Deriving Analysable Petri Nets from Lotos Specifications

Michel Barbeau*, Gregor V. Bochmann
Université de Montréal, Département d'IRO
C.P. 128, Succ. "A", Montréal, Canada, H3C 3J7

January 10, 1990

Abstract

This paper proposes a Place/Transition-net (P/T-net) semantics for a subset of Lotos. The subset is such that finite structure nets can be obtained and it is therefore possible to apply P/T-nets verification techniques since they require finite structures. Moreover, the restrictions we put in order to obtain finite nets are in some sense "minimal" since we show that conversely P/T-nets can be simulated in our Lotos subset.

Topics: Relationships between net theory and other approaches, Verification using nets.

1 Introduction

The goal of this work is to investigate techniques for the verification of discrete event systems that contain dynamic creation and destruction of processes. We are mainly interested in the Lotos language [Bolognesi, Brinksma 1987]. We consider a data typeless Lotos subset that essentially comprises the choice, parallelism, sequential composition and disabling operators and process instantiation. The subset is identified more precisely in Section 3 where we also demonstrate that this language has the power of Turing machines. Consequently, no nontrivial property is decidable.

Our approach consists of reducing the power of the language by imposing easily verifiable syntactical constraints on the language elements. We are more particularly interested in deriving a Petri net, namely a Place/Transition-net (P/T-net), that is equivalent to a given Lotos description. The idea is to take advantage of the large number of verification techniques that have been developed for P/T-nets. P/T-nets are introduced in Section 2. How we obtain Petri nets from syntactically restricted Lotos specifications is described in Sections 4 and 5.

Various Petri net semantics for CCS [Degano et al. 1988], CSP [Olderog 1987] and Lotos [Marchena, Leon 1989] have been proposed. In general, these approaches generate unbounded numbers of places and transitions when recursion occurs inside parallel, sequential composition and

*The first author acknowledges financial support from the Natural Sciences and Engineering Research Council of Canada and the Centre de recherche informatique de Montréal.

disabling operators. It is therefore impossible to apply the techniques elaborated for P/T-nets since they require finite structure nets. Our approach is different in that we treat a restricted language. Recursion is limited inside parallel operators and it is possible to obtain finite structure nets, which means that the traditional P/T-net analysis methods are applicable. The problem of deriving finite nets has also been investigated by Goltz [1988] and Taubner [1989]. Taubner and Goltz consider subsets of CCS which are limited to two-way rendezvous. Our language supports multi-way rendezvous, the Lotos sequential composition and disabling operators. Moreover, Taubner's approach is completly different from ours since he uses the high level Predicate/Transition net model.

Another original feature in our approach is that we show that the restrictions we put in order to obtain finite structure P/T-nets are in some sense "minimal" since conversely P/T-nets can be simulated in our restricted language. This simulation is presented in Section 6. Other authors have proposed simulations of Petri nets in languages such as Prolog [Azema et al. 1984] or Meije [Boudol et al. 1985]. However their simulations are not in languages that where shown translatable to finite structure Petri nets. The relation of our work with other authors' work is further discussed in the conclusion, Section 7.

2 Petri nets

We slightly deviate from the usual notation for P/T-nets. We represent a P/T-net as triple (P, T, M_0) where

- P is a set of places,
- $T \subseteq 2^P \times Act \times 2^P$, is a transition relation,
- Act is a set of transition labels, and
- $M_0 \leq \mathcal{N}^P$ is the initial marking.

A P/T-net has a finite structure if the sets P, T and Act are finite.

Here, 2^P denotes the set of subsets of the set P whereas $\mathcal{N}^P$ denotes the set of multisets of the set P. An element $t = (X, a, Y) \in T$ will also be denoted as $X - a \rightarrow Y$. Its **preset** $pre(t)$ is X, its **postset** $post(t)$ is Y and **action** $act(t)$ is a. The operators $\leq$, $+$ and $-$ denotes respectively multiset inclusion, union and difference.

A transition $t \in T$ is **enabled** in marking M if $pre(t) \leq M$. This is denoted as $M[t >$. An enabled transition can be **fired** and the successor marking M' is defined as

$$M' = M - pre(t) + post(t)$$

this is indicated by $M[t > M'$ or $M - act(t) \rightarrow M'$.

For a P/T-net $N = (P, T, M_0)$, we define the **case graph of N** as the automaton

$$A(N) = (RS, T, M_0)$$

where $M_0 \in RS$ and for all $t \in T, M \in RS$

$$\textbf{if } M[t > M' \textbf{ then } M' \in RS \wedge (M, act(t), M') \in T.$$

3 Lotos

The version of Lotos under consideration is as follows. A Lotos behavior expression is formed out of the following terms:

Inaction	**stop**
Action prefix	$a; B$
Choice	$B_1 [] B_2$
Process instantiation	$p[g_1, ..., g_n]$
Pure interleaving	$B1 \vert\vert\vert B_2$
General parallel composition	$B_1 \vert[g_1, ..., g_n]\vert B_2$
Successful termination	**exit**
Sequential composition	$B_1 >> B_2$
Disabling	$B_1 [> B_2$
Hiding	**hide** $g_1, ..., g_n$ **in** B_1

where B, $B1$ and $B2$ are behavior expressions. The operational semantics of Lotos is given in [Bolognesi, Brinksma 1987]. An operational semantics uses an abstract machine to model the execution of programs. Lotos uses nondeterministic transition systems derivable from the initial behavior expression structure. More precisely, given an expression B, we associate the transition system $A = (S, T, B)$ where

- S is a set of states (behavior expressions),
- $B \in S$ is the initial state,
- $T \subseteq S \times Act \times S$ is transition relation, and
- Act denotes a set of user-defined actions together with the successful termination action δ and internal action i.

An element $(B, a, B') \in T$ is also denoted as $B - a \rightarrow B'$. The transition relation T is defined by means of a set of inference rules. More details can be found in the Reference.

Proposition 1 *The language under consideration has the power of Turing machines.*

Proof. A Turing machine is an abstract device that consists of a finite control box and an infinite read-write tape. We can show that this language has the power of Turing machines by simulating the behavior of an arbitrary Turing machine with three parallel Lotos processes. One process simulates the finite control and two stack processes simulate the read-write tape. The symbols to the left of the read-write head are stored on one stack and the symbols to the right are stored on the other stack [Hopcroft, Ullman 1979]. In [Gotzhein 1986] stack processes are modeled in a Lotos which includes data types. Since a Turing machine alphabet is finite it is possible to model a stack in a Lotos which does not include data types by defining one process `USED_STACK` for each possible tape symbol, similarly for the `push`, `pop` operations.

Since, this subset of Lotos has the power of Turing machines no nontrivial property is decidable.

P/T-nets (with finite structure) do not have the power of Turing machines as Lotos does. In the rest of this section we define a subset of Lotos, PLotos, from which P/T-nets can be derived and, conversely, into which P/T-nets can be simulated. The P/T-net semantics is introduced in the next section whereas P/T-nets simulation in PLotos is described in Section 6.

On the basic Lotos presented above we impose easy to verify syntactical constraints. They are formulated bellow and will be further justified in the sequel.

The main source of difficulty is recursion. A recursive process is a process which calls itself (directly or indirectly). The contraints on the above Lotos subset are:

1. Recursive process instantiations must be well-guarded. A process instantiation term is well-guarded if it is in the scope of a prefixing operator ";" or a right sub-term in a sequential composition ">>" or in a disabling "[>".

2. The general parallel operator "|[]|" may not occur in a recursive process body definition.

3. In the left sub-expression B_1, of a sequential composition $B_1 >> B_2$ or of a disabling $B_1 [> B_2$, instantiation of recursive processes is not allowed.

In Appendix A we express precisely the syntax of PLotos by means of the attribute grammar formalism.

Theorem 1 *PLotos has the power of P/T-nets. That is for every PLotos behavior specification we can derive an equivalent P/T-net. And, conversely, for every P/T-net we can construct an equivalent PLotos behavior specification.*

Proof. In Sections 4 and 5 we shown how to translate to a P/T-net a PLotos specification. The converse is true as well and will be demonstrated in Section 6.

4 P/T-net Semantics

4.1 General Idea

In general a behavior-expression B represents the composition of a set of concurrent activities. The idea is to decompose the expression B in order to obtain an explicit representation of this set of parallel activities. The representation of each activity also contains information on its dependencies with respect to other activities. For instance, the Lotos expression $a; b; p|[b]|b; stop$ represents two concurrent activities. The first activity executes the actions a and b, and next instantiates process p. The second activity performs the action b and stops. Both activities are coupled on gate b and are therefore dependent on each other with respect to the execution of action b. The set of parallel components resulting from the decomposition of behavior $a; b; p|[b]|b; stop$ is denoted as $\{a; b; p|[b]||, ||[b]|b; stop\}$. In this representation, we denote explicitly the fact that the activities are coupled on gate b by concatenating the symbol $||[b]||$ to the right of $a; b; p$ and to the left of $b; stop$.

In the Petri net representation, place names correspond to names of parallel activities. The Petri net for the example above will include places named $a; b; p|[b]||$ and $||[b]|b; stop$. The presence of a token in place named X means that a component with behavior X is active in the given system state. Several tokens in the same place represent several identical parallel activities. This models unbounded parallelism with finite structure nets.

Set of parallel activities will denote P/T-net markings. For instance, a transition from the parallel activities above will be:

$$\{a; b; p|[b]||, ||[b]|b; stop\} - a \rightarrow \{b; p|[b]||, ||[b]|b; stop\}$$

We first introduce the decomposition function in Section 4.2, then we present in Section 4.3 an operational and non-constructive Petri net semantics for PLotos. Based on this semantics, in Section 5 an algorithm is presented which can be used to construct the P/T-nets. The algorithm can be seen as a deductive system implementing the inference rules.

4.2 Decomposition Function

The decomposition function is denoted as *dec*. Its domain is the set of well-formed PLotos behavior-expressions. Its range is the set of all possible multiset of place names. A place name is either a *constant symbol* (stop or exit) or a *symbolic expression* constructed with a prefix function symbol (; or hide $g_1, ..., g_n$ in), a prefix postfix function symbol ($|||$ or $||[g1, ..., gn]||$) or an infix function symbol ([] and $>>$). Expressions constructed with function symbols may contain process instantiation sub-terms $p[g_1, ..., g_n]$.

Let B_1, B_2 denote syntactically correct PLotos behavior expressions, a denote an action name

and $S = g_1, ..., g_n$ a list of synchronization gates,

$$
\begin{array}{lrl}
(d1) & dec(stop) & = \{\} \\
(d2) & dec(a; B_1) & = \{a; B_1\} \\
(d3) & dec(B_1 [] B_2) & = \{B_1 [] B_2\} \\
(d4) & dec(p[g_1, ..., g_n]) & = dec(B_p[g_1/h_1, ..., g_n/h_n]) \\
(d5) & dec(B_1 ||| B_2) & = dec(B_1) + dec(B_2) \\
(d6) & dec(B_1 |[S]| B_2) & = dec(B_1).|[S]| + |[S]|.dec(B_2) \\
(d7) & dec(B_1 >> B_2) & = \{B_1 >> B_2\} \\
(d8) & dec(B_1 [> B_2) & = \{B_1 [> B_2\} \\
(d9) & dec(hide\ S\ in\ B_1) & = hide\ S\ in.dec(B_1)
\end{array}
$$

where

- B_p represents the body of the definition of process p,
- in (d4), $g_1, ..., g_n$ is a list of formal gates,
- $h_1, ..., h_n$ is a list of actual gates,
- $[g_1/h_1, ..., g_n/h_n]$ is the relabeling postfix operator, gate g_i becomes gate h_i $(i = 1, ..., n)$, and
- the expression $dec(B_1).|[S]|$ denotes $\{x|[S]| : x \in dec(B_1)\}$, similarly for $|[S]|.dec(B_2)$ and $hide\ S\ in.dec(B_1)$.

The *dec* function is deterministic taking into account operator precendences. The restriction to guarded recursive processes is necessary to stop recursion in the *dec* function.

4.3 Inference Rules

This section presents the heart of our P/T-net semantics. The P/T-net $N = (P, T, M_0)$ associated to a PLotos behavior B is such that:

1. $M_0 = dec(B) \subseteq P$,
2. if $X \subseteq P$ and $X - a \rightarrow Y$ then $Y \subseteq P$ and $(X, a, Y) \in T$, and
3. only the elements that can be obtained from items 1 or 2 are in P and T.

The transition instances are inferred from the rules bellow.

For all PLotos expressions B_1, B_1', B_2, B_2', action name a, list $S = g_1, ..., g_n$ of synchronization gates and subsets of places M_1, M_2, M_1', M_2':

(r1) $\{a; B_1\} - a \rightarrow dec(B_1)$

(r2) if $B_1 - a \rightarrow B_1'$
then $\{B_1 [] B_2\} - a \rightarrow dec(B_1')$

(r3) if $B_2 - a \rightarrow B_2'$
then $\{B_1 [] B_2\} - a \rightarrow dec(B_2')$

(r4) if $M_1 - a \rightarrow M_1'$ and $a \notin \{S, \delta\}$
then $M_1.|[S]| - a \rightarrow M_1'.|[S]|$

(r5) if $M_2 - a \rightarrow M_2'$ and $a \notin \{S, \delta\}$
then $|[S]|.M_2 - a \rightarrow |[S]|.M_2'$

(r6) if $M_1 - a \rightarrow M_1'$ and $M_2 - a \rightarrow M_2'$ and $a \in \{S, \delta\}$
then $M_1.|[S]| + |[S]|.M_2 - a \rightarrow M_1'.|[S]| + |[S]|.M_2'$

(r7) if $B_1 - a \rightarrow B_1'$ and $a \neq \delta$
then $\{B_1 >> B_2\} - a \rightarrow \{B_1' >> B_2\}$

(r8) if $B_1 - \delta \rightarrow B_1'$
then $\{B_1 >> B_2\} - \delta \rightarrow dec(B_2)$

(r9) if $B_1 - a \rightarrow B_1'$ and $a \neq \delta$
then $\{B_1 [> B_2\} - a \rightarrow \{B_1' [> B_2\}$

(r10) if $B_1 - \delta \rightarrow B_1'$
then $\{B_1 [> B_2\} - \delta \rightarrow \{\}$

(r11) if $B_2 - a \rightarrow B_2'$
then $\{B_1 [> B_2\} - a \rightarrow dec(B_2')$

(r12) if $M_1 - a \rightarrow M_1'$ and $a \notin \{S\}$
then *hide* S *in*.$M_1 - a \rightarrow$ *hide* S *in*.M_1'

(r13) if $M_1 - a \rightarrow M_1'$ and $a \in \{S\}$
then *hide* S *in*.$M_1 - i \rightarrow$ *hide* S *in*.M_1'

In the "if part" of inference rules (r2), (r3), (r7) (r8), (r9), (r10) and (r11) behavior B_1 (B_2) goes to behavior B_1' (B_2') on action a or δ according to the original Lotos semantics in [Bolognesi, Brinksma 1987].

Theorem 2 *The Petri net semantics of Lotos is consistent with respect to the original Lotos semantics. That is, for all PLotos behavior expressions* B, B' *and action* a*:*

$$B - a \rightarrow B' \Longleftrightarrow dec(B) - a \rightarrow dec(B')$$

Proof. The proof technique is by induction on the number of operators in a behavior expression B.

($\Rightarrow$). We must show that for all B, B', a:

$$B - a \rightarrow B' \Rightarrow dec(B) - a \rightarrow dec(B')$$

Basis (Zero operators) The expression B must be *stop* or *exit*. In the first case no transition is possible in both semantics. In the second case functionality of behavior *exit* is *exit* (see Appendix A for the meaning of the functionality attribute). This is necessarily the left sub-expression of a sequential composition and this is handled by case 5 bellow.

Induction (One or more operators) Assume that the theorem is true for behavior expressions with fewer than i operators, $i \geq 1$. Let B have i operators. There are seven cases depending on the form of B.

CASE 1 $B = a; B'$. The conclusion is immediate from (d2) and (r1).

CASE 2 $B = B_1 [] B_2$. In that case the conclusion is immediate because (r2) and (r3) are defined in terms of the original semantics.

CASE 3 $B = B_1 ||| B_2$. Both B_1 and B_2 must have fewer than i operators. According to the original semantics,

if $B_1 - a \rightarrow B_1'$ and $a \neq \delta$
  then $B_1 ||| B_2 - a \rightarrow B_1' ||| B_2$,
if $B_2 - a \rightarrow B_2'$ and $a \neq \delta$
  then $B_1 ||| B_2 - a \rightarrow B_1 ||| B_2'$,
if $B_1 - \delta \rightarrow B_1'$ and $B_2 - \delta \rightarrow B_2'$
  then $B_1 ||| B_2 - a \rightarrow B_1' ||| B_2'$.

First, suppose that sub-expressions B_1 and B_2 have functionality *noexit*, in that case the third rule does not apply. We will treat only the first rule, since the second rule is similar. By the induction hypothesis:

$$B_1 - a \rightarrow B_1' \Rightarrow dec(B_1) - a \rightarrow dec(B_1)$$

Moreover, from (d5)

$$dec(B_1) \leq dec(B_1) + dec(B_2) = dec(B_1 ||| B_2)$$

therefore

$$dec(B_1 ||| B_2) - a \rightarrow dec(B_1' ||| B_2).$$

If sub-expression B_1 and B_2 have functionality *exit*, the expression $B_1 ||| B_2$ is the left sub-expression of a sequential composition and this is handled by case 5 bellow.

CASE 4 $B = B_1 |[S]| B_2$. According to the original semantics,

if $B_1 - a \rightarrow B_1'$ and $a \notin \{S, \delta\}$
  then $B_1 |[S]| B_2 - a \rightarrow B_1' |[S]| B_2$,
if $B_2 - a \rightarrow B_2'$ and $a \notin \{S, \delta\}$
  then $B_1 |[S]| B_2 - a \rightarrow B_1 |[S]| B_2'$,
if $B_1 - a \rightarrow B_1'$ and $B_2 - a \rightarrow B_2'$ and $a \in \{S, \delta\}$
  then $B_1 |[S]| B_2 - a \rightarrow B_1' |[S]| B_2'$,

Let us consider the first rule. By the induction hypothesis:

$$B_1 - a \rightarrow B_1' \Rightarrow dec(B_1) - a \rightarrow dec(B_1')$$

If we substitute $dec(B_1)$, $dec(B_1')$ to respectively M_1, M_1' in (r4) we obtain:

$$dec(B_1).|[S]| - a \rightarrow dec(B_1').|[S]|$$

Futhermore, from (d6):

$$dec(B_1).|[S]| \leq dec(B_1).|[S]| + |[S]|.dec(B_2) = dec(B_1|[S]|B_2)$$

therefore,

$$dec(B_1|[S]|B_2) - a \rightarrow dec(B_1'|[S]|B_2).$$

The second rule is similar to the first one whereas the third rule is not much more difficult to prove.

CASE 5 $B = B_1 >> B_2$. In that case the conclusion is immediate because (r7) and (r8) are defined in terms of the original semantics of Lotos.

CASE 6 $B = B_1[> B_2$. The conclusion is immediate because (r9), (r10) and (r11) are defined in terms of the original Lotos semantics. Note that in (r10) the postset is $\{\}$ if we admit the equivalence $stop|[S]|stop = stop$ and consider the following easy to verify property of PLotos:

$$(\forall B, B')[B - \delta \rightarrow B' \Rightarrow B' = stop]$$

CASE 7 $B = hide\ S\ in\ B_1$. The original semantics inference rules for the hiding operator are:

if $B_1 - a \rightarrow B_1'$ and $a \notin \{S\}$
  then $hide\ S\ in\ B_1 - a \rightarrow hide\ S\ in\ B_1'$
if $B_1 - a \rightarrow B_1'$ and $a \in \{S\}$
  then $hide\ S\ in\ B_1 - i \rightarrow hide\ S\ in\ B_1'$

By the induction hypothesis:

$$B_1 - a \rightarrow B_1' \Rightarrow dec(B_1) - a \rightarrow dec(B_1')$$

If we substitute $dec(B_1)$, $dec(B_1')$ to respectively M_1, M_1' in (r12),(r13) and from (d9) we obtain:

if $a \notin \{S\}$ then $dec(hide\ S\ in\ B_1) - a \rightarrow dec(hide\ S\ in\ B_1')$
if $a \in \{S\}$ then $dec(hide\ S\ in\ B_1) - i \rightarrow dec(hide\ S\ in\ B_1')$

($\Leftarrow$). We must show that for all B, B', a:

$$dec(B) - a \rightarrow dec(B') \Rightarrow B - a \rightarrow B'$$

In this part we must assume the following equivalence laws from [ISO 1988]:

$$\begin{aligned} B_1|||B_2 &= B_2|||B_1 \\ B_1|||(B_2|||B_3) &= (B_1|||B_2)|||B_3 \end{aligned}$$

The proof is similar to above.

Remark 1 *A thing has to be remarked, let us consider the following behavior:*

$$B = (stop|[a]|a; stop)||||(a; stop|[a]|stop)$$

According to the original Lotos semantics no action is possible from this behavior. However as it is, the P/T-net semantics can infer the action a because:

$$dec(B) = \{stop|[a]|, |[a]|a; stop, a; stop|[a]|, |[a]|stop\}$$

Rule (r6) can be applied with

$$M_1 = \{stop, a; stop\} \text{ and } M_2 = \{a; stop, stop\}$$

to deduce action a. To avoid such wrong reasoning, $|[S]|$ operators can be distinguished with distinct labels (e.g. $|[S]|_k$). Since such expressions cannot appear in recursive processes each $|[S]|_k$ can appear at most once in a given global state.

Definition 1 *Two graphs $G_1 = (V_1, E_1, n_1)$ and $G_2 = (V_2, E_2, n_2)$ are* **isomorphic** *if there is a one-to-one onto function $f : V_1 \rightarrow V_2$ such that $f(n_1) = n_2$ and*

$$(v, a, v') \in V_1 \Leftrightarrow (f(v), a, f(v')) \in V_2.$$

An immediate consequence of Theorem 2 is the following.

Corollary 1 *Let the P/T-net $N = (P, T, M_0)$ associated to a PLotos behavior B, $A(N) = (RS, T, M_0)$ its associated case graph and the transition system $A = (S, T, B)$ associated to B from the original semantics of Lotos. The two graphs A and $A(N)$ are isomorphic with the function dec.*

5 Derivation Algorithm

In this section we define a PLotos to P/T-nets procedure, *Lopep*, which constructs the net according to the semantics of section 4. The P/T-net modeling the PLotos behavior B, denoted as $Lopep(B, Env, k)$, is obtained by computing the transitive closure of the transition relation T,

starting from the initial marking $dec(B)$. That leads naturally to a recursive procedure. The second parameter, Env, is a set of process instantiation terms that have been considered so far in the derivation. It is initially empty and used to stop recursion in the Lopep procedure. The third parameter k is used to assign distinct labels to $|[S]|$ operators. Its initial value is zero.

procedure Lopep(B:Expression, Env:Set, **var** k:integer):P/T-net
case B **of**
$stop :\rightarrow (1)$
$a; B_1 :\rightarrow (2)$
$B_1 [] B_2 :\rightarrow (3)$
$p[g_1, ..., g_n] :\rightarrow (4)$
$B_1 ||| B_2 :\rightarrow (5)$
$B_1 |[S]| B_2 :\rightarrow (6)$
$B_1 >> B_2 :\rightarrow (7)$
$B_1 [> B_2 :\rightarrow (8)$
$hide\ S\ in\ B_1 :\rightarrow (9)$
end;

(1) Nothing can be derived from stop.

return $(\{\}, \{\}, \{\})$

(2) Inference rule (r1) says that from $dec(a; B_1)$, a transition t labelled with action a and with postset $dec(B_1)$ can be inferred. By means of a recursive call to *Lopep* we obtain the net modeling the successor behavior B_1.

$(P, T, M) \leftarrow Lopep(B_1, Env, k)$
return $(\{a; B_1\} \cup P, \{(\{a; B_1\}, a, M)\} \cup T, \{a; B_1\})$

(3) In that case rules (r2) and (r3) apply. In P and T we put respectively the places and the transitions derivable from $\{B_1 [] B_2\}$.

$P \leftarrow \{\}, T \leftarrow \{\}$
forall a, B' **suchthat** $B_1 - a \rightarrow B'$ **or** $B_2 - a \rightarrow B'$ **do**
$(P', T', M) \leftarrow$ Lopep(B',Env,k)
$P \leftarrow P \cup P'$
$T \leftarrow T \cup \{(\{B_1 [] B_2\}, a, M)\} \cup T'$
return $(P, T, \{B_1 [] B_2\})$

(4) The net denoted by a process instantiation is constructed only if the process instantiation term has not been considered so far in the derivation, the condition $p[g_1, ..., g_n] \in Env$ is false. In that case solely the initial marking is returned. This prevents unbounded recursive calls to Lopep. Otherwise, Lopep is called recursively on the body definition B_p of process p.

if $p[g_1, ..., g_n] \in Env$
then return $(\{\}, \{\}, dec(B_p[g_1/h_1, ..., g_n/h_n]))$
else return $Lopep(B_p[g_1/h_1, ..., g_n/h_n], Env \cup \{p[g_1, ..., g_n]\}, k)$

(5) For pure interleaving, we combine the nets obtained from $dec(B_1)$ and $dec(B_2)$.

$(P_1, T_1, M_1) \leftarrow Lopep(B_1, Env, k)$
$(P_2, T_2, M_2) \leftarrow Lopep(B_2, Env, k)$
return $(P_1 \cup P_2, T_1 \cup T_2, M_1 + M_2)$

(6) For the general parallel operator case, we first obtain the nets denoted by B_1 and B_2 and we merge the transitions inferred from rules (r4), (r5) and (r6).

$k \leftarrow k + 1$
$(P_1, T_1, M_1) \leftarrow Lopep(B_1, Env, k)$
$(P_2, T_2, M_2) \leftarrow Lopep(B_2, Env, k)$
(* Transitions from (r4) *)
$T_1' \leftarrow \{(M_1.|[S]|_k, a, M_1'.|[S]|_k) : (M_1, a, M_1') \in T_1 \wedge a \notin \{S, \delta\}\}$
(* Transitions from (r5) *)
$T_2' \leftarrow \{(|[S]|_k.M_2, a, |[S]|_k.M_2') : (M_2, a, M_2') \in T_2 \wedge a \notin \{S, \delta\}\}$
(* Transitions from (r6) *)
$T' \leftarrow \{(M_1.|[S]|_k + |[S]|_k.M_2, a, M_1'.|[S]|_k + |[S]|_k.M_2') :$
$\quad (M_1, a, M_1') \in T_1 \wedge (M_2, a, M_2') \in T_2 \wedge a \in \{S, \delta\}\}$
return $(P_1.|[S]|_i \cup |[S]|_i.P_2, T_1' \cup T_2' \cup T', dec(B_1).|[S]|_i + |[S]|_i.dec(B_2))$

(7) Recursion is not allowed in sub-expression B_1. It has been indicated before in [Bolognesi, Smolka 1987] that such a behavior can be mapped to a finite transition system. Let (S, T, B_1) denote this transition system:

$P_1 \leftarrow \{s >> B_2 : s \in S\}$
$T_1 \leftarrow \{(\{s >> B_2\}, a, \{s' >> B_2\}) : (s, a, s') \in T \wedge a \neq \delta\} \cup$ (* r7 *)
$\quad \{(\{s >> B_2\}, \delta, dec(B_2)) : (s, \delta, s') \in T\}$ (* r8 *)
$(P_2, T_2, M_2) \leftarrow Lopep(B_2, Env, k)$
return $(P_1 \cup P_2, T_1 \cup T_2, \{B_1 >> B_2\})$

(8) As in (7), recursion is not allowed in B_1. Let (S, T, B_1) denote this transition system modeling B_1:

$P_1 \leftarrow \{s[> B_2 : s \in S\}$
$T_1 \leftarrow \{(\{s[> B_2\}, a, \{s'[> B_2\}) : (s, a, s') \in T \wedge a \neq \delta\} \cup$ (* r9 *)
$\{(\{s[> B_2\}, \delta, \{\}) : (s, \delta, s') \in T\} \cup$ (* r10 *)
$\{(\{s[> B_2\}, \delta, dec(B_2')) : s[> B_2 \in P_1 \wedge B_2 - a \rightarrow B_2'\}$ (* r11 *)
$(P_2, T_2, M_2) \leftarrow Lopep(B_2, Env, k)$
return $(P_1 \cup P_2, T_1 \cup T_2, \{B_1[> B_2\})$

(9) Inference rules (r12) and (r13) apply. Transitions labelled with action in $\{S\}$ are relabelled with i:

$(P, T, M) \leftarrow Lopep(B_1, Env, k)$
$T_1 \leftarrow \{(hide\ S\ in.M_1, a, hide\ S\ in.M_1') : (M_1, a, M_1') \in T \wedge a \notin \{S\}\}$
$T_2 \leftarrow \{(hide\ S\ in.M_1, i, hide\ S\ in.M_1') : (M_1, a, M_1') \in T \wedge a \in \{S\}\}$
return $(hide\ S\ in.P, T_1 \cup T_2, hide\ S\ in.M)$

We will illustrate the result of Lopep with an example:

specification $p1[a, b, c] :=$
$p2[a, b] || [a] | p2[a, c]$
where
process $p2[x, y] :=$
$x; exit >> (y; stop ||| p2[x, y])$
endproc
endspec

Note that all constraints mentioned in Section 3 are satisfied. The P/T-net is obtained by calling $Lopep(p[a, b, c], \{\}, 0)$ is illustrated in Fig. 1. In the graphical representation, places are shown as circles, transitions as bars and tokens as dots inside places.

Note that the following example is not in PLotos and, as shown in [Goltz 1988], cannot be represented by a finite structure net:

process $p[a, b] :=$
$a; b; (p[a, b] || [b] | p[a, b])$
endproc

The problem is that after each recursive call, the number of activities to be synchronized on action b grows by one. To be represented, it would require an arbitrarily large number of P/T-net transitions.

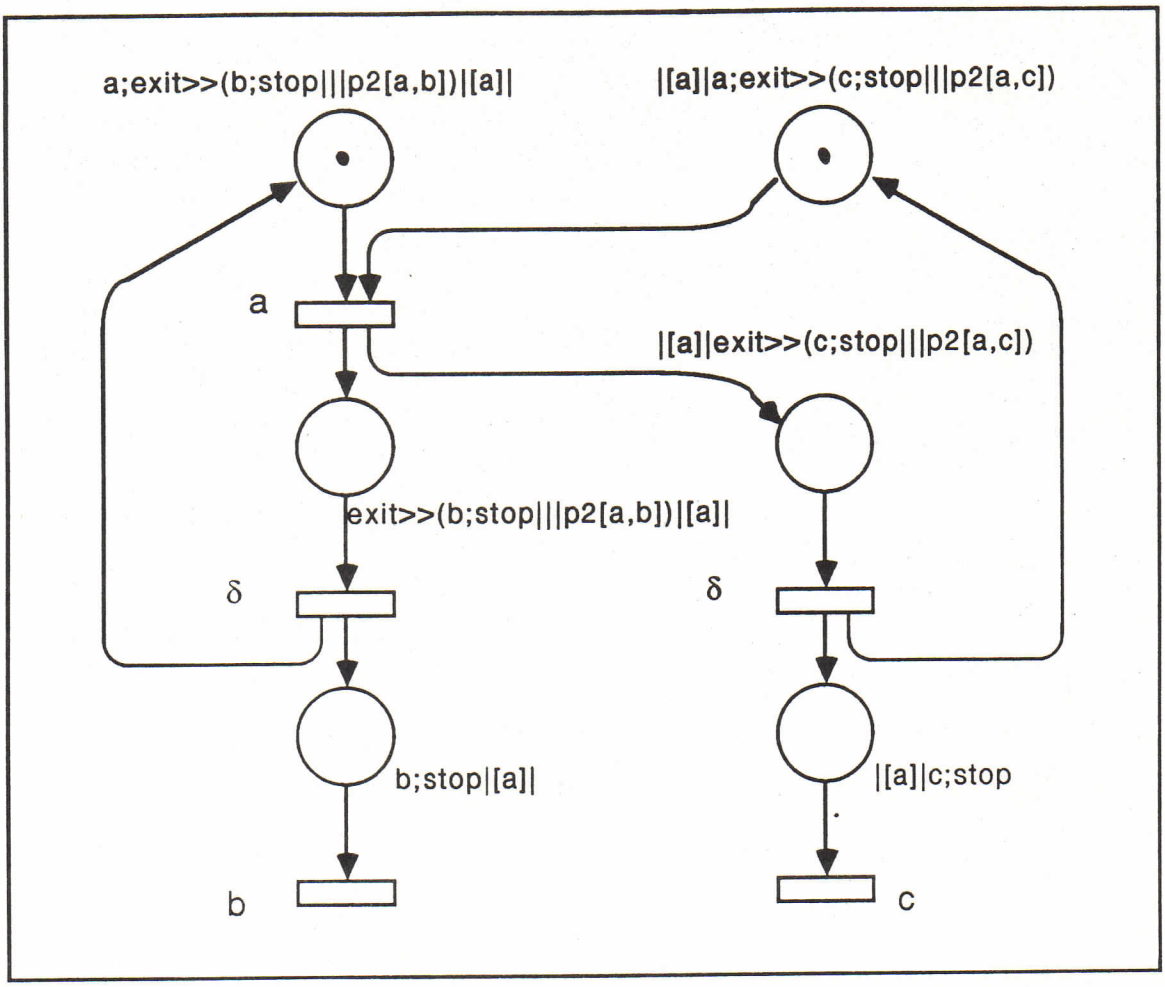


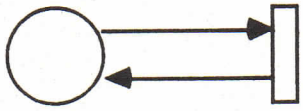
Figure 1: Example

The problem is similar in a sequential composition $B_1 >> B_2$. With recursion combined with parallelism in the sub-expression B_1, the number of processes to be synchronized on the successful termination action δ is dynamic and cannot a priori be determined. In a disabling $B_1[> B_2$, with recursion and parallelism in B_1 the number of processes to be synchronized on δ and any action performed by B_2 is also unbounded.

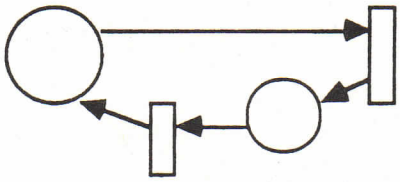
6 Simulation of P/T-nets in PLotos

In this section we complete the proof that PLotos has the power of P/T-nets. That is for every P/T-net we can construct an equivalent PLotos behavior specification.

In the Lotos simulation of P/T-nets, a token is a Lotos process. We make the following hypothesis: No place is simultaneously in the preset and the postset of a given transition. This restriction is not significant since a sub-graph such as:



Can be simulated by the sub-graph:



6.1 Modeling of Places

Let $N = (P, T, M)$ be a P/T-net with $P = \{p_1, ..., p_n\}$ and $T = \{t_1, ..., t_m\}$. We first discuss how tokens into places are represented by Lotos processes.

Given place $p_i \in P$, let

- $\Gamma^{-1}(p_i) = \{t_j : p_i \in post(t_j)\}$ (transitions that deposit tokens into place p_i),
- $\Gamma(p_i) = \{t_j : p_i \in pre(t_j)\}$ (transitions that take tokens from place p_i), and
- $T(X) = \bigcup_{p_i \in X}(\Gamma(p_i) \cup \Gamma^{-1}(p_i))$ (transitions connected to places in X).

A token inside place p_i can participate in the firing of a transition in $\Gamma(p_i)$.

Definition 2 *Let $\Gamma(p_i) = \{t_{o1}, ..., t_{ok}\}$, a token inside place p_i is represented as the following Lotos process.*

process $token_i[t_{o1}, ..., t_{ok}]$:=

t_{o1}; *stop*[]...[]t_{ok}; *stop*

endproc

Let $\Gamma^{-1}(p_i) = \{t_{i1}, ..., t_{ij}\}$, *the place* p_i *is modeled by the following process:*

process $p_i[t_{i1}, ..., t_{ij}, t_{o1}, ..., t_{ok}]$:=

$(t_{i1}; exit[]...[]t_{ij}; exit) >> (token_i[t_{o1}, ..., t_{ok}]|||p_i[t_{i1}, ..., t_{ij}, t_{o1}, ..., t_{ok}])$

endproc

Note that there is no recursion in the left sub-expression of operator >> and the recursive call to p_i is guarded since it is in the right sub-expression of a sequential composition. Informally, this says that when a place p_i input transition is fired, either t_{i1} or ... or t_{ij}, then a new token is deposited into place p_i (an instance of process $token_i$ is created). This newly created token can now contribute to enable and fire a transition in $\Gamma(p_i)$.

The next lemma demontrates the consistency of the PLotos model of a place.

Lemma 1 *Let, for* $k \in \mathcal{N}$, $P_i(k)$ *denotes the place* p_i *containing* k *tokens, defined as:*

$$\begin{aligned} P_i(0) &= p_i[t_{i1}, ..., t_{ij}, t_{o1}, ..., t_{ok}] \\ P_i(k) &= token_i|||P_i(k-1) \end{aligned}$$

For all $k \in \mathcal{N}$,

$$P_i(k) - t \rightarrow p \Leftrightarrow [(t \in \Gamma^{-1}(p_i) \wedge p = P_i(k+1)) \vee (k > 0 \wedge t \in \Gamma(p_i) \wedge p = P_i(k-1))].$$

Proof. The proof is by induction on k.

($\Rightarrow$). *Basis* Let $k = 0$, $P_i(k) = p_i[t_{i1}, ..., t_{ij}, t_{o1}, ..., t_{ok}]$ and according to definition 2:

$$p_i - t \rightarrow p \Rightarrow (t \in \Gamma^{-1}(p_i) \wedge p = token_i|||p_i = P_i(1))$$

Induction Let $k \geq 0$, assume that the lemma is true whenever $n \leq k$. By definition

$$P_i(k+1) = token_i|||P_i(k),$$

and according to the induction hypothesis:

$$\begin{aligned} token_i|||P_i(k) - t \rightarrow p \Rightarrow [\ & (t \in \Gamma^{-1}(p_i) \wedge p = token_i|||P_i(k+1)) \vee \\ & (t \in \Gamma(p_i) \wedge k > 0 \wedge p = token_i|||P_i(k-1)) \vee \\ & (t \in \Gamma(p_i) \wedge k \geq 0 \wedge p = P_i(k))] \end{aligned}$$

This implies that

$$P_i(k+1) - t \rightarrow p \Rightarrow [(t \in \Gamma^{-1}(p_i) \wedge p = P_i(k+2)) \vee (t \in \Gamma(p_i) \wedge p = P_i(k))]$$

($\Leftarrow$). Similar.

6.2 Modeling of P/T-nets

The model of a P/T-net in Lotos is also defined inductively. For $1 \leq i \leq n$, we denote by

$$N_{1,i} = (P_{1,i}, T_{1,i}, M_{1,i})$$

the subnet of $N = (P, T, M)$ with

- $P_{1,i} = \{p_1, ..., p_i\}$,
- $T_{1,i} = \{(X, act(t), Y) : (t \in T) \wedge (X = pre(t) \cap P_{1,i}) \wedge (Y = post(t) \cap P_{1,i}) \wedge (X \neq \{\} \vee Y \neq \{\})\}$,
- $M_{1,i}$, the marking M restricted to places in $P_{1,i}$.

Note that $N_{1,n} = N$. We denote by $M_{1,i}(i)$ the number of tokens inside place p_i for the marking $M_{1,i}$.

Definition 3 *For $1 \leq i \leq n$, the subnet $N_{1,i}$ is modeled by a PLotos process named $N_{1,i}(M_{1,i})$ defined as:*

process $N_{1,1}(M_{1,1})[t_1, ..., t_m] :=$
$P_1(M_{1,1}(1))$
endproc

For $i > 1$, $N_{1,i}(M_{1,i})$ is defined as:

process $N_{1,i}(M_{1,i})[t_1, ..., t_m] :=$
$P_i(M_{1,i}(i))|[T(\{p_i\}) \cap T(\{p_1, ..., p_i\})]|N_{1,i-1}(M_{1,i-1})[t_1, ..., t_m]$
endproc

Note that, for $i = 1, ..., n$, $N_{1,i}(M_{1,i})$ is not recursive.

The model of a P/T-net N in PLotos is the process $N_{1,n}(M_{1,n})$. The next lemma demonstrates the consistency of the PLotos model of P/T-nets.

Lemma 2 *Let $N = (P, T, M)$ a P/T-net and $M' \in \mathcal{N}^P$ a marking. For every $i = 1, ..., n$, let $N_{1,i} = (P_{1,i}, T_{1,i}, M_{1,i})$ the subnets defined as above, then for all $t \in T_{1,i}$:*

$$M_{1,i}[t > M'_{1,i} \Leftrightarrow N_{1,i}(M_{1,i}) - t \rightarrow N_{1,i}(M'_{1,i}).$$

Proof. The proof is by induction on i.

($\Rightarrow$) We must show that

$$M_{1,i}[t > M'_{1,i} \Rightarrow N_{1,i}(M_{1,i}) - t \rightarrow N_{1,i}(M'_{1,i}).$$

Basis (i=1) Suppose that $M_{1,1}[t > M'_{1,1}$, by definition 2,

$$(t \in \Gamma^{-1}(p_1) \wedge M'_{1,1}(1) = M_{1,1}(1) + 1) \vee (t \in \Gamma(p_1) \wedge M'_{1,1}(1) = M_{1,1}(1) - 1)$$

from Lemma 1, it implies that

$$P_1(M_{1,1}(1)) - t \rightarrow P_1(M'_{1,1}(1))$$

and by definition 3 we conclude:

$$N_{1,1}(M_{1,1}) - t \rightarrow N_{1,1}(M'_{1,1})$$

Induction Let $i \geq 1$, assume that the lemma is true whenever $n \leq i$. Suppose that $M_{1,i+1}[t > M'_{1,i+1}$, there are three cases.

CASE 1 $t \in T(\{p_{i+1}\})$ and $t \in T(\{p_1, ..., p_i\})$.

By Lemma 1,

$$P_{i+1}(M_{1,i+1}(i+1)) - t \rightarrow P_{i+1}(M'_{1,i+1}(i+1))$$

and by induction hypothesis,

$$N_{1,i}(M_{1,i}) - t \rightarrow N_{1,i}(M'_{1,i}).$$

Both pocesses are synchronized by the operator $|[T(\{p_{i+1}\}) \cap T(\{p_1, ..., p_i\})]|$ we can therefore conclude from definition 3 that:

$$N_{1,i+1}(M_{1,i+1}) - t \rightarrow N_{1,i+1}(M'_{1,i+1})$$

CASE 2 $t \notin T(\{p_{i+1}\})$ and $t \in T(\{p_1, ..., p_i\})$. In that case $M_{1,i+1}(i+1) = M'_{1,i+1}(i+1)$.

We have that

$$P_{i+1}(M_{1,i+1}(i+1)) = P_{i+1}(M'_{1,i+1}(i+1))$$

and by induction hypothesis,

$$N_{1,i}(M_{1,i}) - t \rightarrow N_{1,i}(M'_{1,i}).$$

The process $N_{1,i}(M_{1,i})$ can make its transition independently since it is not synchronized by the operator $|[T(\{p_{i+1}\}) \cap T(\{p_1, ..., p_i\})]|$ we can therefore conclude from definition 3 that:

$$N_{1,i+1}(M_{1,i+1}) - t \rightarrow N_{1,i+1}(M'_{1,i+1})$$

CASE 3 $t \in T(\{p_{i+1}\})$ and $t \notin T(\{p_1, ..., p_i\})$. In that case $M_{1,i} = M'_{1,i}$.

By Lemma 1,

$$P_{i+1}(M_{1,i+1}(i+1)) - t \rightarrow P_{i+1}(M'_{1,i+1}(i+1))$$

and we have that

$$N_{1,i}(M_{1,i}) = N_{1,i}(M'_{1,i}).$$

The process $P_{i+1}(M_{1,i+1}(i+1))$ can make its transition independently since it is not synchronized by the operator $||[T(\{p_{i+1}\}) \cap T(\{p_1, ..., p_i\})]||$ we can therefore conclude from definition 3 that:

$$N_{1,i+1}(M_{1,i+1}) - a \rightarrow N_{1,i+1}(M'_{1,i+1})$$

($\Leftarrow$). Similar.

7 Conclusion

In the original semantics of Lotos, parallelism is modeled by nondeterministic interleaving of concurrent actions. The goal of Olderog and Degano et al. was to provide a true concurrency semantics for CSP and CCS. In such a semantics a global system transition may involve multiple independent actions. From a verification point of view, interleaving semantics is an interesting abstraction since it avoids the "transition space explosion" problem. The transition domain of an interleaving semantics is T whereas in true concurrency the global transition domain is 2^T. Consider the process:

process $p[a, b, c] :=$
$(a; stop|||b; stop)[]c; stop$
endproc

The semantics of Olderog or Degano et al. would produce the net in Fig. 2 whereas our semantics generate the net in Fig. 3.

Parallelism inside a choice operator is not decomposed in our semantics whereas it is with the other authors' semantics. In general our semantics exhibits less parallelism. In compensation, our semantics does not lead to the problem of non-updated markings. For instance in Fig. 2, when transition a is fired, the successor marking is:

$$M' = \{stop|||, (|||b; stop)[]c; stop\}$$

However, according to the original Lotos semantics:

$$p[a, b, c] - a \rightarrow stop|||b; stop$$

we say that the marking M' is not updated because:

$$M' \neq dec(stop|||b; stop)$$

A consequence of this is that, for a given PLotos specification, the transition system obtained with the original semantics and the case graph obtained with the P/T-net semantics are isomorphic with the function *dec*. In [Barbeau, Bochmann 1990] we show that this property can be exploited

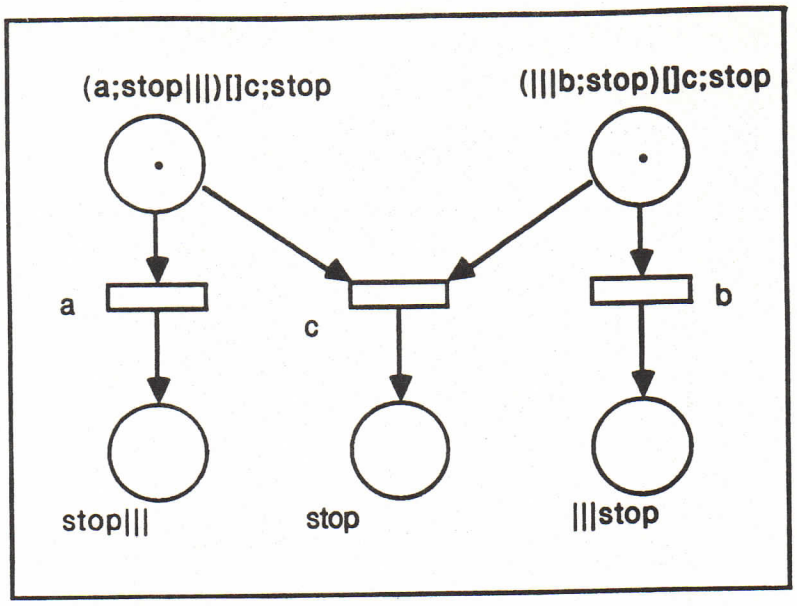


Figure 2: Olderog's Semantics

to apply P/T-net verification techniques to Lotos without even translating explicitly Lotos into P/T-nets.

This paper has presented a derivation system for obtaining an equivalent P/T-net from a given Lotos specification. Since the subset of Lotos that we considered has the power of Turing machines, it is necessary to restrict the language such that finite structure P/T-nets can be obtained. Moreover, we have shown that the converse is also possible. We therefore demonstrated that our PLotos has the power of finite structure P/T-nets.

It is possible to consider different subsets of Lotos from which finite structure P/T-nets can be obtained. For instance, in a sequential composition $B_1 >> B_2$ or disabling $B_1 [> B_2$ we can allow recursion but forbid parallelism since B_1 can still be mapped to a finite transition system. As future area of investigation, we plan to extend this derivation system to support process and interaction parameters with the help of Coloured Petri nets [Jensen 1981] to model Lotos behaviors.

References

AZEMA, P., JUANOLE, G., SANCHIS, E., MONTBERNARD, M. [1984]. *Specification and Verification of Distributed Systems Using Prolog Interpreted Petri Nets*, 7th International Conference on Software Engineering.

BARBEAU, M., BOCHMANN, G. V. [1990]. *Extension of the Karp and Miller Procedure to Lotos Specifications*, in preparation.

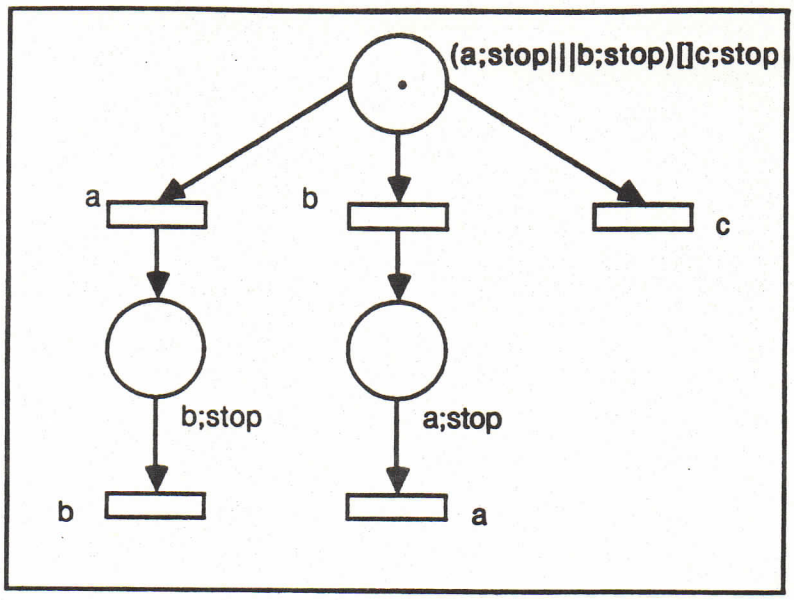


Figure 3: Our Semantics

BOLOGNESI, T., BRINKSMA, E. [1987]. *Introduction to the ISO Specification Language Lotos*, Computer Networks and ISDN, Vol. 14, No. 1, (25-59).

BOLOGNESI, T., SMOLKA, S. A. [1987]. *Fundamental Results for the Verification of Observational Equivalence: A Survey*, Proc. of PSTV VII, Zurich.

BOUDOL, G., ROUCAIROL, G., SIMONE, R. de [1985]. *Petri Nets and Algebraic Calculi of Processes*, Advances in Petri Nets (41-58).

DEGANO, P., NICOLA, R. de, MONTANARI, U. [1988] *A Distributed Operational Semantics for CCS Based on Condition/Event Systems*, Acta Informatica 26, (59-91).

GOLTZ, U. [1988]. *On Representing CCS Programs by Finite Petri Nets*, in : M. Chytil et al. (Eds.), *Mathematical Foundations of Computer Science 1988*, LNCS 324, Springer-Verlag (339-350).

HOPCROFT, J. E., ULLMAN, J. D. [1979]. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley.

ISO [1988]. *Lotos - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*, ISO 8807, E. Brinksma (Ed.).

JENSEN, K. [1981]. *Coloured Petri Nets and the Invariant Method*, TCS, Vol. 14, (317-336).

MARCHENA, S., LEON, G. [1989]. *Transformation from Lotos Specs to Galileo Nets*, in K. J. Turner (Ed.): Formal Description Techniques, North-Holland.

OLDEROG, E.-R. [1987]. *Operational Petri Net Semantics for CCSP*, LNCS 266, Springer-Verlag.

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

TAUBNER, D. [1989]. *The Representation of CCS Programs by Finite Predicate/ Transition Nets*, Proc of 10th Int. Conf. on Application and Theory of Petri Nets, Bonn.

WAITE, M. W., GOOS, G. [1985]. *Compiler Construction*, Springer-Verlag.

Appendix A: PLotos

To formalize our PLotos we use the technique of attribute grammars [Waite, Goos 1985] since they can specify concisely and precisely the syntax and the semantical constraints of a language.

An attribute grammar is a context-free grammar but terminal and nonterminal symbols of the grammar have attributes. To each rule $B_0 ::= B_1, ..., B_n$ is associated a set of assignment statements and assertions, the **attribution**. Assignments define attribute values whereas assertions are logical formulae that put constraints on the values.

The attribute a of grammar symbol B is denoted as $B.a$. The informal interpretation of the attributes of our grammar is as folllows:

- $B.func$ is the functionality of behavior B. It is equal to *exit iff* the behavior terminates with the successful termination action. Otherwise its value is *noexit*.
- $B.callee$ is the name of the process that instantiates behavior B. This attribute is used in rule (p9) to prevent the occurrence of |[]| operators in recursive processes.
- $B.guarded$ is *true iff* the behavior B is guarded.
- $B.call$ is the set of process names instantiated in behavior B.
- $p.rec$ is *true iff* process p is recursive.

We use *italics* for nonterminal symbols and **boldface** for keywords.

Rule (p1)

S ::= **specification** $p[g_1, ..., g_n]$:=
B **where** D
endspec.

Attribution

$B.callee \leftarrow p$

$B.guarded \leftarrow false$

ASSERT $B.func = noexit$

Rule (p2)

$D ::= DP|P.$

Rule (p3)

$P ::=$ **process** $p[g_1, ..., g_n] :=$

B

endproc.

Attribution

$B.callee \leftarrow p$

$B.guarded \leftarrow false$

$p.func \leftarrow B.func$

$p.rec \leftarrow p \in closure(B.call)$

where *closure* is a function which computes the transitive closure of the call relation in order to obtain all the process names directly or indirectly instantiated by p.

Rule (p4)

$B ::=$ **stop.**

Attribution

$B.func \leftarrow noexit$

$B.call \leftarrow \{\}$

Rule (p5)

$B ::= a; B_1.$

Attribution

$B_1.callee \leftarrow B.callee$

$B_1.guarded \leftarrow true$

$B.func \leftarrow B_1.func$

$B.call \leftarrow B_1.call$

Rule (p6)

$B ::= B_1[]B_2.$

Attribution

$B_1.callee \leftarrow B_2.callee \leftarrow B.callee$

$B_1.guarded \leftarrow B_2.guarded \leftarrow B.guarded$

$B.func \leftarrow B_1.func$

ASSERT $B_1.func = B_2.func$

$B.call \leftarrow B_1.call \cup B_2.call$

Rule (p7)

$B ::= p[g_1, ..., g_n].$

Attribution

$p.callee \leftarrow B.callee$

$B.func \leftarrow p.func$

$B.call \leftarrow \{p\}$

ASSERT $p.rec \Rightarrow B.guarded$

Rule (p8)

$B ::= B_1 ||| B_2.$

Attribution

$B_1.callee \leftarrow B_2.callee \leftarrow B.callee$

$B_1.guarded \leftarrow B_2.guarded \leftarrow B.guarded$

$B.func \leftarrow B_1.func$

ASSERT $B_1.func = B_2.func$

$B.call \leftarrow B_1.call \cup B_2.call$

Rule (p9)

$B ::= B_1 |[g_1, ..., g_n]| B_2.$

Attribution

$B_1.callee \leftarrow B_2.callee \leftarrow B.callee$

$B_1.guarded \leftarrow B_2.guarded \leftarrow B.guarded$

$B.func \leftarrow B_1.func$

ASSERT $B_1.func = B_2.func$

$B.call \leftarrow B_1.call \cup B_2.call$

ASSERT $B.callee \notin B.call$

Rule (p10)

$B ::=$ **exit.**

Attribution

$B.func \leftarrow exit$

$B.call \leftarrow \{\}$

Rule (p11)

$B ::= B_1 >> B_2.$

Attribution

$B_1.callee \leftarrow B_2.callee \leftarrow B.callee$

$B_2.guarded \leftarrow true$

$B.func \leftarrow B_2.func$

ASSERT $B_1.func = exit$

$B.call \leftarrow B_1.call \cup B_2.call$

ASSERT $(\forall p \in closure(B_1.call))[p.rec = false]$

Rule (p12)

$B ::= B_1[> B_2.$

Attribution

$B_1.callee \leftarrow B_2.callee \leftarrow B.callee$

$B_2.guarded \leftarrow true$

$B.func \leftarrow B_2.func$

ASSERT $B_1.func = B2.func$

$B.call \leftarrow B_1.call \cup B_2.call$

ASSERT $(\forall p \in closure(B_1.call))[p.rec = false]$

Rule (p13)

$B ::=$ **hide** $g_1, ..., g_n$ **in** $B_1.$

Attribution

$B_1.callee \leftarrow B.callee$

$B_1.guarded \leftarrow B.guarded$

$B.func \leftarrow B_1.func$

$B.call \leftarrow B_1.call$