (i.e. not recognized by the automaton), such ones cannot lead to a terminal extended marking of the product SRPN. ♢

We now adapt the product construction to reduce the model-checking problem of finite sequences to a reachability problem for the product SRPN which is known to be decidable [5].

Theorem 3.5 (Acceptance of finite sequences) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN. The existence of a finite firing sequence σ of S such that $h(\sigma)$ is accepted by A is decidable.*

Sketch of proof:
Let $\langle \langle N', \lceil m_0' \rceil \rangle, \Sigma, h' \rangle$ be the product SRPN of A and S. We construct a new SRPN $\langle N'', \lceil m_0'' \rceil \rangle$ in the following way:

- $N'' = N'$ except for $\Upsilon'' = \Upsilon' \cup \{m \mid \exists q \in F, m \geq q\}$
- $m_0' = m_0''$

It can be shown that the existence of a finite firing sequence σ of S such that $h(\sigma)$ is accepted by A is equivalent to the reachability of $\perp$ by $\langle N'', \lceil m_0'' \rceil \rangle$. The critical point is that considering sequences of N'' reaching $\perp$, then the shortest ones correspond to sequences σ of the original net such that $h(\sigma)$ is accepted by A. ♢

Maximal finite sequences are handled similarly with a more complex construction. In this new product, a pair of places is added to allow the prediction of deadlocks when creating a new node in the stack and the semi-linear set of terminal markings is adapted to detect deadlocks.

Theorem 3.6 (Acceptance of maximal sequences) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN. The existence of a finite firing sequence σ of S such that σ leads to a deadlock of N and $h(\sigma)$ is accepted by A is decidable.*

For the infinite case, the technique based on the SRPN product is not sufficient to obtain a decision procedure. We have developed an original proof technique based on the analysis of the sequences depending on the asymptotic behavior of the depth of the visited extended markings.

We are looking for an infinite firing sequence of the SRPN accepted by a Büchi automaton. We will perform two independent searches depending on a characteristic of the sequence: the asymptotic behavior of the depth of the sequence.

Let $\sigma = \lceil m_0 \rceil \xrightarrow{t_1} tr_1 \xrightarrow{t_2} \dots tr_{i-1} \xrightarrow{t_i} tr_i \dots$ be an infinite sequence, we define $dinf(\sigma) = \liminf_{i \to \infty} depth(tr_i)$ (defined by $\lim_{i \to \infty} inf_{j \geq i} \{depth(tr_j)\}$). $dinf(\sigma)$ always exists but it can be either finite or infinite.

In case of a finite value, there exists a strictly increasing sequence of indexes $i_1, \dots, i_k, \dots$ such that:

- beyond i_1 the set of indexes $\{i_1, i_2, \dots, i_k, \dots\}$ is exactly the indexes for which the depth of the visited extended markings is equal to $dinf(\sigma)$
  $(\forall i \geq i_1, depth(tr_i) = dinf(\sigma) \Leftrightarrow$
  $i \in \{i_1, i_2, \dots, i_k, \dots\})$
- beyond i_1 the depth of the visited extended markings will be greater or equal than $dinf(\sigma)$
  $(\forall i \geq i_1, depth(tr_i) \geq dinf(\sigma))$
- i_1 is the first index from which the depth of the visited extended markings will be no more less than $dinf(\sigma)$
  $(\forall i < i_1, \exists j \geq i, depth(tr_j) < dinf(\sigma))$

So σ will be decomposed as $\lceil m_0 \rceil \xrightarrow{\sigma_0} tr_{i_1} \xrightarrow{\sigma_1} \dots tr_{i_k} \xrightarrow{\sigma_k} tr_{i_{k+1}} \dots$ where σ_0 ends with the firing of an abstract transition leading to an extended marking of depth $dinf(\sigma)$ (with the creation of a new node) and σ_k is either a firing of an elementary transition in this node or a sequence beginning by the firing of an abstract transition in this node and ended by a corresponding cut step.

In case of an infinite value, there exists a strictly increasing sequence of indexes $i_1, \dots, i_k, \dots$ such that:

- k is the depth of the extended marking tr_{i_k}
  $(\forall k, depth(tr_{i_k}) = k)$

- beyond i_k the depth of the visited extended markings will be greater or equal than k
  $(\forall i \geq i_k, depth(tr_i) \geq k)$
- i_k is the first index from which the depth of visited extended markings will be no more less than k
  $(\forall i < i_k, \exists j \geq i, depth(tr_j) < k)$

So σ will be decomposed as:
$\lceil m_0 \rceil = tr_{i_1} \xrightarrow{\sigma_1} tr_{i_2} \xrightarrow{\sigma_2} \dots tr_{i_k} \xrightarrow{\sigma_k} tr_{i_{k+1}} \dots$ where σ_k begins by a firing in an extended marking of depth k, ends with the firing of an abstract transition leading to an extended marking of depth $k+1$ and such that all the extended markings visited by σ_k have a depth greater or equal than k.

So the first step of the proof consists in developping a procedure to check the existence of some finite firing subsequences beginning and ending in the same node of two extended markings and corresponding to paths of the Büchi automaton. Indeed, we need another procedure which restricts the sequences to those which visit an accepting state of the automaton. In either case, these procedures are very similar to the model checking for finite sequences.

We are now in position to explain the two main procedures. Looking for a sequence σ with $dinf(\sigma)$ finite, we first compute the couples of starting markings and automaton states reachable by a firing sequence. We build an ordinary Petri net representing an abstract view of sequences of the SRPN (recognized by the automaton) where the successive extended markings visited by the sequence are infinitely often reduced to a single node. Then, for each couple as initial marking of this Petri net, we look for an infinite sequence visiting a subset of transitions infinitely often (this can be done by the algorithm of [11]).

Looking for a sequence σ with $dinf(\sigma)$ infinite, we build a graph where the nodes are the computed couples of the first procedure and an edge denotes that one node has been reached from the other one by a sequence increasing by one the depth of the visited extended markings and such that the intermediate subsequences never decrease the depth below its initial value. The edges are partitioned depending on the visit by the sequence of an accepting state of the Büchi automaton. Then the existence of an accepting infinite sequence is equivalent to the existence of some kind of strongly connected component.

Theorem 3.7 (Acceptance of infinite sequences) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN. The existence of an infinite sequence σ of $\langle N, \lceil m_0 \rceil \rangle$ such that $h(\sigma)$ is an infinite word accepted by A is decidable.*

Divergent sequences are handled similarly.

Theorem 3.8 (Acceptance of divergent sequences) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled SRPN. The existence of an infinite sequence σ of $\langle N, \lceil m_0 \rceil \rangle$ such that $h(\sigma)$ is a finite word accepted by A is decidable.*

All our decision procedures use the decidability of reachability in RPNs (based on reachability in Petri nets). Thus none of them are primitive recursive. However it must be emphasized that very often unbounded Petri nets correspond to systems with dynamic structure. Modelling such systems with SRPNs leads to infinite states SRPNs with **bounded** places. Moreover the bound may be computed by structural analysis. In such cases, complexity of our decision procedure is reduced as stated by the next theorem.

Theorem 3.9 (Complexity of model-checking) *Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a Büchi automaton and $S = \langle \langle N, \lceil m_0 \rceil \rangle, \Sigma, h \rangle$ a labeled k-bounded SRPN. The problem of existence of a finite (maximal finite, infinite, divergent) sequence σ of $\langle N, \lceil m_0 \rceil \rangle$ such that $h(\sigma)$ is a word accepted by A is PSPACE-complete w.r.t. the size of A and S.*

4. Language Properties

In order to discuss about expressivity of models, different criteria may be applied such like generated languages, behavioural equivalences,... In this section, we focus on the properties of the languages generated by SRPNs and we compare it with the standard hierarchy of languages.

Theorem 4.1 (SRPN closure) *The family of SRPN languages is closed under intersection with regular languages.*

Sketch of proof:
Follows straightforwardly from proposition 3.4 ♢

Theorem 4.2 (Strict inclusion) *SRPN languages strictly include the union of context-free and Petri net languages*

Proof:
It is obvious that any PN is a SRPN. Moreover, in [5], it is demonstrated that any context-free language can be simulated by a RPN. We can remark that the proposed construction of the RPN corresponding to a context-free language leads to a SRPN (i.e. the initial extended marking is limited to a single node and all the reachable states are stacks and only the top node is active). In the same paper, it is shown that RPN languages strictly include the union of context-free and Petri net languages. The proof of this result exhibits a RPN for which its language is neither PN nor context-free language. We can remark that this RPN behaves as a SRPN. Then, we can conclude that the language family of SRPN strictly includes the union of the context-free and PN languages. ♢

Moreover, in [4], it is demonstrated that the RPN languages are not closed under intersection with regular ones. Then the theorem 4.1 leads to the next one.

Corollary 4.3 (SRPN versus RPN) *The family of SRPN languages is strictly included in the family of RPN languages.*

5. Conclusion

In this work, we have introduced sequential recursive Petri nets and studied their theoretical features. From a modeling point of view, an important characteristic of SRPNs is their capability to generate infinite in-degree transition systems. Such a feature makes possible to model dynamic systems which can be handled neither by process algebra nor by Petri nets.

In the second part of the paper, we have focused on the model checking for an action-based linear time logic and obtained different decision procedures depending on the semantics of the logic. These procedures are not primitive recursive in the general case but restricting the SRPNs to bounded ones (with an *a priori* known bound), the model checking problem is shown to be PSPACE-complete.

At last, we have studied the language family of SRPNs and proved that this family strictly includes the union of Petri nets and context-free languages. Moreover, unlike RPNs, this family is closed under intersection with regular languages.

We now plan to study how we can extend SRPNs preserving the model checking decidability.

References

[1] A. Bouajjani and P. Habermehl. Constraint properties, semi-linear systems, and Petri nets. In *Proc. of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[2] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[3] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. of INFINITY'97*, 1997.

[4] S. Haddad and D. Poitrenaud. Decidability and undecidability results for recursive Petri nets. Technical Report 019, LIP6, Paris VI University, Paris, France, Sept. 1999.

[5] S. Haddad and D. Poitrenaud. Theoretical aspects of recursive Petri nets. In *Proc. 20th Int. Conf. on Applications and Theory of Petri nets*, volume 1639 of *Lecture Notes in Computer Science*, pages 228–247, Williamsburg, VA, USA, June 1999. Springer Verlag.

[6] S. Haddad and D. Poitrenaud. A model checking decision procedure for sequential recursive petri nets. Technical Report 024, LIP6, Paris VI University, Paris, France, Sept. 2000. http://www.lip6.fr/reports/lip6.2000.024.html.

[7] S. Haddad and D. Poitrenaud. Modelling and analyzing systems with recursive Petri nets. In *Proc. of the Workshop on Discrete Event Systems - Analysis and Control*, pages 449–458, Gand, Belgique, Aug. 2000. Kluwer Academics Publishers.

[8] E. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. 13th Annual Symposium on Theory of Computing*, pages 238–246, 1981.

[9] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1997.

[10] I. Walukiewicz. Pushdown processes: Games and model checking. In *Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer Verlag, 1996.

[11] H.-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96:119–137, 1992.