Decidability and Undecidability Results for Recursive Petri Nets

Serge Haddad[1] and Denis Poitrenaud[2]

[1] LAMSADE - UPRESA 7024, Université Paris IX, Dauphine
Place du Maréchal De Lattre de Tassigny, 75775 Paris cedex 16
[2] 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, reachability in RPNs remains decidable. Here we focus on three complementary theoretical aspects. At first, we develop decision procedures for new properties like boundedness and finiteness and we show that languages of RPNs are recursive. Then we study the expressiveness of RPNs proving that any recursively enumerable language may be obtained as the image by an homomorphism of the intersection of a regular language and a RPN language. Starting from this property, we deduce undecidability results including undecidablity for the kind of model checking which is decidable for Petri nets. 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.

1 Introduction

Recently recursive Petri nets (RPNs) have been proposed for modeling plans of agents in a multi-agent system [SH96]. 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 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 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.

In [HP99], 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.

Here, we first define new decision procedures for important problems: boundedness, finiteness and recursivity of languages. The two first problems are no more equivalent (unlike the similar problems for Petri nets) but remain decidable. In labelled Petri nets, in order to decide if some word belongs to the language, one builds the synchronized product of the automaton recognizing this word and the Petri net. This product is itself a Petri net and one decides if some markings are reachable. In RPNs such a method is impossible as the synchronized product of an automaton and a RPN is not necessarily a RPN (see the following paragraph). From a complexity point of view, since all our decision procedures use the reachability procedure for Petri nets, none of them is primitive recursive. Again this must be contrasted to the situation of Petri nets where the boundedness problem is exponential space complete.

Then we study the expressive power of RPNs. Following the works of Peterson [Pet81], we focus on finite word language aspects. At first, we show how to simulate computations of a Turing machine by synchronizing a RPN and a finite automaton. Building on this result, we establish that for any recursively enumerable language one can define a RPN language, a regular language and an homomorphism such that the former is the intersection of the latters via the homomorphism. So RPNs are much closer to Turing machine than Petri nets and context-free grammars. Indeed, the two latter models are closed under homomorphism and intersection with regular languages whereas the same closure applied to RPNs leads to Turing machines. A second important consequence is about infinite state systems model checking. For Petri nets, Ezparza [Esp97] has shown that the model checking of linear temporal logic on actions is decidable. Such a model checking procedure is undecidable for RPNs even for a restricted logic.

At last, we compare the model of RPNs with two other models combining Petri nets and context-free grammars features. Net systems have been introduced by A. Kiehn [Kie89a] in order to study partial-order semantics and composition of such systems. RPNs strictly include net systems w.r.t. to the language criteria. Process algebra nets (PANs) is a model of process algebra which include Petri nets and context-free grammars. In [May97], R. Mayr has shown that reachability is decidable for PANs. Here we show that RPNs include also PANs whereas the strict inclusion is an open problem. Due to the space restrictions, only sketches of proof are given in the paper. However in appendixes, we give complete proofs for the main propositions. These appendixes will be omitted in the final version. A technical report including all proofs will soon appear.

2 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 $\mathbb{N}^P$).
- Υ is an effective semi-linear set of final markings.

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 $E \to T_{ab}$. We denote by $v_0(tr)$ the root node of the extended marking tr. The edges E build a tree i.e. for each v different from $v_0(tr)$ there is one and only one $(v', v) \in E$ and there is no $(v, v_0(tr)) \in E$. Any ordinary marking can be seen as an extended marking composed by a unique node. The empty tree is denoted by $\perp$.

A *marked recursive Petri net* (N, tr_0) is a recursive net N associated to an initial extended marking tr_0. This initial extended marking is usually a tree reduced to a unique vertex.

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).

A *branch br* of an extended marking *tr* is one of the subtrees rooted at a son of $v_0(tr)$. One can associate to a branch a couple (t, tr) where t is the abstract transition which labels the edge leading to the subtree and *tr* the subtree taken in isolation. Let us note that the couple (t, tr) characterizes a branch.

In other words, given an extended marking *tr*, a branch *br* with its couple (t, tr') fulfills : tr' is a sub-tree of *tr* verifying $(v_0(tr), v_0(tr')) \in E$ (i.e. in *tr*, the root of tr' is a direct successor of the root of *tr*) and $A(v_0(tr), v_0(tr')) = t$ (i.e. in *tr*, the arc between the root of *tr* and tr' is labeled by t). Let us denote by $Branch(tr)$ the set of branches of an extended marking *tr*. The depth of an extended marking is recursively defined as 0 for $\perp$, 1 for for a unique vertex and (1 plus the maximum depth of its branches) for the general case.

An elementary step of a RPN may be either a firing of a transition or a closing of a subtree. At first, a transition t is enabled in a vertex v of an extended marking *tr* iff $\forall p \in P, M(v)(p) \geq W^-(p, t)$. The firing of an enabled transition t from a vertex v of an extended marking $tr = \langle V, M, E, A \rangle$ leads to the extended marking $tr' = \langle V', M', E', A' \rangle$ depending on the type of t.

***t* is an *elementary transition* ($t \in T_{el}$)** The thread associated to v fires such a transition as for ordinary Petri nets. The structure of the tree is unchanged. Only the current marking of v is updated.

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

***t* is an *abstract transition* ($t \in T_{ab}$)** The thread associated to v consumes the input tokens of t. It generates a new thread v' with initial marking $\Omega(t)$ the starting marking of t. Let us note that the identifier v' is a fresh identifier absent in V.

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

When a marking of a thread belongs to Υ, the RPN may execute a *cut step* denoted by τ. If the thread is associated to the root of the tree, this step leads to the empty tree. In the other case, the subtree rooted at this thread is pruned and the output tokens of the abstract transition which gave birth to the thread are added to the marking of its father.

- $V' = V \setminus Succ(v)$, $E' = E \cap (V' \times V')$, $\forall e \in E', A'(e) = A(e)$
- $\forall v' \in V' \setminus \{pred(v)\}, M'(v') = M(v')$
- $\forall p \in P, M'(pred(v))(p) = M(pred(v))(p) + W^+(p, A(pred(v), v))$

We denote by $tr \xrightarrow{t} tr'$ with $t \in T \cup \{\tau\}$ an elementary step of the RPN from tr to tr'. A firing sequence is usually defined : a transition sequence $\sigma = t_0 t_1 t_2 \dots t_n$ is enabled from an extended marking tr_0 (denoted by $tr_0 \xrightarrow{\sigma}$) iff there exists tr_1, tr_2, $\dots$, tr_n such that $tr_{i-1} \xrightarrow{t_i} tr_i$ for $i \in [1, n]$. We define the depth of σ as the maximal depth of tr_1, tr_2, $\dots$, tr_n.

We denote by $\mathcal{L}(N, tr_0, Tr_f)$ (where Tr_f is a finite marking set) the set of firing sequences of N from tr_0 to a marking of Tr_f. This set is called the language of N. More generally, the languages we will consider are defined via a labeling function. A *labeled recursive Petri net* is a recursive net and a labeling function h defined from the transition set $T \cup \{\tau\}$ to an alphabet Σ plus λ (the empty word). h is extended to sequences and then to languages. The language of a labeled recursive Petri net is defined by $h(\mathcal{L}(N, tr_0, Tr_f))$.

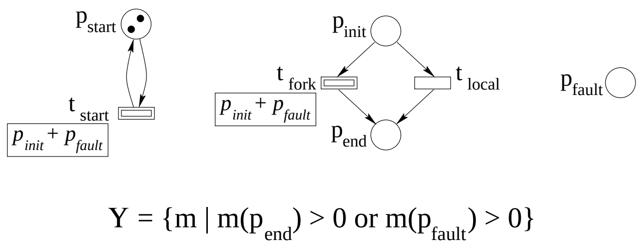

Fig. 1. a simple recursive Petri net

The figure 1 shows the modeling of two similar transactions (represented by two tokens in p_{start}). We represent an abstract transition by a double border rectangle and its initial marking is indicated in a frame. A transaction is started by the firing of the transition t_{start}. When initialized, the transaction may proceed locally by firing t_{local} or starts a new process by firing t_{fork}. Each process may achieve by reaching p_{end} or aborts since p_{fault} is always marked. In the latter case, the nested processes are also stopped due to the cut mechanism.

3 Decidability Results

3.1 Basic Results

The first step for tackling our decidability problems is to determine which words may be generated by the firing of an abstract transition. We distinguish betwen the general case and the case where the net initiated the abstract transition is required to close itself (i.e. to reach a marking of Υ).

Definition 1 (Closable abstract transition). An abstract transition t of a labeled RPN is *closable* w.r.t to a word w if there exists a firing sequence σ from $\Omega(t)$ to $\perp$ with $h(\sigma) = w$. Such a sequence is called a *closing sequence* of the abstract transition t w.r.t. w. $Closable^{(k)}(t, w)$ is true if there is a closing sequence of t w.r.t w of depth $\leq k$. $Closable(t, w)$ is true if there is a closing sequence of t w.r.t w. $Closable(t)$ is true if there is a closing sequence of t w.r.t. some word w.

Definition 2 (Observable abstract transition). An abstract transition t of a labeled RPN is *observable* w.r.t to a word w if there exists a firing sequence σ from $\Omega(t)$ to some marking with $h(\sigma) = w$. Such a sequence is called an *observable sequence* of the abstract transition t w.r.t. w. $Observable^{(k)}(t, w)$ is true if there is an observable sequence of t w.r.t w of depth $\leq k$. $Observable(t, w)$ is true if there is an observable sequence of t w.r.t w.

Proposition 3 (Closable and observable transitions). *Let N be a labeled RPN, let w_0 be a word then: $Closable(t, w_0)$, $Observable(t, w_0)$ and $Closable(t)$ are decidable.*

Sketch of Proof. The complete proof is given in appendix 7.1.

We proceed by successive rounds. At each round we compute the relations $Closable^{(k)}$ and $Observable^{(k)}$ for all abstract transitions and all subwords of w_0 (obtained by erasing letters) starting from $Closable^{(k-1)}$ and $Observable^{(k-1)}$. We stop if a round does not increase the relations. At first, as the number of abstract transitions and the number of subwords are finite and each successful round increases one of the relations $Closable$ or $Observable$, the algorithm stops. From the construction used in each round and informally described below, it is clear that when the algorithm stops, the relations $Closable$ and $Observable$ are stabilized.

We are looking for a sequence of depth k starting from $\Omega(t)$ and producing the subword w. If such a sequence exists, this word will be produced by a shuffle of firing subsequences of depth $< k$ in direct subtrees of the root and a subsequence of the root level. So we try to find a sequence w.r.t. any shuffle decomposition of w (there are only a finite number). The existence of subsequences in the inner level is checked by the relations computed at the previous round. In order to check the existence of the sequence at the root level, we define an ordinary net obtained from the original RPN including ordinary copies of abstract transitions which mimic the observable behaviours of the original abstract transitions at the root level. Morevover we add places in order to ensure that any firing in this net respects the precedence constraints of the chosen shuffle. We define as final markings a semi-linear set depending on which relation we want to check. At last we show that reaching the new semi-linear set in the ordinary net is equivalent to producing the word w in the RPN (and reaching a marking of Υ in case of the *Closable* relation). In order to check $Closable(t)$, we simply use the homomorphism h which maps each letter to the empty word λ and we check $Closable(t, \lambda)$. □

3.2 Boundedness, Finiteness

In this section we focus on boundedness and finiteness of RPNs. 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 space exponential in the size of the net [Rac78]. In RPNs, the equivalence does not hold but decidability remains for both properties. However, as we use in the decision procedure a reachability test [May81,Kos82] for some Petri nets, the complexity of our procedure does no more operate in primitive recursive space.

Proposition 4 (Boundedness of RPNs). *The boundedness problem is decidable for recursive Petri nets.*

Sketch of Proof. Let us suppose that some place p is unbounded, then for any integer n there is an extended reachable marking visited and a node of this extended marking for which the marking of p is greater than n. One can notice that the number of initial markings of nodes is finite (the initial markings of nodes composing the initial extended marking and the initial markings associated to abstract transitions). So the place p is unbounded in the root of some RPN with the same structure as the original one and an initial extended marking which may be: either a simple node labeled by some $\Omega(t)$ where t is enabled in some extended reachable marking, or some subtree of the initial extended marking.

Using a similar but simpler construction as given in the sketch of proof of proposition 3, one can show that the enabling of a transition in a reachable marking is decidable and that a place p is unbounded in the root of such a RPN N iff p is unbounded in an ordinary Petri net derived from N. □

Proposition 5 (Finiteness of RPNs). *The finiteness problem is decidable for recursive Petri nets.*

Sketch of Proof. The reachability set of a RPN is infinite iff either the RPN is unbounded or it is bounded and the depth of reachable extended markings is unbounded. From the previous proposition, we have just to decide for a bounded RPN whether the depth is unbounded. We build a reachability graph until either we finish the building or we find an extended marking tr such that there are two "fresh" nodes of tr issued by the same abstract transition, one ancestor of the other. We call a fresh node, a node which was not present in the initial extended marking. As the RPN is bounded, this construction will terminate. One can show that the termination in the second case is equivalent to the unboundedness of the depth. □

3.3 Recursivity of languages

As explained in the introduction, unlike the situation in Petri nets, the recursivity of the languages can not be proved using the reachability procedure for RPNs. However it turns out that these languages are still recursive.

Proposition 6 (Recursivity of RPNs languages). *The language of a labeled recursive Petri net is recursive.*

Sketch of Proof. At first, we observe that a sequence induces a classification of branches (i.e. immediate subtrees of the root) of the initial state and the final state: a *permanent branch* remains present during the whole sequence, a *closed branch* is present initially but disappears, an *opened branch* appears during the sequence and remains present. Then the word w produced by the sequence may be decomposed in subwords, where each subword is associated to either the root, either a branch or the firing of an abstract transition which fulfills the corresponding *Observable* or *Closable* relation (see proposition 3). Let us note that the number of classifications, decompositions and associations for a given sequence and a word is finite.

So the algorithm checks the existence of a sequence for each classification, decomposition and association. Inside a branch, it induces a recursive call, which is ensured to terminate as the sum of the depths for initial and final states decreases.

At the root level, the algorithm builds an ordinary net deduced from the RPN and checks some reachable relation. Roughly speaking, the ordinary net includes copies of elementary transitions for letters in the subword produced at the root level, special copies of abstract transitions for the subwords produced inside a closed or an opened branch or by the firing of an abstract transition. All these copies are firable at most once. Copies of elementary transitions labeled by the empty word and abstract transitions *closable* or *observable* w.r.t. to the empty word are also inserted. At last, auxiliary places are added to take into account the precedence relations between the beginnings and the ends of the different subwords in the word. □

4 Undecidability Results

In this section, we first show how a RPN and a finite automaton can be constructed to mimic the behavior of any Turing machine. First, we define a RPN $Net(Tm)$ (see Fig. 2) for which its language is a superset of possible behaviors of a Turing machine Tm. In the following, we restrict it by defining a regular language to be "synchronized" with the RPN.

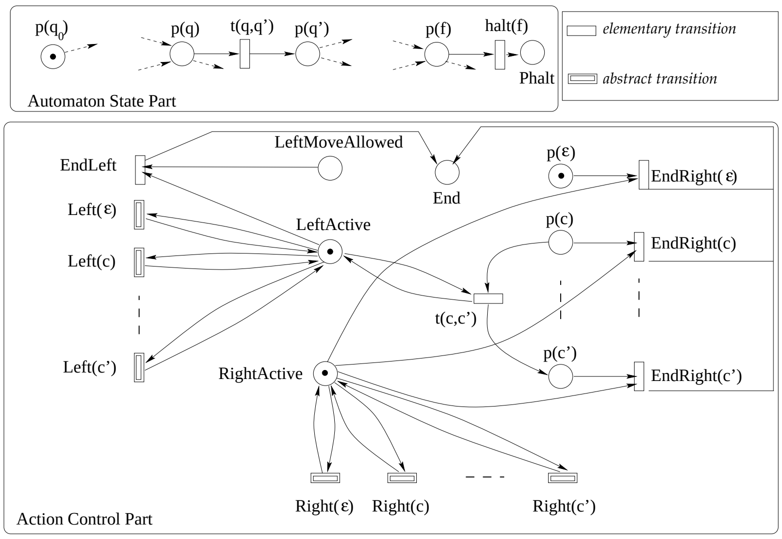

Fig. 2. a recursive Petri net modeling a superset of the Turing machine behaviors

The net is composed by two distinct parts. The upper part modelizes the automaton of the Turing machine by a classical state machine and the lower one its tape. There is one place $p(q)$ per state q of the Turing machine automaton and $p(c)$ per character of Σ its alphabet (including ϵ the blank). Moreover, the three places $RightActive$, $LeftActive$ and $LeftMoveAllowed$ control the possible transition firings in the different threads and the place End is used to close nodes (different from the root) of extended markings.

A configuration of the Turing machine is represented by an extended marking having two branches. Due to the structure of the net and the initial marking, firing transitions is only possible either at the root level or in the leaves. The state of the automaton is always stored in the root of the tree. For each letter different from ϵ and composing the state of the tape, there is a corresponding node in the extended marking. The part of the tape which is before and includes the tape head is stored in the left side of the tree beginning at the root and the remaining part is in the right side. The letters of the left side of the tree are stored in a descending way when an ascending order is used at the right side. Moreover, the last node at the right has the place $RightActive$ marked and the last node at the left has the place $LeftActive$ marked. Finally, the place $LeftMoveAllowed$ is marked in the last left node if this one is different from the root. This coding is illustrated in Fig. 3 with the three significant cases. Notice that the first blank is always explicitly represented in the extended markings.

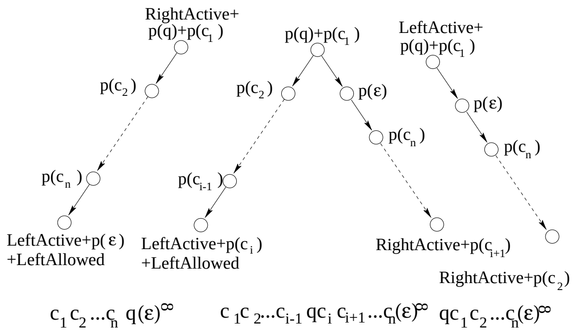

Fig. 3. extended markings coding Turing machine configurations

We briefly describe each kind of transitions. The transitions $\{t(q,q')\}$ having $p(q)$ as input place and $p(q')$ as output place simulate the state change of a step. Moreover, for any terminal state f, the corresponding place $p(f)$ has an output transition $halt(f)$ marking p_{halt}. Notice that, the firing of such transitions are only possible at the root level. The transitions $\{t(c,c')\}$ having $p(c)$ as input place and $p(c')$ as output place simulate the reading of c followed by the writing of c' on the tape of the machine. Notice, that transitions $\{t(c,c')\}$ have $LeftActive$ as input place. As this place is only marked in the last left node of the extended marking, only this thread can fire such a transition. The transitions $Left(c)$, $EndLeft$, $Right(c)$ and $EndRight(c)$ modelize the moves of the tape and are controlled by the place $LeftMoveAllowed$, $LeftActive$ and $RightActive$.

The ordinary markings associated to abstract transitions of the model are the following ones:

- $\forall \alpha \in \Sigma, \Omega\,(Left\,(\alpha)) = p\,(\alpha) + LeftActive + LeftMoveAllowed$
- $\forall \alpha \in \Sigma, \Omega\,(Right\,(\alpha)) = p\,(\alpha) + RightActive$

The final markings are defined by $\Upsilon = \{m \mid m(End) = 1 \text{ or } m(p_{halt}) = 1\}$. Closing a node by marking End corresponds to a move on the tape whereas marking p_{halt} may be possible only at the root of the extended marking and corresponds to reaching a terminal state of the machine.

The initial extended marking of $Net(Tm)$ is reduced to a unique node labeled with the ordinary marking $p\,(q_0) + p\,(\epsilon) + LeftActive + RightActive$.

The simulation of a step of the Turing machine on the current extended marking is illustrated in appendix 7.2. However this simulation requires firings in different threads and as in RPNs threads are not synchronized, this net may exhibit behaviors which are not simulations of the Turing machine.

As we want to simulate successful computations of the Turing machine, we consider the language $\mathcal{L}(Net(Tm), \perp)$ and so the set of sequences terminated by the firing of a $halt(f)$ transition followed by a cut τ. Among the firing sequences of the RPN, the simulating firing sequences can be characterized by a regular language. This language focuses on the simulation of successive steps of the machine independently of the tape content and the state automaton since such dependencies are ensured by the RPN. Let us call s a step of the machine. It is characterized by five finite domain parameters (input and output states, read and written characters and the move) and so there is only a finite number of such steps. Let us call $g(s)$ the finite set of RPN sequences simulating s (a complete definition of g can be found in appendix 7.2). So the desired language is: $Reg(Tm) = \left(\bigcup_{s\in\delta} g(s)\right)^* . \bigcup_{f\in F} halt(f)\cdot\tau$. The following proposition expresses the correctness of our simulation (proved in appendix 7.2).

Proposition 7 (Simulation of successful computations). *Let Tm be a Turing machine and σ be a word of δ^*. σ is a successful computation of Tm iff* $\exists\, seq \in \mathcal{L}(Net(Tm), \perp) \cap \left[g(\sigma). \bigcup_{f\in F} halt(f)\cdot\tau\right]$

Corollary 8 (Emptiness Problem). *The emptiness problem of the intersection between a (bounded) RPN language and a regular language is undecidable.*

The next proposition (proved in appendix 7.2) shows that recursive Petri net languages are much closer to Turing machine languages than context-free and Petri net languages.

Proposition 9 (RPN languages & recursively enumerable languages). *Let $\mathcal{L}$ be a recursively enumerable language. One can build a (bounded) RPN language $\mathcal{L}_1$, a regular language $\mathcal{L}_2$ and an homomorphism f such that $\mathcal{L} = f(\mathcal{L}_1 \cap \mathcal{L}_2)$.*

The restrictions on the behaviour of the RPN that we have specified with the regular language may also be specified by the linear temporal logic RTL (the fragment of LTL which replaces the operator *until* by the operator *sometimes*). So we also obtain an undecidability result for model checking of RPNs.

Proposition 10 (Model Checking Problem). *Checking the truth of a RTL formula on a (bounded) RPN is undecidable.*

5 Comparison with other models

RPNs combine features of Petri nets and context-free grammars. So it is interesting to compare RPNs with similar models. In her thesis, A. Kiehn has introduced a model called net systems [Kie89b]. 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. Informally, we add a place per net which loops on its transitions. The initial marking of the net is extended by a token in this place. The final making set is simply the union of final markings of the nets each one augmented by a token in its new place. Moreover as the languages of Petri nets are not included in the the language of net systems [Kie89b] we obtain the following proposition.

Proposition 11 (Net systems versus RPNs). *The family of net systems languages is strictly included in the family of RPNs languages.*

A Process Algebra Net (PAN) introduced by R. Mayr [May97] is defined as follows. It has a constant ϵ the empty term, a set of process variables $Var = \{X, Y, Z, ...\}$ and two operators: the sequential composition (.) (an associative operator) and the parallel composition ($\|$) (an associative and commutative operator). A term is a syntactically valid expression built with the constant ϵ, the variables and the operators. Each net has a finite set of rules Δ of the form:

$$X_1 \parallel X_2 \parallel ...X_n \xrightarrow{a} t$$

where X_i is a variable, t is a general term and a is an action which labels the transition rule.

The semantics of a term is defined inductively. If the term is the left-hand side of a rule then it may transform into the right-hand side. If the term is a parallel composition of terms, (using if necessary the commutativity and the associativity of the operator) it may transform itself by transforming one of its term. If the term is a sequential composition of terms, it may transform itself by transforming its first term. With the additional built-in equivalence $\epsilon.t = t$, the second term of a sequential composition becomes active when the first term becomes empty.

PAN is also an extension of context-free grammars and Petri nets (where a variable denotes a token in the corresponding place and the right-hand side of a rule is similar to the left-hand side). The main result for PANs is the decidability of the reachability problem. The next proposition shows that RPNs include PANs and thus that our reachability result subsumes the one for PANs [May97]. The simulation of a PAN by a RPN which is the key for the proof of the proposition is given in appendix 7.3.

Proposition 12 (PANs versus RPNs). *The reachability problem for PANs is reducible to the reachability problem for RPNs. The family of PANs languages is included in the family of RPNs languages.*

Whereas we do not know whether the inclusion is strict, we emphasize that the main difference between RPNs and the two other models is the ability to prune subtrees from the state. This mechanism is indispensable for the modeling of plans in multi-agents systems [SH96].

6 Conclusion

In this work, we have studied theoretical features of recursive Petri nets which complement the ones studied in [HP99] about reachability and expressivity. At first we have shown how to decide boundedness, finiteness of a RPN and we have proved that the languages of RPNs are recursive. Analyzing the expressiveness of RPNs, we have proved that any recursively enumerable language may be effectively obtained by an homomorphism of the intersection of a regular language and a RPN language. As a consequence, the general model checking for Petri nets becomes undecidable even for a restricted temporal logic. At last, we have shown that (to the best of our knowledge) RPN is the largest model including Petri nets and context-free grammars for which the reachability remains decidable

We have summarized in Fig. 8 (given in appendix 7.4) the position of RPNs relative to classical languages generators. 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

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[HP99] 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.

[Kie89a] A. Kiehn. 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, 1989.

[Kie89b] A. Kiehn. A structuring mechanism for Petri nets. Technical Report TUM-I8902, Technische Universitat Munchen, Germany, MARCH 1989.

[Kos82] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th Annual Symposium on Theory of Computing*, pages 267–281, 1982.

[May81] E.W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. 13th Annual Symposium on Theory of Computing*, pages 238–246, 1981.

[May97] R. Mayr. 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, 1997. Springer Verlag.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice Hall, 1981.

[Rac78] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.

[SH96] A. El Fallah Seghrouchni and S. Haddad. A recursive model for distributed planning. In *Second International Conference on Multi-Agent Systems*, Kyoto, Japon, December 1996.

7 Appendixes

7.1 Proof of proposition 3

At first, we can transform any labelled RPN such that the language is unchanged and the abstract transitions and the cut are labelled by the empty word. The figure 4 shows the transformation for an abstract transition : we split the transition in an elementary one which is labeled by the character followed by a λ-labeled abstract one. The marked place freezes the token game between the two firings. The transformation is similar for the cut.

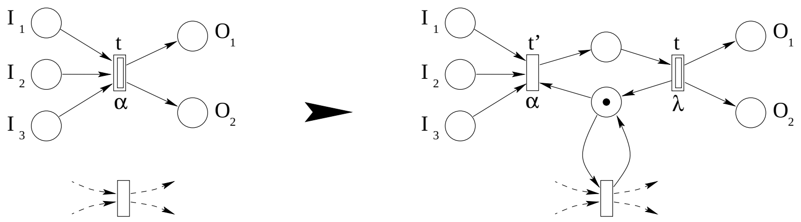

Fig. 4. recursive Petri nets having same languages

Then, we give a formal definition of the different kinds of branches inside a sequence.

Definition 13 (Permanent branch). Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'.
A couple of branches $((t_a, tr_a), (t_b, tr_b)) \in Branch(tr) \times Branch(tr')$ denotes a *permanent branch* in σ if $v_0(tr_a) = v_0(tr_b)$.

The previous definition expresses that the node $v_0(tr_a)$ is never removed by a cut step in $v_0(tr_a)$. Remark that in this case, we have necessary $t_a = t_b$.

If a branch is not permanent and occurs in the final marking then it has been "opened" by an abstract transition.

Definition 14 (Opened branch). Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch $(t_b, tr_b) \in Branch(tr')$ denotes an *opened branch* in σ if $\forall (t_a, tr_a) \in Branch(tr), v_0(tr_a) \neq v_0(tr_b)$.

In the same way, if a branch is not permanent and occurs in the initial marking then it has been "closed" by a cut.

Definition 15 (Closed branch). Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch $(t_a, tr_a) \in Branch(tr)$ denotes an *closed branch* in σ if $\forall (t_b, tr_b) \in Branch(tr'), v_0(tr_a) \neq v_0(tr_b)$.

At last, some branches may appear in an intermediate marking and disappear before the final marking.

Definition 16 (Transient branch). Let tr, tr' be two extended markings and σ be a firing sequence from tr to tr'. A branch (t_c, tr_c) of an extended marking tr'' visited by σ is a *transient branch* if
$\forall (t_a, tr_a) \in Branch(tr) \cup Branch(tr'), v_0(tr_a) \neq v_0(tr_c)$.

In this section, w_0 is a fixed word. A subword w' of a word w is obtained by erasing arbitrary characters in w. $\sigma_1 \parallel \sigma_2$ denotes a *particular* shuffle (clear from the context) of σ_1 and σ_2.

Definition 17 (Partition of a word). Let w be a word,
then $part = \langle \{w_i\}_{i\in\{1..n\}}, pos, start, end, ind\rangle$ is a partition of w iff :

- w is a shuffle of $\{w_i\}_{i\in\{1..n\}}$,
- $w_1 = \alpha_1 . \ldots . \alpha_m$ or $w_1 = \lambda$ (i.e. $m = 0$),
- $\forall 1 \leq j \leq m, pos(j)$ is the index of α_j in w
- $\forall 2 \leq i \leq n$,
  - $w_i \neq \lambda$,
  - $start(i)$ is the index of the first character of w_i in w,
  - $end(i)$ is the index of the last character of w_i in w,
- ind is an index which partitions the set $\{w_i\}_{i\in\{2..n\}}$ in two (possibly empty) subsets so that $2 \leq ind \leq n+1$.

The next definition which is indeed a construction of an ordinary Petri net is the kernel of our method. This construction of a $Testing$ Petri net will enable us to check closability and observability of order k, once the relations of order $k-1$ for all subwords have been computed. Let us note that there are numerous but finite such nets associated to a word and abstract transition.

Definition 18 (Testing Petri Net). Let $N = \langle P, T, W^-, W^+, \Omega, \Upsilon, h\rangle$ be a labeled recursive Petri net.

- let t be an abstract transition of N,
- let w be a word and $part = \langle \{w_i\}_{i\in\{1..n\}}, pos, start, end, ind\rangle$ be a partition of w (with $w_1 = \alpha_1 . \ldots . \alpha_m$),
- $\forall 1 \leq j \leq m$, let te_j be an elementary transition of N with $h(te_j) = \alpha_j$,
- $\forall 2 \leq i < ind$, let ta_i be an abstract transition such that $Closable^{(k-1)}(ta_i, w_i)$,
- $\forall ind \leq i$, let ta_i be an abstract transition such that $Observable^{(k-1)}(ta_i, w_i)$.

Then $Ntest^{(k)}(N, t, part, \{te_j\}_{j\in\{1..m\}}, \{ta_i\}_{i\in\{2..n\}})$ an *ordinary* Petri net is built by the following steps:

- its set of places is initialized to P with $\Omega(t)$ as initial marking,
- its set of transitions is initialized to the elementary transitions of N labeled by λ,
- for each abstract transition ta such that $Closable^{(k-1)}(ta, \lambda)$, ta is added as an ordinary transition ta' with the same input and output places,
- for each abstract transition ta, ta is added as an ordinary transition ta'' with the same input places and no output places,

- for each $1 \le j \le m$, we add a transition t'_j with the same inputs and outputs as te_j (some transitions may be duplicated here and after),
- for each $1 \le j \le m$, we add a place p'_j with output transition t'_j and (if $1 < j$) input transition t'_{j-1} . There is initially one token in p'_1,
- for each $2 \le i < ind$, we add t_i^- with the same input places as ta_i but with only one new output place p_i^+ and another transition t_i^+ which has p_i^+ as input place and the same output places as ta_i,
- for each $ind \le i \le n$, we add t_i^- with the same input places as ta_i but with no output place,
- for each $1 \le j \le m$ and each $2 \le i < ind$ such that $pos(j) < end(i)$ we add a place $p'(j,i)$ which is a new output place of t'_j and a new input place of t_i^+
- for each $2 \le i' \le n$ and each $2 \le i < ind$ such that $start(i') < end(i)$ we add a place $p^+(i',i)$ which is a new output place of $t_{i'}^-$ and a new input place of t_i^+
- for each $1 \le j \le m$ and each $2 \le i \le n$ such that $pos(j) > start(i)$ we add a place $p"(i,j)$ which is a new output place of t_i^- and a new input place of t'_j
- for each $1 \le i \le n$,
  - if $i = 1$ and $m > 0$, we add a new place P_1^{End} such that P_1^{End} is a new output place of t'_m,
  - if $2 \le i < ind$, we add a new place P_i^{End} such that P_i^{End} is a new output place of t_i^+,
  - if $ind \le i \le n$, we add a new place P_i^{End} such that P_i^{End} is a new output place of t_i^-.

Notation: When ever the context is clear, we denote by $Ntest$ an ordinary testing Petri net omitting the complete notation. Moreover, we associate to such a net $Ntest$:

- a set of tuples $Ab(Ntest) = \{\langle ta, mode, wa \rangle\}$ where ta is an abstract transition used to introduce elementary transitions in $Ntest$,
  $mode \in \{Observable, Closable\}$ denotes the required property for its introduction and wa is the subword required to be produced in $Ntest$ by the transformation of ta. Let us note that a same abstract transition may occur in different tuples.
- $Sclos(Ntest)$ the semi-linear set of ordinary markings defined by $\{m \mid \exists m' \in \Upsilon : m = m' + \sum P_i^{End}\}$.
- $Sobs(Ntest)$ the semi-linear set of ordinary markings defined by $\{m \mid m \ge \sum P_i^{End}\}$.

Lemma 19. *Let $N = \langle P, T, W^-, W^+, \Omega, \Upsilon, h \rangle$ be a labeled recursive Petri net and w be a word. $\forall k \ge 0, \forall t \in T_{ab}$,*

1. *$Closable^{(k)}(t,w)$ is true iff*

   - *$\exists part = \langle \{w_i\}_{i \in \{1..n\}}, pos, start, end, ind \rangle$ and*

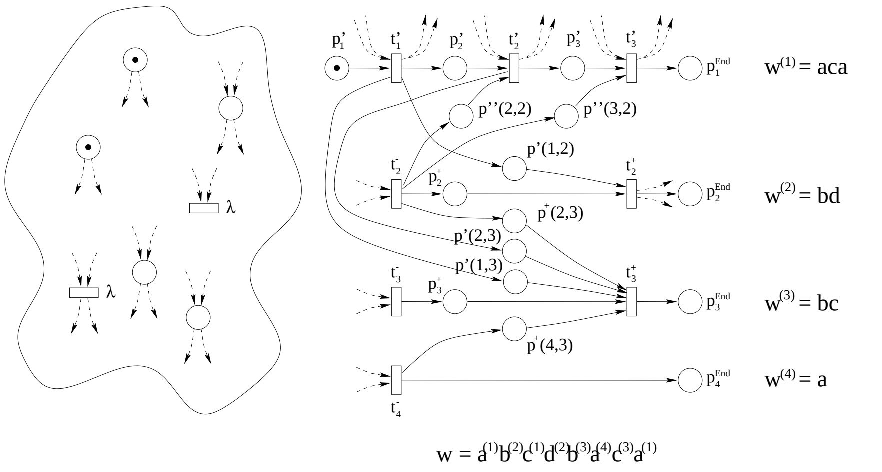

Fig. 5. an ordinary net $Ntest$

– $\exists Ntest^{(k)}(N, t, part, \{te_j\}_{j\in\{1..m\}}, \{ta_i\}_{i\in\{2..n\}})$ *such that a marking of* $Sclos(Ntest)$ *is reachable.*

Moreover, if $Closable^{(k-1)}(t,w)$ *is false then there is a tuple* $\langle ta, mode, wa\rangle \in Ab(Ntest)$ *such that either* $mode = Closable$ *and* $Closable^{(k-2)}(ta,wa)$ *is false or* $mode = Observable$ *and* $Observable^{(k-2)}(ta,wa)$ *is false.*

2. $Observable^{(k)}(t,w)$ *is true iff*

   – $\exists part = \langle \{w_i\}_{i\in\{1..n\}}, pos, start, end, ind\rangle$ *and*

   – $\exists Ntest^{(k)}(N, t, part, \{te_j\}_{j\in\{1..m\}}, \{ta_i\}_{i\in\{2..n\}})$ *such that a marking of* $Sobs(Ntest)$ *is reachable.*

   Moreover, if $Observable^{(k-1)}(t,w)$ *is false then there is a tuple* $\langle ta, mode, wa\rangle \in Ab(Ntest)$ *such that either* $mode = Closable$ *and* $Closable^{(k-2)}(ta,wa)$ *is false or* $mode = Observable$ *and* $Observable^{(k-2)}(ta,wa)$ *is false.*

Proof. We only handle the case of the $Closable$ relation, the other case follows the same pattern.

– Let us suppose there exists a closing sequence σ of t w.r.t. to w of depth $\leq k$. We are going to build a partition of w, a corresponding testing Petri net $Ntest$ and a sequence of reachability in this net leading to $Sclos(Ntest)$. Necessarily, the last firing of σ is a cut at the root level of the extended marking. let us considere the subsequence σ' of σ obtained by removing this cut. Since σ' starts from $\Omega(t)$, there can only be opened and transient branches. Furthermore, we can suppose that when the word produced by the firings in a opened branch is the empty word, then the subsequence is immediately fired since the new sequence produces the same word. Similarly, we can suppose that the subsequence of a transient branch which produces the empty word follows immediatly in w the firing of the opening abstract transition. Indeed for the reachability relation, only the root level is affected by the anticipation of the firings but here again it leads to increasing intermediate markings with necessarily the same marking before the cut and the new sequence produces the same word. Now we are in position to define a partition $part$.

- We index from 2 to $ind - 1$ the transient branches which produce non-empty words ($ind - 2$ is the number of such branches). w_i is the word produced by the i^{th} branch.
- We index from ind to n the opened branches which produce non-empty words ($n - ind + 1$ is the number of such branches). w_i is the word produced by the i^{th} branch.
- $w_1 = \alpha_1 \ldots \alpha_m$ is the word produced by the root level (which may be λ).

We denote by σ_1 the subsequence at the root level and te_j the elementary transition which produces α_j.

For $i \geq 2$, we denote by ta_i the abstract transition which opens the i^{th} branch and by σ_i the subsequence in the i^{th} branch excluding the optionnal last cut step. By construction, w is a shuffle of $\{w_i\}$ (with the pos, $start$ and end mappings usually defined).

- for all $i \in \{2, ind{-}1\}$, σ_i has a depth $\leq k{-}1$ and so $Closable^{(k-1)}(ta_i, w_i)$ is true.
- for all $i \in \{ind, n\}$, σ_i has a depth $\leq k{-}1$ and so $Observable^{(k-1)}(ta_i, w_i)$ is true.

The testing Petri net $Ntest^{(k)}(N, t, part, \{te_j\}_{j\in\{1..m\}}, \{ta_i\}_{i\in\{2..n\}})$ is now uniquely defined. It remains only to exhibit the firing sequence in $Ntest$. We iteratively transform σ' into a sequence σ_{test} of transitions in $Ntest$. We later demonstrate that it is a firing sequence.

- We considere a transient branch producing an empty word i.e.: $\sigma' = s_1.ta.\sigma_\lambda.\tau.s_2$ where ta is the abstract transition opening the branch, σ_λ is the sequence in the branch and τ is the cut step closing the branch. Then, the new σ' equals $s_1.ta'.s_2$.
- We considere an opened branch producing an empty word i.e.: $\sigma' = s_1.ta.\sigma_\lambda.s_2$ where ta is the abstract transition opening the branch, σ_λ is the sequence in the branch. Then, the new σ' equals $s_1.ta''.s_2$.
- We considere the i^{th} transient branch producing w_i i.e.: $\sigma' = s_1.ta_i.(\sigma_i \parallel s_2).\tau.s_3$. Then, the new σ' equals $s_1.ta_i^-.s_2.ta_i^+.s_3$.
- We considere the i^{th} opened branch producing w_i i.e.: $\sigma' = s_1.ta_i.(\sigma_i \parallel s_2)$. Then, the new σ' equals $s_1.ta_i^-.s_2$.
- for each $1 \leq j \leq m$, the transition firing te_j is changed into t'_j

After this transformation, the new sequence σ_{test} is a sequence of $Ntest$. Let us show that it is a firing sequence reaching to $Sclos(Ntest)$. We prove it by examining the different subsets of places of $Ntest$.

- As σ_{test} describes the effect of σ' at the root level, it is a firing sequence w.r.t. to the copy of P in $Ntest$ leading to a marking $\in \Upsilon$.
- As σ' produces the subword w_1 at the root level, the token in p'_1 is successively moved through p'_j by the firings of p'_j until P_1^{End}.
- As for $2 \leq i < ind$, ta_i^- is fired once and precedes the unique firing of ta_i^+, a token is produced in p_i^+ and moved to P_i^{End}.
- As for $ind \leq i \leq n$, ta_i^- is fired once, a token is produced in P_i^{End}
- As for each $1 \leq j \leq m$ and each $2 \leq i < ind$ such that $pos(j) < end(i)$, in σ' the final transition of the i^{th} branch must follow the firing of te_j, then in σ_{test} the firing of t'_j must precede the firing of t_i^+. Thus a token is first produced in $p'(j,i)$ before is is consumed.
- As for each $2 \leq i' \leq n$ and each $2 \leq i < ind$ such that $start(i') < end(i)$ in σ' the final transition of the i^{th} branch must follow the opening of the i'^{th} branch by the firing of $ta_{i'}$, then in σ_{test} the firing of $t_{i'}^-$ must precede the firing of t_i^+. Thus a token is first produced in $p^+(i',i)$ before is is consumed.
- As for each $1 \leq j \leq m$ and each $2 \leq i < n$ such that $pos(j) > start(i)$, in σ' the opening of the i^{th} branch must precede the firing of te_j, then in σ_{test} the firing of t_i^- must precede the firing of t'_j. Thus a token is first produced in $p"(i,j)$ before is is consumed.

which concludes the first part of the proof.

– Let us suppose now that there is a $Ntest$ and a sequence σ_{test} leading to $Sclos(Ntest)$. We will show how to build a closing sequence for t and w of order k. At first we associate to each abstract transition ta' used in $Ntest$ to produce the empty word, its closing sequence of order $k-1$, denoted σ_{ta} and to each t_i producing the subword w_i its closing or observable sequence of order $k-1$ denoted σ_i. In case of a closing sequence, σ_i does not include the cut step.

  Then we construct the sequence σ iteratively while "writing" w and reading simultaneously σ_{test} and the closing and observable sequences we will generate :

  We start with σ empty and at the beginning of σ_{test}. The marking at the root level will always correspond to the projection on P of the current marking in $Ntest$ ensuring that σ is firable.

  - If the next transition in σ_{test} corresponds to an elementary transition of N associated to the empty word, we complete σ by the firing of this transition at the root level.
  - if the next transition ta' in σ_{test} corresponds to an abstract transition ta then we complete σ by the firing of $ta.\sigma_{ta}$.
  - if the next transition ta'' in σ_{test} corresponds to an abstract transition ta then we complete σ by the firing of ta.
  - if the next transition in σ_{test} is t_i^- then we complete σ by the firing of t_i followed by the maximal prefix of σ_i composed by λ-labelled transition; we keep the suffix in the current state.

- if the next transition in σ_{test} is t_i^+ then the place p_i^+ ensures that t_i^- has been fired and that either the current suffix σ_i is empty or we have a suffix of σ_i in the current state where its first transition is labeled by some character say c_l the l^{th} character of w. In the former case, we do the cut step and in the latter case we proceed as follows.

  Let us suppose that the next character to write is $c_{l'}$ the l'^{th} of w with $l' < l$. $c_{l'}$ cannot be associated to a transition t'_j used for w_1 as the unmarked place $p'(j,i)$ forbids the firing of t_i^+. So $c_{l'}$ is associated to a word $w_{i'}$ with $i' > 1$. The place $p^+(i',i)$ ensures that $t_{i'}^-$ has been fired. Necessarily, $c_{l'}$ labels to the next transition to fire in $\sigma_{i'}$ so we fire it followed by a maximal subsequence of λ-labelled transitions in the suffix. Iteratively, we are ensured to reach the index l in which case we fire its corresponding transition followed again by a maximal subsequence of λ-labelled transitions. We repeat the procedure until we reach the end of σ_i and so we can do the cut step.
- if the next transition in σ_{test} is an elementary transition t'_j labeled by c_l the l^{th} character of w. Let us suppose that the next character to write is $c_{l'}$ the l'^{th} of w with $l' < l$. $c_{l'}$ cannot be associated to a transition $t'_{j'}$ used for w_1 as the unmarked place p'_j forbids the firing of t'_j. So $c_{l'}$ is associated to a word w_i with $i > 1$. The place $p"(i,j)$ ensures that t_i^- has been fired. Necessarily, $c_{l'}$ corresponds to the next transition to fire in σ_i so we fire it followed by the maximal subsequence of λ-labelled transitions in the sequence. Iteratively, we are ensured to reach the index l in which case we fire t'_j.
- if we have reached the end σ_{test}, it can remain suffix of observable sequences in which case we fire them piecemeal following the order given by w (let us recall that these sequences are independent).

At the end of σ_{test} as we have reached $Sclos(Ntest)$, the projection on P of the marking at the root level of $\sigma \in \Upsilon$.

The additionnal assertion is straightforward since
if for all tuples $\langle ta, mode, wa\rangle \in Ab(Ntest)$ then either $mode = Closable$ and $Closable^{(k-2)}(ta, wa)$ is true or $mode = Observable$ and $Observable^{(k-2)}(ta, wa)$ is true then σ is of order $k-1$. □

Proof. (of proposition 3) We remark that
$\forall k \leq 0, t, w, Closable^{(k)}(t,w) = Observable^{(k)}(t,w) = false$. We proceed by successive rounds, at each round we compute $Closable^{(k)}$ and $Observable^{(k)}$ from $Closable^{(k-1)}$ and $Observable^{(k-1)}$ trying for each item of the relations all partitions and all testing nets. We stop if a round does not increase the relations. At first, as the number of abstract transitions and the number of subwords are finite and each sucessfull round of the external increases one of the relation $Closable$ or $Observable$, the algorithm stops. The correctness of the algorithm follows directly from the previous lemma as it iteratively computes the $Closable^{(k)}$ and $Observable^{(k)}$ relations and we know also that when it stops these relation are stabilized. □

7.2 Proofs of section 4

Turing Machine A non-deterministic Turing machine is defined by a couple $Tm = \langle \Sigma, \langle Q, q_0, F, \delta \rangle \rangle$ where

- Σ is an alphabet including a particular blank character ϵ,
- $\langle Q, q_0, F, \delta \rangle$ is an automaton where
  - Q is a set of states,
  - $q_0 \in Q$ is an initial state,
  - $F \subseteq Q$ is a set of terminal states,
  - $\delta \subseteq Q \times \Sigma \times Q \times (\Sigma \setminus \{\epsilon\}) \times \{\rightarrow, \leftarrow, \uparrow\}$ is a transition table.

A *configuration* of a Turing machine is defined by the current state of its tape (i.e. an infinite word on Σ), the current position $i \in \mathbb{N}^*$ of its tape head and the current state q of its automaton. By the Turing machine semantic and the initial configuration, only a finite prefix of the tape is fulfilled by characters different to the blank in any reachable configuration. Then a configuration is denoted by an infinite sequence $c_1 \ldots c_{i-1} q c_i \ldots c_n \, (\epsilon)^\infty$ where $\forall j \in [1, n], c_j \neq \epsilon$. For a given configuration m, we denote by $q(m)$ the state of the automaton in m and by $c(m)$ the finite word of characters different to ϵ fulfilling the tape in m.

Initially, the tape is fulfilled by the character ϵ, the tape head is positioned on the head of the tape and the automaton is in the initial state q_0 (i.e. the Turing machine is in the configuration $q_0 \, (\epsilon)^\infty$).

A step $s = (q, c, q', c', d) \in \delta$ from a configuration $m = c_1 \ldots c_{i-1} q c_i \ldots c_n \, (\epsilon)^\infty$ is enabled (denoted by $m \xrightarrow{s}$) iff $c = c_i$ and $(d = \leftarrow) \Rightarrow (i \neq 1)$. The character c is called the *input* character.

Let $m = c_1 \ldots c_{i-1} q c_i \ldots c_n \, (\epsilon)^\infty$ be a configuration of Tm and $s = (q, c_i, q', c_i', d)$ a step enabled in m. The execution of s from m leads to the configuration m' (denoted by $m \xrightarrow{s} m'$) defined by:

- if $d = \leftarrow$ then $m' = c_1 \ldots q c_{i-1} c_i' \ldots c_n \, (\epsilon)^\infty$
- if $d = \rightarrow$ then $m' = c_1 \ldots c_{i-1} c_i' q \ldots c_n \, (\epsilon)^\infty$
- if $d = \uparrow$ then $m' = c_1 \ldots c_{i-1} q c_i' \ldots c_n \, (\epsilon)^\infty$

The character c_i' is called the *output* character. Enabling and execution notation are usually extended to computations.

A *computation* of a Turing machine is a finite sequence of steps from the initial configuration. We denote by $Comp\,(Tm) = \{\sigma \in \delta^* \mid q_0 \, (\epsilon)^\infty \xrightarrow{\sigma}\}$ the computation set of Tm.

Moreover, a computation is said to be *successful* if the automaton is in a terminal state in the reached configuration. We denote by $SuccComp(Tm) = \{\sigma \in Comp(Tm) : q_0 \, (\epsilon)^\infty \xrightarrow{\sigma} m \mid q(m) \in F\}$ the set of successful computations of a Turing Machine Tm.

The *language* of a Turing machine Tm, denoted by $\mathcal{L}(Tm)$, is the set of finite words on Σ written on the tape by the successful computations ($\mathcal{L}(Tm) = \{c \in \Sigma^* \mid \exists \sigma \in SuccComp(Tm) : q_0 \, (\epsilon)^\infty \xrightarrow{\sigma} m \wedge c = c(m)\}$).

Before developing proofs, we illustrate the effect of a step of the Turing machine on the current extended marking, in the case where the tape head moves to the right and the input character is not a blank, in Fig. 6. The execution of such a step of the Turing machine is simulated by the firing of transitions of $Net(Tm)$ in different threads of the extended marking. The transition $t(q, q')$ is fired in the root of the extended marking. The transition $t(c, c')$ and then the abstract one $Left(c_{i+1})$ are fired in the last left node. In the last right node, the terminal transition $EndRight(c_{i+1})$ is also fired. Notice that firings of different threads are independent and then can be fired in any order. This sequence is a valid sequence to simulate the behavior of the Turing machine but, as the synchronization between threads (as for the firings of $EndRight(c_{i+1})$ and $Left(c_{i+1})$) is impossible in recursive Petri nets, invalid simulations can be easily exhibited.

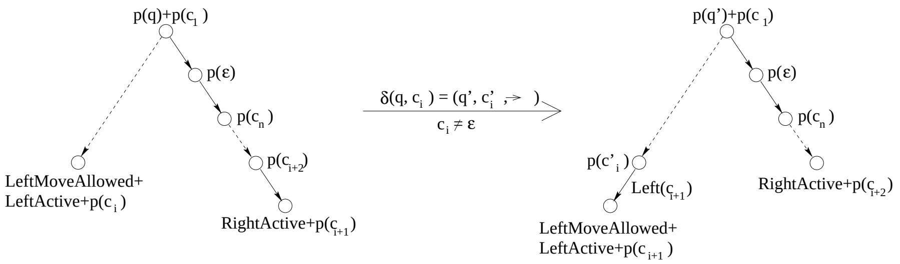

Fig. 6. simulation of a step of a Turing machine

Now, we define the regular language $Reg\,(Tm)$ on the transition set of the RPN $Net\,(Tm)$. We first define a mapping g from the transition table of Tm to finite languages of $Net(Tm)$.

Let $qcq'c'd \in \delta$, then:

- $d = \uparrow$ and $c \neq \epsilon \Rightarrow g\,(qcq'c'd) = t\,(c, c')\,.t\,(q, q')$
- $d = \uparrow$ and $c = \epsilon \Rightarrow g\,(qcq'c'd) = t\,(\epsilon, c')\,.Right\,(\epsilon)\,.t\,(q, q')$
- $d = \leftarrow$ and $c \neq \epsilon \Rightarrow g\,(qcq'c'd) = EndLeft.\tau.Right\,(c')\,.t\,(q, q')$
- $d = \leftarrow$ and $c = \epsilon \Rightarrow g\,(qcq'c'd) = EndLeft.\tau.Right\,(\epsilon)\,.Right\,(c')\,.t\,(q, q')$
- $d = \rightarrow$ and $c \neq \epsilon \Rightarrow g\,(qcq'c'd) = t(c, c').\sum_{\alpha \in \Sigma}\left(EndRight\,(\alpha)\,.\tau.Left\,(\alpha)\right).t\,(q, q')$
- $d = \rightarrow$ and $c = \epsilon \Rightarrow g\,(qcq'c'd) = t\,(\epsilon, c')\,.Left\,(\epsilon)\,.t\,(q, q')$

As usual, we extend g to a morphism between languages. The regular language $Reg(Tm)$ denotes the encoding (via g) of possible successful computations where a success is represented by the firing of a transition $halt(f)$.

$Reg\,(Tm) = \left(\bigcup_{s \in \delta} g\,(s)\right)^* . \bigcup_{f \in F} halt\,(f)\,.\tau.$

The following lemma expresses the correctness of our simulation.

Lemma 20 (Simulation of Turing machine computations). *Let Tm be a Turing machine and σ be a word of δ^*.*

1. *If σ is a computation of Tm leading to the configuration m then in $Net(Tm)$ there is a firing sequence $seq \in g(\sigma)$ leading to $tr(m)$.*
2. *If in $Net(Tm)$ there is a firing sequence $seq \in g(\sigma)$ then σ is a computation of Tm leading to a configuration m such that seq leads to $tr(m)$.*

Proof. By induction on n the length of σ. The base case ($n = 0$) is trivial. Suppose it is true for n. Let $\sigma = \sigma'.qcq'c'd$. We only handle the case where $d = \rightarrow$ and $c \neq \epsilon$.

1. Suppose σ is a computation of Tm of length $n+1$ with σ' leading to m' and $qcq'c'd$ a step from m' to m. We already know that there is $seq' \in g(m')$ leading to $tr(m')$ in $Net(Tm)$. Starting from $tr(m')$, let us complete seq' to some firing sequence belonging to $g(\sigma)$.
   $tr(m')$ is described in Fig. 6. As in its left node, $LeftActive$ and $p(c)$ are marked, $t(c, c')$ is enabled and its firing unmarks $p(c)$ and marks $p(c')$. In the right node, $RightActive$ and $p(c_{i+1})$ are marked, so $EndRight(c_{i+1})$ is enabled and its firing followed by a cut step prunes this node and marks $RighActive$ in the new right node. As $LeftActive$ and $LeftMoveAllowed$ are marked in its left node, one can fire $Left(c_{i+1})$ unmarking $LeftActive$ and creating a new left node. This node has for marking $p(c_{i+1})+LeftActive+LeftMoveAllowed$. At last, $p(q)$ is marked in the root of $tr(m')$ so one can fire $t(q, q')$ unmarking $p(q)$ and marking $p(q')$. The obtained extended marking is exactly $tr(m)$.
2. Let $seq \in g(\sigma)$ be a firing sequence of $Net(Tm)$. By induction, we know that σ' is a computation of Tm leading to a configuration m' such that the prefix of seq, $seq' \in g(\sigma')$ leads to $tr(m')$ in $Net(Tm)$. Now we examine all the ways to complete seq' into a firing sequence $\in g(\sigma)$ and we show that the only possible completion is the subsequence described in the first part of the proof and that it corresponds to a computation step of Tm in configuration m' (i.e. c is the input character and q the current state automaton). By the induction hypothesis, we know that seq' leads to $tr(m')$. So the marked place $p(c)$ in the left node of $tr(m')$ corresponds to the input character c in m' and that the marked place $p(q)$ in the root corresponds to the current automaton state q. At first, the left node of $tr(m')$ is the only one with $LeftActive$ and $p(c)$ marked so $t(c, c')$ is only enabled in the left node. The right node of $tr(m')$ is the only one with the place $RightActive$ marked, as $p(c_{i+1})$ is the only marked place among $p(c'')$ places in this node, $EndRight(c_{i+1})$ is the only possible firing followed again by the cut step (the only one possible in this state). Looking for the next firing $Left(c_{i+1})$, we only find $LeftActive$ marked in the left node. At last, the only marked place $p(q)$ is in the root and so $t(q, q')$ is enabled in this node.

Other cases are similar and even simpler. □

Now we are in position to characterize successful computations by our simulation.

Proof. (of proposition 7)

Let σ be a successful computation leading to a configuration m having $f \in F$ as current state automaton. From the Lemma 20, there is a sequence $seq' \in g(\sigma)$ leading to $tr(m)$. As the place $p(f)$ is marked in the root, $halt(f)$ is enabled and leads via a cut step to $\perp$.

Let seq be a sequence such that

$seq \in \mathcal{L}(Net(Tm), \perp) \cap \left[g(\sigma).\bigcup_{f\in F} halt(f)\ \tau\right]$ with $(seq = seq'.halt(f))$

From the Lemma 20, σ is a computation of Tm leading to m such that seq' leads to $tr(m)$. As $halt(f)$ is enabled in $tr(m)$ iff $p(f)$ is marked in its root, we conclude that f is the current automaton state and then that σ is successful. □

It is worth to notice that the RPN simulating the Turing machine is a bounded net. Indeed, this simulation is possible due to two factors, the recursiveness and the ability for parallel invocation of abstract transitions.

Proof. (of proposition 9)

Let Tm be a Turing machine generating $\mathcal{L}$. At first, we construct a Turing machine Tm' corresponding to Tm which has the following behavior

1. Its first action consists to write a special mark c_m at the first position of the tape and moves its tape head to the right.
2. It simulates Tm until Tm leads a terminal state. Special care must be taken to avoid entering terminal state while the tape head is on the special mark c_m.
3. It moves the tape head to the first position with the help of the special mark c_m.
4. It moves to the right visiting a particular intermediate automaton state corresponding to the input character and it terminates when the first character ϵ is read. As an example, if the state of the tape is $abcba\,(\epsilon)^{\infty}$, the sequence of automaton states which are visited is $q_a q_b q_c q_b q_a q_\epsilon$ with q_ϵ the unique terminal state of Tm'.

According to Prop. 7, for every successful computation of Tm' corresponds a word of $\mathcal{L}(Net(Tm', \perp)) \cap Reg(Tm')$ and vice versa. For instance, let $aab\,(\epsilon)^*$ be an accepted word of Tm, the corresponding word is of the form

$\sigma \ldots t(q_{c_m}, q_{c_a}) \ldots t(q_{c_a}, q_{c_b}) \ldots t(q_{c_b}, q_{c_b}) \ldots t(q_{c_b}, q_\epsilon) \ldots halt(q_\epsilon).\tau$

where σ is the sequence of $Net\,(Tm')$ corresponding to the three first steps of Tm' and the remainder of the sequence corresponds to fourth step.

Now taking $\mathcal{L}_1$ as $\mathcal{L}(Net(Tm', \perp))$, $\mathcal{L}_2$ as $Reg(Tm')$, we define f as follows:

$\forall c, c' \in \Sigma, f\,(t\,(q_c, q_{c'})) = c$ and otherwise $f\,(t) = \lambda$

The result is now clear from the construction. □

We finally give a last corollary giving necessary conditions for a family of languages to include RPNs languages.

Corollary 21 (Expressive power of RPNs). *Any family of recursive languages closed by intersection with a regular language and by homomorphism can not contain the family of recursive Petri net languages.*

Proof. Immediate from Prop. 9. □

7.3 Simulation of a PAN

At first we remark than a general rule of a PAN can be written:

$Y_1 \parallel ...Y_I \xrightarrow{a} X_1 \parallel ...X_J \parallel t_1.t'_1 \parallel ... \parallel t_K.t'_K$

The principles of the translation are the following ones:

- Independantly of the rules, places are defined for every variable.
- A special place *cpt* counts the number of branches in each vertex (it is incremented before the opening and decremented after the closing of the branch) which avoids to close a vertex which is not a leaf.
- Consequently, the semi-linear set of final markings is simply the empty marking.
- For the transaltion of the rule, an elementary transition is created which consumes $Y_1 \parallel ...Y_I$ acts as a and produces the X_j , $K.cpt$ and $\{p_k\}$ new places associated to the translation of $\{t_k.t'_k\}$.
- Then each p_i is the input of a new abstract transition with r_i a new place as the output (meaning that the term t_i has becomed empty) and q_i a new place as the starting marking of this abstract transition. This leads to a recursive translation of an "internal" rule $q_i \xrightarrow{\epsilon} t_i$.
- So $r_i + cpt$ is marked at the closing of the branch associated to t_i which "activates" t'_i. This leads again to a recursive translation of an "internal" rule $r_i \parallel cpt \xrightarrow{\epsilon} t'_i$

Here is a the recursive algorithm translation of a rule.

Algorithm 7.1 Generate

Generate($Y_1 \parallel ...Y_I \xrightarrow{a} X_1 \parallel ...X_J \parallel t_1.t'_1 \parallel ... \parallel t_K.t'_K$)
  begin
    for $k := 1$ **to** K **do**
      p_k := **new** Place();// p_k is a local identifier
      q_k := **new** Place();// idem
      r_k := **new** Place();// idem
      new AbstractTransition($in : p_k, out : r_k, \Omega : q_k, label : \epsilon$);
      Generate($q_k \xrightarrow{\epsilon} t_k$);
      Generate($r_k \parallel cpt \xrightarrow{\epsilon} t'_k$);
    od
      new ElementaryTransition($in : Y_1 + ... + Y_I, out : X_1 + ... + X_J + p_1 + ... + p_K + K.cpt,$
      $label : a$);
  end

Now it remains to translate the initial term which can be done by simulating a pseudo-rule with left-hand side a new place and right-hand side the initial term. Here is the general translation of a PAN.

Algorithm 7.2 Translate

Translate(t_0, Δ)
begin
for each variable X occuring in Δ or t_0 do
X := **new** Place();
od
cpt := **new** Place();
for each rule $R \in \Delta$ do
Generate(R);
od
p_0 := **new** Place();
Generate$(p_0 \xrightarrow{\epsilon} t_0)$;
new InitialMarking$(1.p_0)$;
new FinalMarking(0);
end

As an example, we give in figure 7 the translation of the rule:
$X_1 \xrightarrow{\epsilon} (X_2 \parallel X_3).(X_4.(X_2 \parallel X_3))$

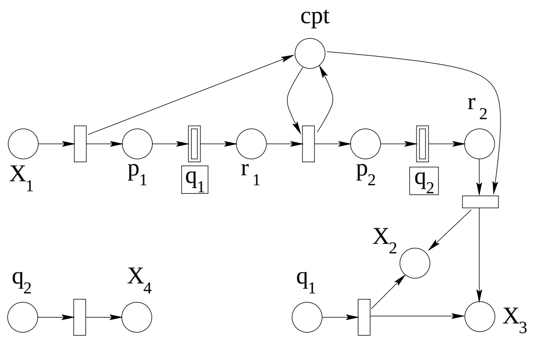

Fig. 7. Translation of a rule

7.4 Position of RPNs languages

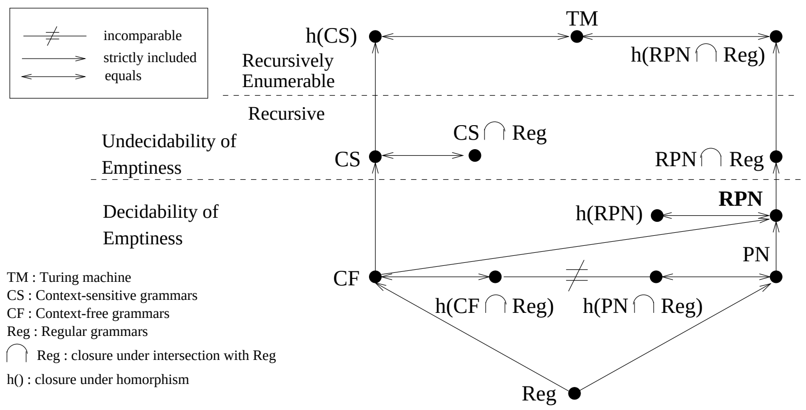

Fig. 8. hierarchies of models