MODELLING AND ANALYZING SYSTEMS WITH RECURSIVE PETRI NETS

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

Denis Poitrenaud
LIP6 - UMR 7606, Université Paris VI, Jussieu
4, Place Jussieu, 75252 Paris cedex 05
Denis.Poitrenaud@lip6.fr

Keywords: Recursive Petri nets, Modellization, analysis, expressive power

Abstract Recursive Petri nets (RPNs) have been introduced to model systems with dynamic structure. In a previous work, we have shown that this model is a strict extension of Petri nets, whereas reachability in RPNs remains decidable. Here, we focus on its modelling features and on some additional theoretical aspects. Three different kinds of discrete event systems are modellized by RPNs in order to give an insight of their capabilities to express various mechanisms. Decision procedures for new properties like boundedness and finiteness are presented and recursiveness of languages of RPNs is proved. At last, we compare RPNs with two other models combining Petri nets and context-free grammars features showing that these models can be simulated by RPNs.

Introduction

In the area of verification theory, a great attention has been paid on infinite state systems where an essential topic is to find a compromise between expressivity of the models and decidability of property verification. Among such models, Petri nets present interesting characteristics. On the one hand, Petri nets are now in widespread use for many different practical purposes due to their great modelling capabilities [Jensen, 1987]. On the other hand, it has been shown that the reachability problem [Mayr, 1981] is decidable.

Recently a new extension of Petri nets, recursive Petri nets (RPNs), have been proposed with the aim to combine Petri net and context-free grammar behaviours [Haddad and Poitrenaud, 1999b]. Roughly speaking, in recursive Petri nets some transitions emulate concurrent procedure calls by initiating a new token game in the net. The return mechanism is ensured by reachability conditions. In [Haddad and Poitrenaud, 1999b], we have shown how to decide the reachability problem for RPNs and we have studied the expressive power of RPNs proving that RPNs strictly include the union of Petri nets and context-free grammars w.r.t. the generated languages.

From a modelling point of view, RPNs have been successfully used for specifying plans of agents in a multi-agent system [Seghrouchni and Haddad, 1996]. We complement this work with the modelling of three usual mechanisms of discrete event systems : goal-oriented programming, fault occurrences and interruptions..

We also define new decision procedures for important problems: boundedness, finiteness and recursivity of languages. Finally, we compare the model of RPNs with two other models combining Petri nets and context-free grammars features: net systems introduced by A. Kiehn [Kiehn, 1989] and process algebra nets (PANs) [Mayr, 1997]. We show that RPNs include net systems and PANs. Complete proofs for the main propositions are given in the technical report [Haddad and Poitrenaud, 1999a]

1. RECURSIVE PETRI NETS

A RPN 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 a semi-linear 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 RPN there is a dynamical tree of threads (denoting the fatherhood relation) where each thread plays its own token game. The step of a RPN is thus a step of one of its threads. 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 generates a new child which begins its token game with the starting marking of the transition. If the thread reaches a final marking, it may terminate aborting its whole descent of threads and producing (in the token game of its father) the output tokens of the abstract transition which gave birth to him. In case of the root thread, one obtains an empty tree.

Definition 1 (Recursive Petri nets) *A* 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 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 effectively semilinear set of final markings (any usual syntax can be accepted for its specification).*

Definition 2 (Extended marking) *An* extended marking tr *of a recursive Petri net* $N = \langle P, T, W^-, W^+, \Omega, \Upsilon \rangle$ *is a labeled tree* $tr = \langle V, M, E, A \rangle$ *where*

- V *is the set of vertices,* M *is a mapping* $V \to \mathbb{N}^P$,
- $E \subseteq V \times V$ *is the set of edges and* A *is a mapping* $E \to T_{ab}$.

A marked recursive Petri net $\langle N, tr_0 \rangle$ *is a recursive Petri net* N *associated to an initial extended marking* tr_0.

We denote by $v_0(tr)$ the root node of the extended marking tr. The empty tree 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. For a vertex v of an extended marking, we denote by $pred(v)$ its (unique) predecessor in the tree (defined only if v is different from the root) and by $Succ(v)$ the set of its direct and indirect successors including v ($\forall v \in V, Succ(v) = \{v' \in V \mid (v, v') \in E^*\}$ where E^* denotes the reflexive and transitive closure of E). An *elementary step* of a RPN may be either a firing of a transition or a closing of a subtree (called a *cut step* and denoted by τ).

Definition 3 *A transition* t *is* enabled *in a vertex* v *of an extended marking* tr *(denoted by* $tr \xrightarrow{t,v}$*) if* $\forall p \in P, M(v)(p) \geq W^-(p, t)$ *and a cut step is* enabled *in* v *(denoted by* $tr \xrightarrow{\tau,v}$*) if* $M(v) \in \Upsilon$

Definition 4 *The* firing *of an enabled elementary step t from a vertex v of an extended marking $tr = \langle V, M, E, A\rangle$ leads to the extended marking $tr' = \langle V', M', E', A'\rangle$ (denoted by $tr \xrightarrow{t,v} tr'$) depending on the type of t.*

- $t \in T_{el}$
  - $V' = V$, $E' = E$, $\forall e \in E, A'(e) = A(e)$, $\forall v' \in V \setminus \{v\}$, $M'(v') = M(v')$
  - $\forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t) + W^+(p,t)$
- $t \in T_{ab}$, *(v' is a fresh identifier absent in V)*
  - $V' = V \cup \{v'\}$, $E' = E \cup \{(v,v')\}$, $\forall e \in E, A'(e) = A(e)$, $A'((v,v')) = t$
  - $\forall v'' \in V \setminus \{v\}, M'(v'') = M(v''), \forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t)$
  - $M'(v') = \Omega(t)$
- $t = \tau$
  - $V' = V \setminus Succ(v)$, $E' = E \cap (V' \times V')$, $\forall e \in E', A'(e) = A(e)$
  - $\forall v' \in V' \setminus \{pred(v)\}, M'(v') = M(v')$
  - $\forall p \in P, M'(pred(v))(p) = M(pred(v))(p) + W^+(p, A(pred(v), v))$

  Let us notice that if v is the root of the tree then the firing of τ leads to to empty tree $\perp$.

At first sight, associating the same net to all the abstract transitions may seem restrictive and artificial from a modelling point of view. Nevertheless, it easy to simulate with RPNs the activation of different nets depending on the abstract transitions. Using only one net greatly simplify notations and proofs.

Natural condition for the termination of a thread are almost expressible by an effectively semilinear set of markings. For instance, one can express deadlock of a net, transition enabling, submarking reachability, ... As the effectiveness of representation is preserved under union, intersection and complementation, we will not fix some particular representation of the semilinear sets.

2. MODELLING

The three behaviors that we model with RPNs are difficult or even impossible to specify with typical models such like Petri nets or process algebra.

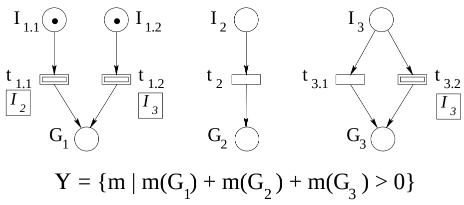

Figure 1 a concurrent goal oriented program

Modelling of goal-oriented programs

With the development of artificial intelligence, new programming paradigms have been introduced with associated languages (e.g. Prolog). The execution of such a program consists in successive applications of rules until some predicate is satisfied. This kind of programming is qualified as goal-oriented. On the other hand, parallel architectures have led to concurrent programming. Goal-oriented and concurrent programming have been merged in such a way that several processes can be executed concurrently in order to satisfy a same goal. As soon as the goal is satisfied by one process, all of them terminate.

The RPN shown in figure 1 models a concurrent goal-oriented program. We represent an abstract transition by a double border rectangle and its initial marking is indicated in a frame. The goal of the program is achieved when the RPN reaches the extended marking $\perp$ from the initial marking consisting in a single node labeled by $I_{1.1} + I_{1.2}$. From this initial state, both abstract transitions $t_{1.1}$ and $t_{1.2}$ are enabled and their firings lead to the creation of two independent threads. As soon as one of these threads is completed, the place G_1 is marked at the root level and then a cut step is firable at this level (see the definition of Υ) and leads to $\perp$. The first thread executes a simple sequential program represented by the elementary transition t_2. The second one chooses either to execute also a simple program (the elementary transition $t_{3.1}$) or to make a recursive call (the abstract transition $t_{3.2}$).

A firing sequence of this RPN is presented in the figure 2. The thread in which the following step is fired is represented in black. One can notice that each firing of abstract transition leads to the creation of a new node in the tree whereas the firing of the last cut step prunes the complete tree.

Let us remark that the first thread may achieve its program while the second one is at any level of recursion. In other words, the state $\perp$ can be reached from an infinite number of states. More generally the transition system associated to a RPN may have some nodes with an infinite in-degree. This capability is not shared by Petri nets or process

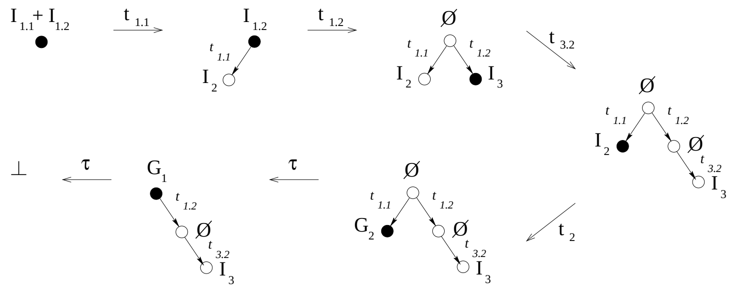

Figure 2 a firing sequence

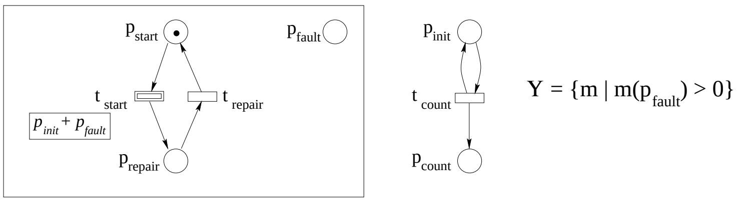

Figure 3 a basic fault-tolerant system

algebras. Consequently, such models cannot directly represent this kind of systems.

Modelling of faults

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_r). The number of measures is stored in place p_{count}. The complete system is obtained by adding the left part of the figure 3. The behavior of the RPN can be described as follows. Initially and in all the crash states, 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).

The RPN switches between states with a single vertex and states with a root and a leaf. However, the number of reachable markings in the leaf

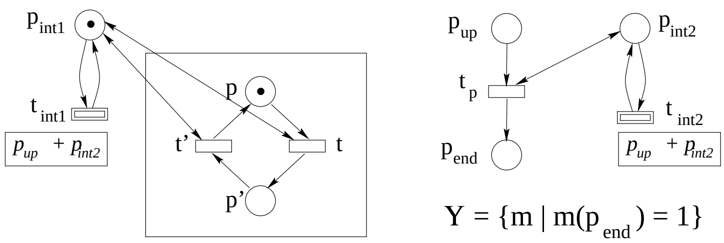

Figure 4 an interruption mechanism

is infinite (the place p_{count} is unbounded). Once again, the associated transition system has a node with an infinite in-degree.

Modelling of interruptions
Let us suppose that we have specify a one level execution system and that we want to add an interruption mechanism. In the RPN of the figure 4, the cycle p, t, p', t' represents this first level. The place p_{int1} controls this execution. When the abstract transition t_{int1} is fired this execution is interrupt and a second level modellized by a token in p_{up} and p_{int2} is activated. The same construction applies again on this component net making possible a recursive interruption process. We should have bound the number of interruption levels with additional places. In comparison the same modelling with Petri net is rather difficult as it requires to keep each context of suspended process.

3. ANALYSIS

As stated in the introduction, the Petri net model appears as a limit model for the decidability of properties for infinite systems. Indeed, slight extensions give it the Turing machine expressive power. The surprising fact with RPN model is that most of the properties decidable in PN remains decidable for it while its expressive power is much larger than the PN one.

Decidability results
The reachability problem consists to determine if a given state is reachable from another one. This problem has been demonstrated to be decidable for PN (see [Mayr, 1981]).

Proposition 5 ([Haddad and Poitrenaud, 1999b]) *The reachability problem is decidable for RPNs.*

The decision procedure presented in [Haddad and Poitrenaud, 1999b] reduces the problem to some applications of the decision procedure for the ordinary Petri nets.

The boundedness property ensures that there is a bound for any place of any reachable extended marking and the finiteness property states that the number of reachable extended markings is finite. In Petri nets, these two properties are equivalent and decidable. In RPNs, the equivalence does not hold but decidability remains for both properties. Another important property is the capability to decide if a given word belongs to the language generated by the system. Such a language is said to be recursive and this characteristic is a key point for the verification of safety properties. We demonstrate that this problem is also decidable for RPNs.

Proposition 6 ([Haddad and Poitrenaud, 1999a]) *The problems of boundedness and finiteness are decidable for RPNs and the language of a labeled RPN is recursive.*

The decision procedures for boundedness and finiteness consists to some applications of the reachability procedure for the ordinary Petri nets. Unlike the situation in Petri nets, the recursivity of the languages can not be proved using the reachability procedure for RPNs.

Expressiveness results

It has been demonstrated in [Haddad and Poitrenaud, 1999b] that RPNs combine features of Petri nets and context-free grammars. It is interesting to compare RPNs with similar models.

In her thesis, A. Kiehn has introduced a model called net systems [Kiehn, 1989]. Net systems are a set of Petri nets with special transitions denoted *caller* transitions which start a new Petri net. A *call* to a Petri net may return if this net reach a final marking. All the nets are required to be safe and the constraints associated to the final marking ensure that a net may not return if it has engaged calls. It is straightforward to simulate a net system by a RPN. Moreover as the languages of Petri nets are not included in the languages of net systems, the family of net system languages is strictly included in the family of RPN languages.

Process Algebra Nets (PANs), introduced by R. Mayr [Mayr, 1997], are a model of process algebra having the sequential composition as well as the parallel one. The left term of any rule of a PAN may use only the parallel composition of variables whereas the right side is a general term. This model includes Petri nets and context-free grammars. We demonstrate that RPNs also include PANs.

Proposition 7 ([Haddad and Poitrenaud, 1999a])

- *The union of context-free and Petri net languages is strictly included in the family of RPN languages,*
- *the family of net system languages is strictly included in the family of RPN languages,*
- *the family of PAN languages is included in the family of RPN languages.*

Whereas we do not know whether the inclusion of the PAN languages by the RPN ones is strict, we emphasize that the main difference between RPNs and the two other models is the ability to prune subtrees from the extended marking. This mechanism is indispensable for the modelling of plans in multi-agents systems [Seghrouchni and Haddad, 1996].

This inclusion has an important consequence. Indeed, in [Bouajjani and Habermehl, 1996] it has been demonstrated that a PA-process (a much less expressive model than PAN) and a finite automaton together can simulate a 2-counter machine. We can conclude that the model checking of linear time temporal logic on action is undecidable for RPNs whereas this problem is decidable for Petri nets [Esparza, 1997].

However, one can notice that PANs as well as Process Rewrite Systems (a more expressive model) can not represent a transition system with an infinite in-degree.

4. CONCLUSION

In this work, we have deepened the analysis of recursive Petri nets. Their modelling capabilities have been illustrated on various mechanisms used in discrete event systems. Moreover some of them cannot be modellized neither by Petri nets nor by process algebra. We have also studied theoretical features of recursive Petri nets which complement the ones studied in [Haddad and Poitrenaud, 1999b] about reachability and expressivity. We have shown how to decide boundedness, finiteness of a RPN and we have proved that the languages of RPNs are recursive. At last, we have shown that RPNs include some previous models combining Petri nets and context-free grammars for which the reachability remains decidable. As a consequence, the general model checking for recursive Petri nets becomes undecidable even for a restricted temporal logic.

We plan to extend our studies in two different ways. On the one hand we want to add new features for recursive Petri nets and examine whether the main properties of RPNs remain decidable. We are interested to introduce some context when a thread is initiated (e.g. the starting marking could depend from the depth in the tree). On the other hand, we are looking for an intermediate model between RPN and PN for which model checking remains decidable.

References

[Bouajjani and Habermehl, 1996] Bouajjani, A. and Habermehl, P. (1996). Constrained properties, semilinear systems and Petri nets. In Montanari, U. and Sassone, V., editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag.

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

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

[Haddad and Poitrenaud, 1999b] Haddad, S. and Poitrenaud, D. (1999b). 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. Springer Verlag.

[Jensen, 1987] Jensen, K. (1987). Coloured Petri nets. In Brawer, W., Reisig, W., and Rozenberg, G., editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 248–299. Springer Verlag, Bad Honnef, West Germany.

[Kiehn, 1989] Kiehn, A. (1989). Petri nets systems and their closure properties. In *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 306–328. Springer Verlag.

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

[Mayr, 1997] Mayr, R. (1997). Combining Petri nets and PA-processes. In *Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *Lecture Notes in Computer Science*, pages 547–561, Sendai, Japan. Springer Verlag.

[Seghrouchni and Haddad, 1996] Seghrouchni, A. E. F. and Haddad, S. (1996). A recursive model for distributed planning. In *Second International Conference on Multi-Agent Systems*, Kyoto, Japon.