Extensions of PRS

BLAISE GENEST

25 novembre 2001

Introduction

Petri Net systems are a well studied formalism. Nevertheless, some results, as the complexity of the reachability problem, although known decidable, are still open. A natural path to explore is to extend Petri Net systems, which was done with Reset Petri Nets, Transfert Petri Nets, Recursive petri nets [HP99], Process Rewriting Systems [May] etc... . For some extensions, reachability is known to be undecidable (Reset Petri Nets...), while for others, reachablity is known to be decidable (Recursive Petri Nets, Process Rewriting Systems...). We've chosen one of this system, namely PRS (process rewritting system) because of his universal definition, and try to extend it again, in order to find the frontier between decidability and undecidability for reachability, and may be find complexity of reachability for some extensions. Unfortunately, for the extensions chosen, namely allowing variables X or $*$, we've shown the reachability whether is undecidable or has the same complexity as for Petri Nets. Hopefully, theses results allowed us to show RPN (Recursive Petri Nets) is in some way a subset of PRS, and thus that the complexity of the reachability for RPN has the same complexity has for Petri Nets, which was known only for sequential RPN [HP00].

1 Notations and Definitions

1.1 PRS

Definition 1 *Let C be a set of constants. The configurations over C for PRS are called process terms. They are defined by the following grammar :*

$t ::= \epsilon \mid c \mid t_1.t_2 \mid t_1||t_2$, *where $c \in C$. We will denote T_C the set of all process terms. We juste consider the quotient of T_C with the associativity of $\bullet$ and $||$, and the commutativity of $||$.*

Definition 2 *A Process Rewritting System (PRS) is a tuple (C, R) where C is the set of constants and $R \subsetneq T_C \times T_C$ is the set of rules. The semantic of PRS is the transition system whose states are the process terms T_C, and transitions are described by Δ. Δ is defined inductively by the following way : for all $t, t', u, u' \in T$,*

$$\frac{t \longrightarrow t' \in R}{t \longrightarrow t' \in \Delta}, \quad \frac{t \longrightarrow t' \in \Delta}{t.u \longrightarrow t'.u \in \Delta}, \quad \frac{t \longrightarrow t' \in \Delta}{t||u \longrightarrow t'||u \in \Delta}$$

Example 1 *Let C be an alphabet. Let define a PRS system $(C \cup \{p, q\}, R)$, where for all $a \in C$, $p \longrightarrow p.a \in R$, $p \longrightarrow q \in R$, and $q.a \longrightarrow q \in R$.*

If we label $p \longrightarrow p.a$ and $q.a \longrightarrow q$ by the same label l_a, then the transitions from the configuration p to q are the palyndromic words of $\{l_a \mid a \in C\}$.

We want to extend PRS with some rules. An intuitive and powerful instrument is considering the variable X meaning "anything". Another intuitive instrument is considering the variable $*_{C'}$ meaning everything in C' where C' is a subset of C.

Definition 3 *An extended PRS is a tuple (C, R) where C is a set of constants and R is a set of rules of one of the following kinds :*

- $t \longrightarrow t'$ *with* $t, t' \in T_C$ *(the normal rules).*
- $t(X) \longrightarrow t'(X)$ *where* t, t' *are terms of* $T_{C\cup\{X\}}$.
- $t.* \longrightarrow t'$ *with* $t, t' \in T_C$.
- $t||*_{C'} \longrightarrow t'$ *with* $t, t' \in T_C$ *and* $C' \subseteq C$.

For this kind of rules, we can't use the old definition of Δ. We need subsets of Δ, namely $\Delta^{||}$ and $\Delta^{\bullet}$, that have different closure properties.

Definition 4 *The semantic of extended PRS is the transition system whose states are the process terms T_C, and transitions are described by Δ. Let $\Delta^{||}$ and $\Delta^{\bullet}$ be defined inducively by the following way :*

$\forall t, t', u, u' \in T_C$,

$$\frac{t \longrightarrow t' \in R}{\begin{array}{c} t \longrightarrow t' \in \Delta^{||} \\ t \longrightarrow t' \in \Delta^{\bullet} \end{array}}$$

$$\frac{t(X) \longrightarrow t'(X) \in R}{\begin{array}{c} t(u) \longrightarrow t'(u) \in \Delta^{||} \\ t(u) \longrightarrow t'(u) \in \Delta^{\bullet} \end{array}}$$

$$\frac{t.* \longrightarrow t' \in R}{t.u \longrightarrow t' \in \Delta^{||}}, \quad \frac{t||*_{C'} \longrightarrow t' \in R}{t||u||u' \longrightarrow t||u' \in \Delta^{\bullet}}, \text{ where } u, u' \in T_{C'} \times T_{\overline{C'}}$$

Now, we define the closure properties for $\Delta^{||}$ and $\Delta^{\bullet}$: For $u \neq \epsilon$,

$$\frac{t \longrightarrow t' \in \Delta^{||}}{\begin{array}{c} t||u \longrightarrow t'||u \in \Delta^{||} \\ t||u \longrightarrow t'||u \in \Delta^{\bullet} \end{array}}$$

$$\frac{t \longrightarrow t' \in \Delta^{\bullet}}{\begin{array}{c} t.u \longrightarrow t'.u \in \Delta^{||} \\ t.u \longrightarrow t'.u \in \Delta^{\bullet} \end{array}}$$

Last, we define $\Delta = \Delta^{||} \cup \Delta^{\bullet}$.

We can remark we don't allow rule with sequential composition of $*$ over a restricted alphabet because it is meaningless. Moreover, the condition of non-emptyness of u for the closure implies the u holding for $*$ is always the greatest possible.

1.2 properties of PRS

Definition 5 *Let $\mathcal{S} = (S, \Delta)$ and $\mathcal{S}' = (S', \Delta')$ be two transition systems. A simulation of $\mathcal{S}$ by $\mathcal{S}'$ is a mapping $g : S \longrightarrow S'$ such that*

For all $c, d \in S$, the configuration d is reachable (after any number of operations) from c with Δ iff $g(d)$ is reachable (after any number of operations) from $g(c)$ with Δ', ie

$$\forall c, d \in S, c \xrightarrow{*}_{\Delta} d \Leftrightarrow g(c) \xrightarrow{*}_{\Delta'} g(d)$$

If such a function exists, $\mathcal{S}$ is said to be simulable by $\mathcal{S}'$.

Moreover, if for all $\mathcal{S}$ of a class A, there exist $\mathcal{S}' \in A'$, then A is said to be simulable by A'.

Claim 1 *If $\mathcal{S}$ is simulable by $\mathcal{S}'$ with a polynomial time function, then the reachability for $\mathcal{S}$ is polynomialy reduced to the reachability in $\mathcal{S}'$.*

Proposition 1 – *Pushdown automata are simulable by PRS sytems.*
– *Petri nets are simulable by PRS systems.*

proof Let $N = (P, T, W^+, W^-)$ be a petri net.

We consider a PRS S over the alphabet P.

Each transition t with $W^-(t) = p_1. \cdots p_n$ and $W^+(t) = q_1 \cdots q_m$ is associated with a rule $r_t \in R$, where $r_t := p_1|| \cdots ||p_n \longrightarrow q_1|| \cdots ||q_m$.

□

Theorem 1 ([May]) *Reachabilty for PRS is decidable. The compexity is polynomially equivalent to reachability for Petri nets.*

1.3 RPN

We now define another class that extends petri nets. The philosophy of this system is that a transition (abstracted) can create another parallel petri net. If it dies, it kills its descendants, and gives a fixed amount of tokens to the petri net that gave birth to it.

Definition 6 *A Recursive Petri Net (RPN) is a tuple $N = (P, T, W^-, W^+, \Omega)$, where :*
- *P is the finite set of places,*
- *$T = T_a \cup T_e \cup T_f$ is a finite set of (abstracted, elementary, final) transitions,*
- *$W^- : T \longrightarrow (P \longrightarrow \mathbb{N})$ and $W^+ : (T_e \cup T_a) \longrightarrow (P \longrightarrow \mathbb{N})$ are respectively the pre and post flow functions,*
- *$(\Omega_t)_{t \in T_a} : P \longrightarrow \mathbb{N}$ is the initial labelling of the new node created by $t \in T_a$.*

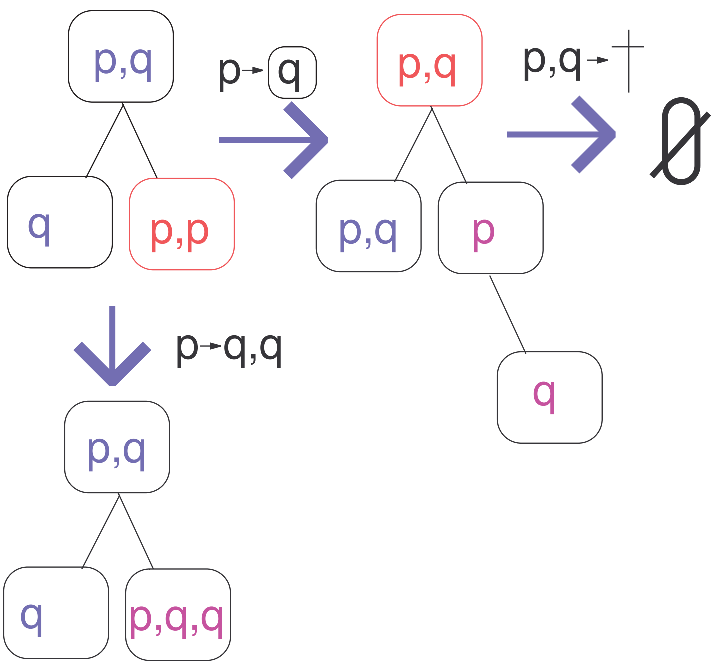
Definition 7 *A configurations of a RPN is called extended marked tree. It is an unordered tree where nodes are labeled by a marking for places, and the edges are labeled by transitions.*

More formaly, it is a tuple $tr = (N, Father, \lambda)$, where
- *N is the set of nodes.*
- *$Father : N \longrightarrow N \times T_a$ is the fatherhood relation, and asociates a node v with his father $Father(v)$ and the transition that gave birth to v.*
- *$\lambda : N \longrightarrow (P \longrightarrow \mathbb{N})$ is the labelling of the nodes.*

Definition 8 *The Semantic of RPN is the transition system whose configurations are extended marked trees, and transitions are given by Δ, whose description follows :*

for all marked tree c and all node n of c, for all transition $t \in T$ such that the transition is fireable, i.e. for all $i, \lambda(n)(i) > W^-(t)(p)$, then $(N, Father, \lambda) \longrightarrow (N', Father', \lambda') \in \Delta$, where

- $t \in T_e$:
  1. $N' = N$,
  2. $Father' = Father$,
  3. $\lambda'(m \neq n) = \lambda(m)$,
  4. $\lambda'(n) = \lambda(n) - W^-(t) + W^+(t)$.
- $t \in T_a$:
  1. $N' = N \cup \{n'\}$,
  2. $Father'(m \neq n') = Father(m)$,
  3. $\lambda'(m \neq n, n') = \lambda(m)$,
  4. $\lambda'(n) = \lambda(n) - W^-(t)$,
  5. $Father'(n') = (n, t)$,
  6. $\lambda'(n') = \Omega_t$.
- $t \in T_f$: *Let $father(n) = (n', t)$, and $tree(n)$ denote the subtree rooted at n.*
  1. $N' = N \backslash tree(n)$,
  2. $Father' = Father$,
  3. $\lambda'(m \neq n') = \lambda(m)$,
  4. $\lambda'(n') = \lambda(n) + W^+(t)$.



2 A usefull tool : the slave groups

2.1 Notations

Definition 9 *The set G of groups is defined by the following grammar :* $g ::= \epsilon \mid c \mid g_1||g_2$*, where* $c \in C$.

Definition 10 *A blank term $v()$ is such that there exists $x_1 \cdots x_n$ and $y_1 \cdots y_n$ such that for all $t \in T$,*

$$v(t) = [x_1||(x_2|| \cdots ||(x_n||t).y_n \cdots).y_2].y_1$$

Definition 11 *The set $SG_u \subset G$ of slave group of a process term u is the set of all the $sg \in G$ such that $u = t(x.sg.y)$, where t is a blank term and $x \neq \epsilon$.*

Claim 2 *We have* $|\bigcup_{u_0 \xrightarrow{*} u} SG_u| = O(|u_0| + |\Delta|)$*. Moreover, a set of linear size containing* $\bigcup_{u_0 \xrightarrow{*} u} SG_u$ *is computable in linear time.*

proof
Each group in $\bigcup_{u_0 \xrightarrow{*} u} SG_u$ appears either as a slave group of u_0 or in a rule.

$\square$

2.2 The Function f

Let (C, R) be a PRS
For each slave group $g \in SG$, we create a new constant c_g.
$\forall t \in T$,
- If $t \in G$, let $f(t) = t||test$. (It works also if $t = \epsilon$).
- Else $t = t_2||(t_1.g)$ with $g \in GS$ and $t_1, t_2 \in T$.
  Then let $f(t) = f(t_2)||(f(t_1)||c_g).suite$.

Example 2 *: if* $t = y\ ||(x.(x||y)).z$,
then $f(t) = f(y)\ ||\ f(x.(x||y))||c_z).suite$
ie $f(t) = test\ ||\ y\ ||\ [c_z||test||\ (test||x||c_{x||y}).suite]$

Proposition 2 *f is well defined, that is to say for all decomposition of t, the result of $f(t)$ is the same.*

proof Let show this property by induction over the size of t.
For $|t| = 1$, it's ok because there is only one decomposition.
Else $|t| > 1$. Assume that there exists two entirely specified functions g and g' that match f. So there exist two decompositions of t. That means $t = u.a\ ||\ v_1 = u'.a'\ ||\ v_2$, i.e. $t = u.a\ ||\ u'.a'\ ||\ v$, where u, u', v, v_1, v_2 are terms and a, a' are groups.
Moreover, we have $g(t) = g(u.a\ ||\ v_1) = [g(u)\ ||\ c_a\ ||\ test].suite\ ||\ [g(u'.a\ ||\ v)]$.
We don't know the way g decomposes $(u'.a||v)$, but we can apply the induction hypohesis to f, so to g, for $u'.a||v$ whose size is smaller than $|t|$.
So $g(u'.a||v) = [g(u')||c_{a'}||test].suite||g(v)]$.
Hence $g(t) = [g(u)||c_a||test].suite||[g(u')||c_{a'}||test].suite||g(v)$.

But every terms for which we evaluate f have a size smaller than $|t|$. So $f = g$ (because there is only one result for them, and g is one result possible).

So $g(t) = [f(u)||c_a||test].suite \ || \ [f(u')||c_{a'}||test].suite \ || \ f(v)$.

By simetry and associativity of $||$, we have $g(t) = g'(t)$.

□

Proposition 3 *f is injective*

proof

We can define a function $f^{-1} : Im(f) \longrightarrow T_C$:

We have $f^{-1}(g||test) = g$, where g is a group.

We have $f^{-1}1(t \ || \ [(b||c_g).suite]) = f^{-1}(a)||f^{-1}(b).g$ where g is a group.

□

Proposition 4 *For all blank term $v()$, there exists a blank term $v_f()$ such that $f(v(t)) = v_f(f(t))$.*

Conversely, for all blank term $v()$ s.t. there exists a with $v(f(a)) \in Im(f)$, there exists a blank term $v^f()$ such that $v(f(t)) = f(v^f(t))$.

proof

Let v be a blank term.

There exists $x_1 \cdots x_n$ and $y_1 \cdots y_n$ such that for all $t \in T$,

$$v(t) = [x_1||(x_2|| \cdots ||(x_n||t).y_n \cdots).y_2].y_1$$

Applying the definition of f, the result is obvious.

Conversely, $v(f(a)) = f(u)$, ie $u = f^{-1}(v(f(a)))$. With the definition of f^{-1}, we know there exists v^f s.t. $u = v^f(a)$. Easily, for all t, $f(v^f(t)) = f(f^{-1}(v(f(t))) = v(f(t))$.

□

2.3 A useful result

Theorem 2 *All PRS S is simulable by some S' with only one slave group.*

proof

Let $S = C, R$ be PRS.

W.l.o.g, we can suppose there is no rule of kind $\epsilon \longrightarrow u$.

We construct $S' = (C', R')$. We define $C \subset C'$.

Hence, $C' = C \cup \{suite; test\} \cup \{c_g \mid g \in GS\}$.

For the rules, $R' = \{f(a) \longrightarrow f(b) \mid a \longrightarrow b \in R\}$.

Let notice that the rule $(test||c_g).suite \longrightarrow g$ is already in R' because of the implicit rule $\epsilon.g \longrightarrow g$.

In the following, $v()$ designs blank terms.

Let $t \longrightarrow t'$ in Δ. Let $a \longrightarrow b \in R$ be the corresponding rule : $t = v(a), t' = v(b)$. We have $f(t) = v_f(f(a))$ and $f(t') = v_f(f(b))$. So $f(t) \longrightarrow f(t')$ in Δ'.

Conversely, Let $f(t) \longrightarrow f(t')$ in Δ'. Let $f(a) \longrightarrow f(b) \in R$ be the corresponding rule : $f(t) = v(f(a)), f(t') = v(f(b))$. We have $f(t) = f(v^f(a))$ and $f(t') = f(v_f(b))$. Because f is injective, $t = v^f(a)$ and $t' = v^f(b)$, so $t \longrightarrow t' \in \Delta$.

□

All the slave groups are now master groups (they appear in the first level of sequential hierarchy), so we can use them more easily. The usefullness of *test* will be obvious later.

3 Extension of PRS with variable X

We defined the semantic of PRS with the variable X. The question is :"does this variable gives more power to PRS". In the general case, it is easy to see it allows to simulate a 2 counters machine. We now try to find the minimal rules with X allowing to simulate a 2 counters machine, and the maximal rules that don't give more power to PRS.

Theorem 3 *The following rules are simulable by PRS.*

1. $a.X \longrightarrow b.X$.
2. $a||X \longrightarrow b||X$.
3. $X.g \longrightarrow X.g'$ *where* g, g' *are groups.*

proof

1. We just have to replace each rule $a.X \longrightarrow b.X$ by $a \longrightarrow b$, because the rules are closed under sequential composition.
2. We just have to replace each rule $a||X \longrightarrow b||X$. by $a \longrightarrow b$, because the rules are closed under parrallel composition.
3. Let (C, R) be a PRS system with one rule $r = X.g \longrightarrow X.g \in R$, where g, g' are groups.
   We apply the result of section 2.3 to $(C, R\backslash\{r\}$, giving a PRS (C', R').
   We had to R' the rule $r' = c_g||test \longrightarrow c_{g'}||test$.
   Assume that $t = v(u.g) \xrightarrow{r} t' = v(u.g')$.
   So $f(t) = v^f(f(u.g)) = v^f(test \mid\mid (f(u)||c_g).suite)$.
   Since $f(u) = test||w$, we have $f(t) \xrightarrow{r'} v^f(f(u.g'))) = f(t')$.
   Conversely, if $f(t) \xrightarrow{r'} f(t')$, we have $f(t) = v(f(u.g))$.
   So $v(f(u.g')) = f(v^f(u.g')) = f(t')$.
   Since f is injective, $t = v^f(u.g)$ and $t' = v^f(u.g')$.
   So $t \longrightarrow t'$.

$\square$

If we allow few more general rules, we can simulate any 2 counters machine.

Theorem 4 *If one can use rules of the kind $p||(a.X) \longrightarrow q||(b.X)$, he can simulate any 2 counters machine*

proof Let M be a 2 counters machine. Let define (C, R) a PRS associated with (S, Δ). $C = \{p \mid$ p is a line of the machine $\} \cup \{c_1, c_2, c_1^0, c_2^0\}$.
If p : "if $c_k = 0$ then goto q ", we add the rule $p||c_k^0 \longrightarrow q||c_k^0$ to R.
If p : "if $c_k > 0$ then $c_k - -$ and goto q ", we add $p||c_k.X \longrightarrow q||X$ to R.
If p : "$c_k + +$ and goto q ", we add the rule $p||c_k.X \longrightarrow q||c_k.c_k.X$ to R.
Let define a function g :

$$g(p, m, n) = p \ || \ c_1.c_1.\cdots.c_1.c_1^0 \ || \ c_2.\cdots.c_2.c_2^0$$

with m symbols c_1 and n symbols c_2.
(C, R) simulates M with g. □

Theorem 5 *If one can use rules of the kind $p.(a||X).b \longrightarrow q.(c||X).d$, he can simulate any 2 counters machine*

proof Let M be a 2 counters machine. Let define (C, R) a PRS associated with (S, Δ). $C = \{p \mid$ p is a line of the machine $\} \cup \{c_1, c_2, c_1^0, c_2^0\}$.
If p : "if $c_1 = 0$ then goto q ", we add the rule $p.c_1^0 \longrightarrow q.c_1^0$ to R.
If p : "if $c_2 = 0$ then goto q ", we add the rule $p.X.c_2^0 \longrightarrow q.X.c_2^0$ to R.
If p : "if $c_1 > 0$ then $c_1 - -$ and goto q ", we add $p.(c_1||X) \longrightarrow q.X$ to R.
If p : "if $c_2 > 0$ then $c_2 - -$ and goto q ", we add $p.X.c_2 \longrightarrow q.X$ to R.
If p : "$c_1 + +$ and goto q ", we add $p.X \longrightarrow q.(X||c_1)$ to R.
If p : "$c_2 + +$ and goto q ", we add $p.X \longrightarrow q.X.c_2$ to R.
Let define a function g :

$$g(p, m, n) = p \ . \ (c_1||c_1||\cdots||c_1||c_1^0) \ . \ c_2.\cdots.c_2.c_2^0$$

with m symbols c_1 and n symbols c_2.
(C, R) simulates M with g. □

Theorem 6 *If one can use rules of the kind $X.u \longrightarrow X.u'$ where u, u' are process terms, he can simulate any 2 counters machine*

proof Let M be a 2 counters machine. Let define (C, R) a PRS associated with (S, Δ). $C = \{p \mid$ p is a line of the machine $\} \cup \{c_1, c_2, c_1^0, c_2^0\}$.
If p : "if $c_1 = 0$ then goto q ", we add the rule $c_1^0.p \longrightarrow c_1^0.q$ to R.
If p : "if $c_2 = 0$ then goto q ", we add the rule $X.p.c_2^0 \longrightarrow X.q.c_2^0$ to R.
If p : "if $c_1 > 0$ then $c_1 - -$ and goto q ", we add $X.c_1.p \longrightarrow X.q$ to R.
If p : "if $c_2 > 0$ then $c_2 - -$ and goto q ", we add $X.p.c_2 \longrightarrow X.q$ to R.
If p : "$c_1 + +$ and goto q ", we add $X.p \longrightarrow X.c_1.q$ to R.
If p : "$c_2 + +$ and goto q ", we add $X.p \longrightarrow X.q.c_2$ to R.
Let define a function g :

$$g(p, m, n) = (c_1^0.c_1.c_1.\cdots.c_1) \ .p. \ c_2.\cdots.c_2.c_2^0$$

with m symbols c_1 and n symbols c_2.
(C, R) simulates M with g. □

Theorem 7 *If one can use rules of the kind $a.X \longrightarrow a||X$ and $a||X \longrightarrow a.X$, he can simulate any 2 counters machine*

proof Let M be a 2 counters machine. Let define (C, R) a PRS associated with (S, Δ). Let $C = \{p_1, p_2 \mid$ p is a line of $M\} \cup \{c_1, c_2, debut, c_1^0, c_2^0\}$.

For all line p : "[...] goto q" of M, we had

– $p_1||X \longrightarrow p_1.X$

– $p_2.X \longrightarrow p_2||X$

– $p_2 \mid\mid debut \longrightarrow q_1 \mid\mid debut$

If p : "if $c_k = 0$ then goto q ", we add the rule $p_1.c_k^0.c_k^0 \longrightarrow p_2.c_k^0.c_k^0$ to R.

If p : "if $c_k > 0$ then $c_k - -$ and goto q ", we add $p_1.c_k^0.c_k \longrightarrow p_2$ to R.

If p : "$c_1 + +$ and goto q ", we add $p_1.c_k^0 \longrightarrow p_2.c_k^0.c_k$ to R.

Let define a function g :

$$g(p, m, n) = p_1 \mid\mid debut \mid\mid (c_1^0.c_1.c_1.\cdots.c_1.c_1^0) \mid\mid (c_2^0.c_2.\cdots.c_2.c_2^0)$$

with m symbols c_1 and n symbols c_2.

(C, R) simulates M with g : obviously, if $u \xrightarrow{*} u'$, then $g(u) \xrightarrow{*} g(u')$.

Conversely, assume that $g(u) \xrightarrow{*} g(u')$. Let take a minimal path ρ that goes from $g(u)$ to $g(u')$. If $u = u'$, $u \longrightarrow u'$.

We can assume that there isn't any v such that $g(v)$ belongs to ρ. We have $g(u) = g(p, m, n)$ and $g(u') = g(q, m', n')$. There should be a rule $p_2 \mid\mid debut \longrightarrow p_1' \mid\mid debut$ in ρ. But then we see a $g(v)$. So $v = u'$ and $p' = q$.

Moreover,there should be a rule that goes from p_1 to p_2 : There are three posibilities, they all asked the configuration to be of the kind :

$p_1.c_k^0.c_k.\cdots.c_k.c_k^0 \mid\mid debut \mid\mid c_{k'}^0.c_{k'}.\cdots.c_{k'}.c_{k'}^0$.

Hence, we should have any positive number k of rule $p_1||X \longrightarrow p_1.X$ before the rule that goes from p_1 to p_2, and any positive number l of rule $p_2.X \longrightarrow p_2||X$ after.

If $k > 1$, after the first rule, we have $p_1.X$, so the other firing of the rule $p_1||X \longrightarrow p_1.X$ holds with $X = \epsilon$, that is to say we can forget it. So $k = 1$. Moreover, $X = c_k^0.c_k.\cdots.c_k.c_k^0$, else there can't be any firing of a rule from p_1 to p_2.

After the rule that goes from p_1 to p_2, we have a configuration $p_2||debut||t$. But the transition that is taken after is $p_2 \mid\mid debut \mid\mid t \longrightarrow q_1 \mid\mid debut \mid\mid t = g(u')$, so $l = 1$ and the X in $p_2.X \longrightarrow p_2||X$ denote $c_k^0.c_k.\cdots.c_k.c_k^0$.

Hence, we have a rule that goes from u to u'.

$\square$

4 Extension of PRS with variable $*_\sigma$

We've seen if one can use rules with a variable X that is usable in a right and left term of rules, he can simulate any 2 counters machine.

May be if we give less power to X, for exemple X can't be use in a right term, the reachability is decidable. Such rules are easy to simulate with PRS, so we have to consider a restriction of X to have more control over theses rules. We call the new variable $*$.

Theorem 8 *Rules of kind $a.* \longrightarrow b$ can be simulated with PRS.*

proof

Let assume that there is only one rule $r = a.* \longrightarrow b \in R$

We apply the result of section 2.3 to $(C, R\backslash\{r\})$, giving a PRS (c', R').

we add constant $videseq_b$ to C'. Then, we add to R' rules

- $(a \mid\mid test \mid\mid c_g).suite \ \longrightarrow \ videseq_b$,
- $(test \mid\mid videseq_b \mid\mid c_g).suite \ \longrightarrow \ videseq_b$ for all $g \in G$,
- $test \mid\mid videseq_b \mid\mid c \ \longrightarrow \ test \mid\mid b \mid\mid c$ where $c \in C$.

If the system takes the third rule, this means $c \neq c_g$, so the group is in parrallel composition with another group : it's the end of the sequential composition.

It is obvious if $t \xrightarrow{r} t'$, then $f(t) \xrightarrow{*} f(t')$. Assume that $f(t) \xrightarrow{*} f(t')$. Let take ρ a minimal path from $f(t)$ to $f(t')$. We can assume there is no other $f(t'')$ in ρ. Moreover, we can assume there is no rules but only ones of the kind we add to R'. Each of these rules need a $videseq$, so $videseq$ is created. Only the first rule create it, so there is a $(a||test||c_g).suite \longrightarrow videseq_b$ in the begining of ρ. That is to say the transition $a.* \longrightarrow b$ can be fired from t to obtain some t''.

In $f(t')$, there is no $videseq$, so it has been deleted :

there is a $test \mid\mid videseq_b \mid\mid c \ \longrightarrow \ test \mid\mid b \mid\mid c$ in the end of ρ. All the other rules taken are of the second kind. Hence, $f(t')$ coresponds to $f(t'')$.

Moreover, we can add each rule $a.* \longrightarrow b$ once at a time, creating a new constant $videseq$.

□

Theorem 9 *If one can use rules of kind $a||*_\sigma \longrightarrow b$, he can simulate reset petri nets.*

proof

We make the same simulation as with petri nets, and use the new rules to simulate a reset.

Let $N = (P, T, W^+, W^-)$ be a reset petri net.

We consider a PRS (P, R).

Each transition t with $W^-(t) = p_1. \cdots p_n$ and $W^+(t) = q_1 \cdots q_m$ is associated with a rule $r_t \in R$, where $r_t := p_1|| \cdots ||p_n \longrightarrow q_1|| \cdots ||q_m$.

A reset t over the place i with $W^-(t) = p$ and $W^+(t) = q$ is associated with a rule $p||*_{\{c_i\}} \longrightarrow q$.

Let g denote the function $g(m_1, \cdots, m_n) = (c_1|| \cdots ||c_1) \mid\mid \cdots \mid\mid (c_n|| \cdots ||c_n)$ with m_i symbols c_i.

So (P, R) simulates N with g.

□

So PRS can't simulate this kind of rules, because reachability is undecidable for reset petri nets.

Theorem 10 *If one can use rules of kind* $a||* \longrightarrow \epsilon$*, he can simulate recursive petri nets.*

proof

Let $N = (P, T, W^+, W^-)$ be a recursive petri net. Let define a PRS (P, R) associated with N.

- For each elementary transition t of N, we add $W^-(t) \longrightarrow W^+(t)$ to R.
- For each abstracted transtion t of N, we add $W^-(t) \longrightarrow \Omega_t.W^+(t)$ to R.
- For each final transition t of N, we add $W^-(t)||* \longrightarrow \epsilon$ to R.

Let g(node labeled by $m_1, \cdots, m_n) = (c_1|| \cdots ||c_1) \ || \cdots || \ (c_n|| \cdots ||c_n)$ with m_i symbols c_i.

Let t be an extended marking tree rooted at r. The children of r are $t_1 \cdots t_n$, and the edge from t to t_i is labelled by the transition u_i.

Then $g(t) = g(r) \ || \ g(t_1).W^+(u_n) \ || \ \cdots \ || \ g(t_n).W^+(u_n)$.

The simulation is easy to prove since one rule of R coresponds to one rule of N.

□

Remark 1 We just need PRS system where rules have no sequential concatenation on right-hand side (called (P,G)-PRS). Conversely, one can simulate a (P,G)-PRS with RPN.

Theorem 11 *Rules of kind* $a||*_\sigma \longrightarrow \epsilon$ *can be simulated with PRS*

proof

Let assume that there is only one rule $r = a||* \longrightarrow \epsilon \in R$.

We apply the result of section 2.3 to $(C, R\backslash\{r\})$, giving a PRS (c', R').

We had constants *vide* and *videparral* to C'. Then, we add the ability for each group to close itself.

In order to do that, we add the following rules to R'

- $test \longrightarrow vide$.
- $vide||c \longrightarrow vide$ for all constant $c \in C$.
- $(vide||c_g).suite \longrightarrow vide$ for all slave group $g \in GS$.
- $test||a \longrightarrow videparral$.
- $videparral||d \longrightarrow videparral$ for all $d \in C \cup \{vide\}$.
- $videparral \longrightarrow \epsilon$
- $(videparral||c_g).suite \longrightarrow g$ for all slave group $g \in GS$.

If one takes the $videparral \longrightarrow \epsilon$ too early, there is a deadlock (no *test* nor *videparral* in a group, so this group can't be changed by a rule).

In the same way, if a $tet \longrightarrow vide$ is taken while none of its brother has a *videparral*, then this *vide* can't be deleted, and there will be a deadlock.

Let note that we can delete with *vide* only constants from σ. If *vide* begins to delete constant but there are constants $\notin \sigma$ that can t be deleted, there is a deadlock. So we can simulate a rule $a||*_\sigma \longrightarrow \epsilon$.

Moreover, we can add each rule $a||*_\sigma \longrightarrow \epsilon$ once at a time, creating new constants *vide* and *videparral*.

□

Theorem 12 *Reachability for RPN is decidable, and its complexity is polynomialy equivalent to reachability for Petri Nets.*

proof
it is straightforward from **Claim 1** and **Theorem 1,10 and 11**.

□

Conclusion

With the extensions we have considered, we just need few rules to make the reachability undecidable. Nevertheless, some particular rules are simulable by PRS. To show that, an important thing is to understand there are only a few slave groups, that is to say we can't do many things we them. Moreover, there seems to be no alternative between being simulable by PRS and having the reachability procedure undecidable. At last, this paper shows once more the universality of PRS, whose subsets are bisimilar to Petri Nets for (P,P) PRS, Pushdown Automata for (S,S) PRS ... and now RPN that is very close to (P,G)-PRS : actualy, nothing is different but the ending transitions, which explains that we need another tool to simulate RPN with PRS.

Références

[HP99] S. Haddad and D. Poitrenaud, *Theoretical Aspects of Recursive Petri Nets, 1999*

[HP00] S. Haddad and D. Poitrenaud, *A Model Checking Decision for Sequential Recursive Petri Nets, 2000*

[May] R. Mayr, *Process Rewrite System, 2000*