

SIMULATION OF COLORED PETRI NETS IN CONCURRENT LOGIC PROGRAMMING

Ah. Naïm - J. Tisseau

*LIMI / Ecole Nationale d'Ingénieurs de Brest
Case Postale 15 - 29608 Brest Cedex - France
Téléphone : (33).98.05.66.65 - Télécopie : (33).98.05.66.21
e-mail : naim@li2.enib.fr*

Abstract :

This paper deals with a modelization of colored Petri Nets (colored PN) in concurrent logic programming. The manufacturing system simulation well shows a growing interest for the PN implementation in logic programming, where the nodes of PN will be seen as interprocess communication.

Keywords :

Simulation, Colored Petri Nets, Concurrent logic programming, Parlog.

Submission area :

Modelization and Simulation by Petri Nets.

Since their introduction in the early sixties the PN have been widely used in the specification and the validation of discret system events, as we can come across in the production systems or in the study of the communication protocols.

The PN are a powerful tool which describes and explains systems using concurrency, synchronization and resources shared. This capacity makes them a relevant object of analysis for parallel logic programming language.

We propose to show that a guarded language such as **Parlog** is particularly adapted for the modelization and the simulation of the PN. We will focus here on the simulation of colored PN.

1 Colored Petri Nets

Proposed by Carl Adams Petri [petri62], the PN have been the subject of numerous theoretical researches [brams83]. In this paper, we will adopt the notations and the terminology of David and Alla [david92].

The complexity of systems and the growth of the states number have promoted the appearance of some abbreviations, such as generalized PN, capacity PN, and colored PN, which make the subject of our paper.

We will focus on the study of the colored PN which enable the reduction of the graph (by merging all the same parts of PN) while keeping the clearness and the lisibility of modeled systems.

1.1 Definition

A colored PN is a an oriented bipartite graph, consisting of two nodes : places and transitions. These nodes are connected by directed arcs. An arc can connect either a place to a transition or a transition to a place.

In general, a place of a colored PN is represented by a circle, which contains any or several tokens. They can be either of the same color or of different colors; we associate a set of colors called validation colors to each transition. The colored PN arcs are labelled by functions, which enable the correspondence between the validation colors and the colors of tokens.

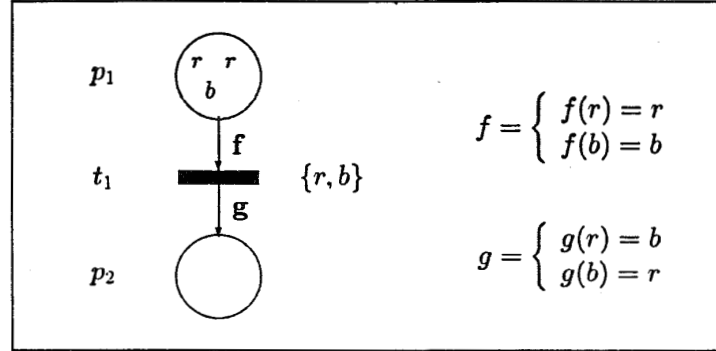


Figure 1 : Colored PN

Formaly, colored PN is a 6-tuple $S = \{P, T, Pre, Post, M_0, C\}$ where

1. $P = \{p_1, p_2, \dots, p_n\}$ finite set of places, $P \neq \{\}$,
2. $T = \{t_1, t_2, \dots, t_m\}$ finite set of transitions, $T \neq \{\}$,
3. $C = \{c_1, c_2, \dots, c_k\}$ finite set of colors, $C \neq \{\}$;
4. $P \cap T = \{\}$,
5. Pre : is the pre-function or input function defined on $P \times T \times C$ such as

$$Pre(p_i, t_j/c_k) = \begin{cases} p_i t_j / c_k & \text{if } p_i \rightarrow t_j \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

$p_i t_j / c_k$ represent the weight of the $p_i \rightarrow t_j$ arc; it is a linear combination of colors.

6. $Post$: is the post-function or output function defined on $P \times T \times C$ such as

$$Post(p_i, t_j/c_k) = \begin{cases} t_j p_i / c_k & \text{if } t_j \rightarrow p_i \text{ arc exists} \\ 0 & \text{otherwise} \end{cases}$$

$t_j p_i / c_k$ represent the weight of the $t_j \rightarrow p_i$ arc; it is a linear combination of colors.

7. M_0 : marking of colored PN.

To the former structural definition, we associate a marking, the movement of which translates the dynamics behavioral system, that we aim to modelize.

The places of colored PN represent states variables, which take for values the linear combination of colors. These values ($M(p_i)$) are represented by many tokens (colors) in the corresponding p_i place (figure 1).

1.2 Marking evolution

The marking of colored PN varies by firing transitions. A t_j transition of colored PN is enabled for $M(p_i)$ marking in relation to c_k color only if

$$\forall p_i \in P, M(p_i/c_k) \geq Pre(p_i, t_j/c_k)$$

where $M(p_i)$ is the total marking of the p_i place. So, the marking of the p_i place (figure 1) gives :

$$M = \begin{cases} M(p_1) = 2r + b \\ M(p_2) = 0 \end{cases}$$

After the t_j/c_k firing transition, the $M'(p_i)$ new marking is

$$\forall p_i \in P, M'(p_i) = M(p_i) - Pre(p_i, t_j/c_k) + Post(p_i, t_j/c_k)$$

In others words, a t_j transition is fired in relation to the c_k color (t_j/c_k) by removing $Pre(p_i, t_j/c_k)$ tokens from each of its input places and adding $Post(p_i, t_j/c_k)$ tokens to each of its output places. The firing (t_j/c_k) transition is illustrated below on figure 2 where t_1 is fired in relation to the r color (t_1/r).

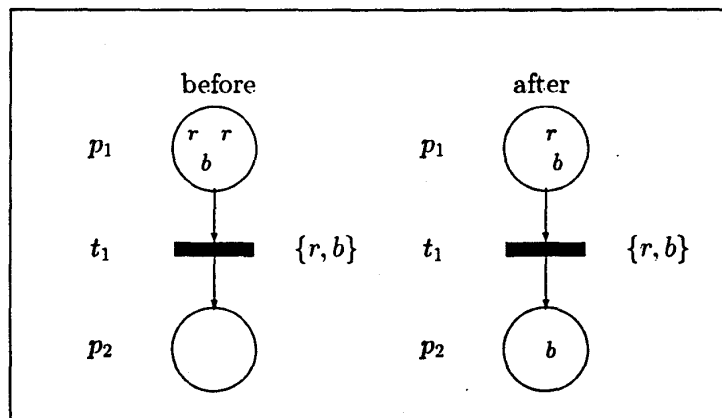


Figure 2 : A transition firing (t_1/r)

2 Parlog

The Parlog language [clark86] like Concurrent Prolog (CP) [ueda85] and Guarded Horn Clauses (GHC) [shapiro83] belong to the guarded languages family of the concurrent logic programming [chassin89].

A Parlog program is a finite set of guarded Horn clauses such as

$$Head \leftarrow Guard \mid Body$$

where

Head : the head of the clause,
Guard : the guard of the clause,
Body : the body of the clause,
| : symbol that separates the
guard from the body, is known
as the commit operator.

2.1 Semantics of Parlog

Like in Prolog, the declarative semantics of a Parlog guarded clause is interpreted as a conjunction clause :

$$Head \leftarrow Guard, Body$$

where the guard forms an integral part of the clause body. The reading logic of clause is :

If *Guard* and *Body* then *Head*

On the other hand, the operational semantics of a Parlog clause [rizk88] is fundamentally different from a Prolog clause [lloyd84] [apt82]. Actually, the goal resolution causes the simultaneous execution of as many processes as clauses in the definition of the goal (Or-parallelism). Each of these processes tries to unify the head of clause to the query and attempts to evaluate the guard of the clause. As soon as the guard of one of these processes has been passed, all the others processes are killed and only the body of the selected clause is evaluated (don't care non-determinism). At its turn, the evaluation of the body of selected clause leads to the concurrently execution of as many processes as subgoals in the definition of the goal (And-parallelism). Moreover, the specification of the mode declarations(input, output) of variables, enables the synchronisation of processes by means of the shared variables(stream communication). These different features (non-determinism, stream parallelism) make Parlog a well adapted language for simulation of the PN.

But, Prolog is a sequential logic programming language. Its research strategy (backtracking) enables to come back on the firing transition : this is fundamentally different of the operational semantics of PN.

2.2 Example

Let's consider a system of the following type Producer/Consumer (figure 3) to illustrate the essential notions [conlon89]. The producer generates a list of integer

numbers comprised in the interval [lower,upper]. The system may be coded in Parlog by two processes : generate/3, test/1.

```
mode generate(lower?,upper?,list^).
generate(Lower,Upper,[Lower|Rest]) <-
  Lower < Upper
  :
  Lower1 is Lower + 1,
  generate(Lower1,Upper,Rest).
generate(Upper,Upper,[Upper]).
```

```
mode test(list?).
test([T|Q]) <-
  T > 0,
  test(Q).
test([]).
```

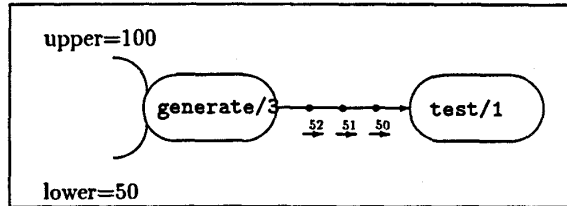


Figure 3 : Data stream

At the following query : `<- test(Integers), generate(50,100,Integers)`, the Parlog interpreter executes concurrently the two processes of the conjunction. They are synchronised by means of a `Integers` shared variable. As soon as an element of the list is instantiated, it is immediately sent to the consumer through the communication canal.

While the received value is treated by the consumer, the producer is involved in the determination of the following element : this is the stream parallelism in Parlog.

3 Modelization of colored PN

3.1 Places modelization

A place of a colored PN is modelized in Parlog by `place/4` predicate, so by a process, the task of which is

1. reorganizing the input stream of the place in just one stream by color (`sortInStream/2`) by
 - (a) bringing together all the streams of the same color in a Parlog list,
 - (b) merging all the streams of the same list in just one stream.
2. bringing together the output stream of the place according to their color (`sortOutputStream/2`),
3. transforming the initial marking for each color in tokens stream and merging it (`builtTokenStream/3`),
4. orienting the stream of tokens towards the output stream of the place (`identifyOutputStream/3`).

```
mode place(name?, in ?, out ?, marking ?).
place(Name, In, Out, Marking) <-
  sortInStream(In, NewIn),
  identifyOutputStream(Name, Token, NewOut),
  sortOutputStream(Out, NewOut),
  builtTokenStream(NewIn, Marking, Token).
```

where the `place/4` predicate arguments mean :

- `name` : the name of the place,
- `in` : the input tokens stream of the place,
- `out` : the output tokens stream of the place,
- `marking` : the initial marking of the place,

3.2 Transitions modelization

A transition of a colored PN is modeled by `transition/3` process, the task of which is

1. bringing together in a Parlog list all the input streams which have the same color (`reorganizingInStream/2`),
2. bringing together in a Parlog list all the output streams which have the same final nodes (`bringingOutStream/2`),
3. producing a tokens stream for each color (`mergingTokenStream/2`) by
 - (a) combining all the streams of the same color in just one stream by color,
 - (b) building a tokens stream from each resulting stream.
4. orienting the stream of tokens towards the output stream of the transition (`dispatchingTokenStream/3`).

```
mode transition(name ?, in ?, out ?).
transition(Name, In, Out) <-
  bringingOutStream(Out, NewOut),
  reorganizingInStream(In, NewIn),
  mergingTokenStream(NewIn, Token),
  dispatchingTokenStream(Name, Token, NewOut).
```

where the `transition/3` predicate arguments mean :

- `name` : the name of the transition
- `in` : the tokens input stream of the transition
- `out` : the tokens output stream of the transition.

3.3 Colored PN modelization

A colored PN is made up of a set of processes, representing the different nodes. During the simulation, they will be executed concurrently. The colored PN (figure 1) could be modeled by the `petriNet/3` predicate :

```
mode petriNet(type?, name ?, marking ?).
petriNet(color,figure1,[M1,M2]) <-
  place(p1,[],[t(t1,(r,P1t1r)),t(t1,(b,P1t1b))],M1),
  place(p2,[(r,T1p2r),(b,T1p2b)],[],M2),
  transition(t1,[(r,P1t1r),(b,P1t1b)],
             [(p2,(r,T1p2r)),t(p2,(b,T1p2b))]).
```

The three previous processes (`place/4` and `transition/3`) will be executed concurrently. they are synchronized by the following shared variables : `P1t1r`, `P1t1b`, `T1p2r` and `T1p2b`, which represent the tokens streams for each `r` and `b` colors. This process (`petriNet/3`) will be executed by the goal :

```
<- petriNet(color,figure1,[[r,2),(b,1)],[(r,0),(b,0)]).
```

4 Example of a colored PN simulation

4.1 Study of a simple manufacturing system

Let us consider an example of a simple manufacturing system, composed of two serial machining stations M_1 and M_2 . Each machine has respectively S_1 and S_2 stock and product respectively $pc1$ and $pc2$ kind of pieces (figure 4).

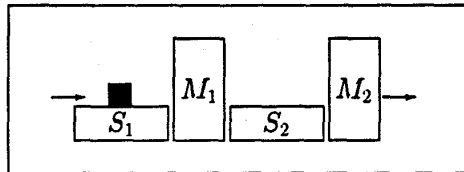


Figure 4 : A simple manufacturing system

We are placed in the following hypothesis :

1. In the production system, we can only process just one piece,
2. S_1 and S_2 stocks have an unlimited capacity,
3. Initial conditions :
 - (a) They are n products $pc1$ in S_1 stock,
 - (b) They are m products $pc2$ in S_1 stock,
 - (c) The M_1 machine waits for the $pc1$ product,
 - (d) The M_2 machine waits for the $pc1$ piece,
4. Obtaining $pc1, pc2, pc1, \dots, pc1, pc2, pc1$, in output system.

4.2 Modelization by a colored PN

Both machines and products have a similar behaviour; which justifies the colored PN use.

Let's suppose that the *pc1* product is the c_1 color and the *pc2* product is the c_2 color. So, we will get a color set $C = \{c_1, c_2\}$. The colored PN, modeling the manufacturing system is given in figure 5. The successor function (*succ/2*) is defined by

$$succ \begin{cases} succ(c_1) = c_2 \\ succ(c_2) = c_1 \end{cases}$$

This definition insures the pieces scheduling *pc1* and *pc2* : *pc1*, *pc2*, *pc1*, ..., *pc1*, *pc2*, *pc1*.

No labelled arcs are defined by the identity function (*id/2*)

$$id \begin{cases} id(c_1) = c_1 \\ id(c_2) = c_2 \end{cases}$$

4.3 Modelization of PN in Parlog

So, the representation of colored PN gives :

```
mode petriNet(type ?, name ?, marking ?).
petriNet(color,figure5,[P1,P2,P3,P4,P5,P6]) <-
  place(p1,[],[t(t1,(c1,P1t1c1)),t(t1,(c2,P1t1c2))],P1),
  place(p2,[(c1,T1p2c1),(c2,T1p2c2)],[t(t2,(c1,P2t2c1)),t(t2,(c2,P2t2c2))],P2),
  place(p3,[(c1,T2p3c1),(c2,T2p3c2)],[t(t3,(c1,P3t3c1)),t(t3,(c2,P3t3c2))],P3),
  place(p4,[(c1,T3p4c1),(c2,T3p4c2)],[t(t4,(c1,P4t4c1)),t(t4,(c2,P4t4c2))],P4),
  place(p5,[(c1,T2p5c1),(c2,T2p5c2)],[t(t1,(c1,P5t1c1)),t(t1,(c2,P5t1c2))],P5),
  place(p6,[(c1,T4p6c1),(c2,T4p6c2)],[t(t3,(c1,P6t3c1)),t(t3,(c2,P6t3c2))],P6),
  transition(t1,[(c1,P1t1c1),(c2,P1t1c2),(c1,P5t1c1),(c2,P5t1c2)],[t(p2,(c1,
    T1p2c1)),t(p2,(c2,T1p2c2))]),
  transition(t2,[(c1,P2t2c1),(c2,P2t2c2)],[t(p3,(c1,T2p3c1)),t(p3,(c2,T2p3c2)),
    t(p5,(c1,T2p5c1)),t(p5,(c2,T2p5c2))]),
  transition(t3,[(c1,P3t3c1),(c2,P3t3c2),(c1,P6t3c1),(c2,P6t3c2)],[t(p4,(c1,
    T3p4c1)),t(p4,(c2,T3p4c2))]),
  transition(t4,[(c1,P4t4c1),(c2,P4t4c2)],[t(p6,
    (c1,T4p6c1)),t(p6,(c2,T4p6c2)),
    t(out,(c1,T4outc1)),t(out,(c2,T4outc2))]).
```

The modelization of the corresponding functions is translated by the following predicate :

```
database function/4.
function(t1,p2,c1,c1).    function(t1,p2,c2,c2).    function(t2,p3,c1,c1).
function(t2,p3,c2,c2).    function(t2,p5,c1,c2).    function(t2,p5,c2,c1).
function(t3,p4,c1,c1).    function(t3,p4,c2,c2).    function(t4,p6,c1,c2).
function(t4,p6,c2,c1).    function(t4,out,c1,c1).    function(t4,out,c2,c2).
```

The colored PN simulation starts by the query of the Parlog interpreter :

```
<- petriNet(color,figure5,[[c1,10],[c2,10]],[c1,0],[c2,0],[c1,0],[c2,0],
  [[c1,0],[c2,0]],[c1,1],[c2,0]],[c1,1],[c2,0]]).
```

where, we have supposed that $n = m = 10$

pc1 : Output manufacturing system
 pc2 : Output manufacturing system
 pc1 : Output manufacturing system
 ...
 pc2 : Output manufacturing system

and is interrupted when there is no more product.

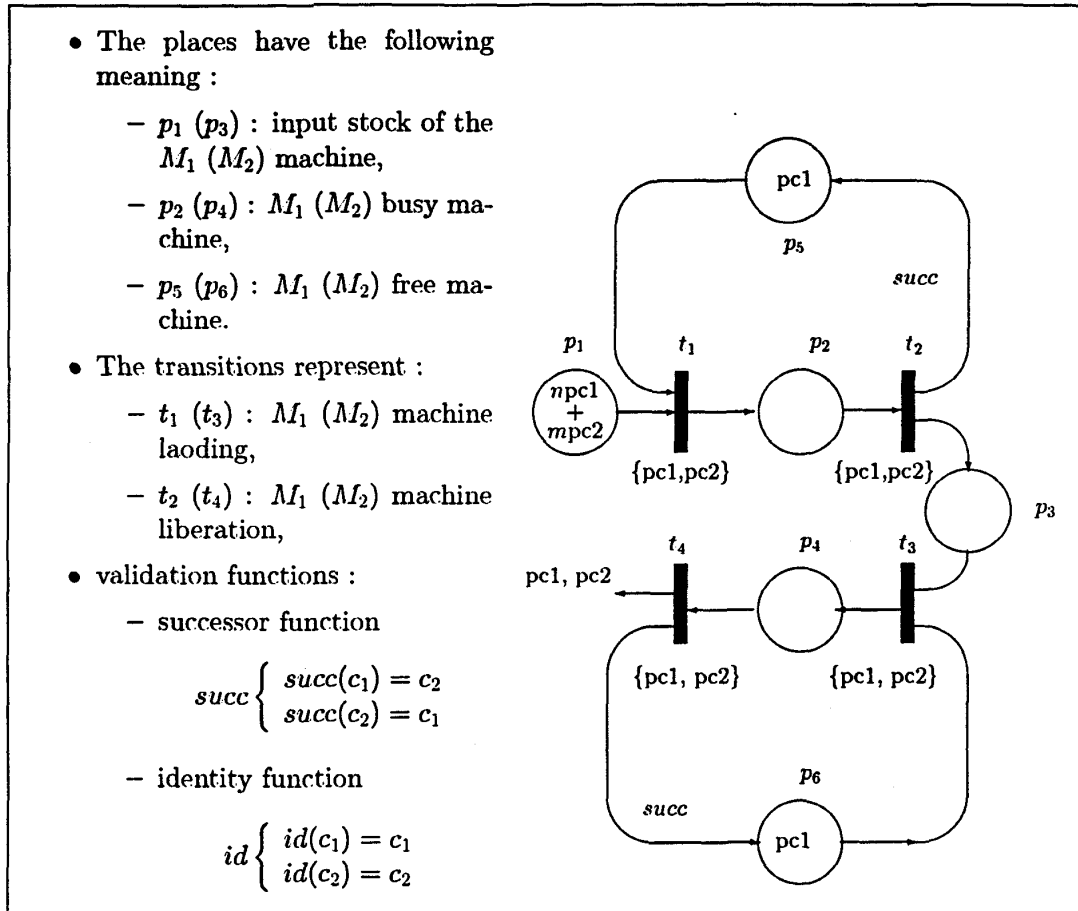


Figure 5 : Modelization of manufacturing system

5 Conclusion

The non-determinism (don't care non-determinism) of the concurrent logic programming illustrated here by the Parlog language enables to describe and to simply simulate a colored PN. A colored PN is seen as a set of processes, which communicate through a stream of tokens (stream parallelism). This approach could be spread to others PN abbreviations : generalized PN, capacity PN, as well as some PN extensions : synchronized PN and timed PN.

Références

- [alla82] Alla H, Ladet P., *Spécification et Commande des Ateliers Flexibles - Utilisation des Réseaux de Petri Colorés*. Ouvrage point en Productique, Vol. 1, Editions Lavoisier, 1982.
- [alla87] Alla H., *Réseaux de Petri Colorés et Réseaux de Petri Continus.*, Thèse d'Etat, Université Grenoble, 1987
- [apt82] Apt K. R., Vanemdem M. H., *Contribution to the theory of logic programming.*, Journal of the ACM29, 841-862, 1982
- [brams83] Brams G.W., *Réseaux de Petri : théorie et pratique.*, Masson Ed., Paris, 1983
- [clark86] Clark K., Gregory S., *Parlog : parallel programming in logic.*, ACM Transactions on Programming Languages and Systems, vol. 8, n°2, pp. 1-49, 1986
- [chassin89] Chassin de Kergommeaux J., Codognet Ph., Robert Ph., Syre J-C., *Une programmation logique parallèle : langages gardés.*, Technique et Sciences Informatiques, vol. 8, n°3, pp. 205-224, 1989
- [colom86] Colom, J.M. Silva, J.L. Villarroel(1986) : *On Software Implementation of Petri Nets and Colored Petri Nets Using High-Level Concurrent Languages". 7th European Workshop On Application and Theory of Petri Nets, Oxford, England, Juin 1986.*
- [conlon89] Conlon T., *Programming in Parlog*, Addison Wesley, 1989
- [david92] David R., Alla H., *Du Grafset aux réseaux de Petri.*, Hermes Ed., Paris, 1992
- [lloyd84] Lloyd J. W., *Foundation of the logic programming.*, Series in Symbolic Computation. Springer Verlag, 1984
- [petri62] Petri C.A., *Kommunikation mit Automaten.*, Westfälischen Institutes für Instrumentelle Mathematik, Bonn, 1962
- [rizk88] Richard G., Rizk A., *Semantics of the Concurrent Logic Programming Language Parlog.*, Rapport de recherche INRIA, n°848, mai 1988
- [shapiro83] Shapiro E.Y., *A subset of Concurrent Prolog and its interpreter.*, Technical report TR-003, ICOT, Tokyo, 1983
- [ueda85] Ueda K., *Guarded Horn Clauses.*, Technical report TR-103, ICOT, Tokyo, 1985