A Subset of Lotos with the Computational Power of Place/Transition-Nets [1]

Michel Barbeau, Gregor v. Bochmann[2]

Abstract

In this paper, we define a subset of Lotos that can be modelled by finite Place/ Transition-nets (P/T-nets). That means that specifications in that Lotos subset can be translated into finite P/T-nets and validated using P/T-net verification techniques. An important aspect of our work is that we show that conversely P/T-nets can be simulated in our Lotos subset. It means that the constraints we put on Lotos in order to obtain finite nets are minimally restrictive. We may also conclude that our Lotos subset and P/T-nets have equivalent computational power. To the best of our knowledge, no such bidirectional translation scheme has been published before.

Topics:

Relationships between net theory and other approaches.

1. Introduction

In this paper, we define a subset of Basic Lotos [Bolo 87, ISO 88] that can be modelled by finite Place/Transition-nets (P/T-nets). That means that specifications in that Lotos subset can be represented and translated into finite P/T-nets and validated using P/T-net verification techniques. An important aspect of our work is that we show that conversely P/T-nets can be simulated in our Lotos subset. It means that the constraints we put on Lotos in order to obtain finite nets are minimally restrictive. We may also conclude that our Lotos subset and P/T-nets have equivalent computational power. To the best of our knowledge, no such bidirectional translation scheme has been published before.

The problem of modelling process-oriented languages, and more specifically CCS and CSP like languages, by Petri nets has been tackled by several authors. Cindio et al. [Cind 83], Degano et al. [Dega 88], Glabbeek [Glab 87], Goltz [Golt 84a, 84b, 88], Nielsen [Niel 86], Olderog [Olde 91] and Taubner [Taub 89] considered CCS or CSP, or both. Lotos has been worked by Marchena and Leon [Marc 89], and Garavel and Sifakis [Gara 90]. The approaches may be

[1] **This work was performed within a research project on object-oriented specifications funded by Bell-Northern Research (BNR) and the Computer Research Institute of Montréal (CRIM). Funding from the Natural Sciences and Engineering Research Council of Canada is also acknowledged.**

[2] First author's address: Université de Sherbrooke, Département de mathématiques et d'informatique, Sherbrooke (Québec), Canada, J1K 2R1. Second author's address: Université de Montréal, Département d'IRO, C.P. 128, Succ. "A", Montréal (Québec), Canada, H3C 3J7.

categorized based on the following criteria: i) style of definition, ii) finiteness of the representation, and iii) distinction of concurrency and nondeterminism.

One of two definition styles may be adopted, namely denotational or operational. A denotational style is used in: [Cind 83], [Gara 90], [Glab 87], [Golt 84a, 84b, 88], [Niel 86], [Marc 89] and [Taub 89], whereas an operational style, à la Plotkin, is used in: [Dega 88], [Olde 91] and in the present paper. In opposition to the operational approach, the denotational style is constructive. It means that the definition yields directly to a procedure for translating terms of the process-oriented language to Petri nets. However, we shown in [Barb 91a, b] that thanks to our operational definition an important P/T-net verification method can be adapted to Lotos without even translating the latter to the former.

Another important matter is whether or not the Petri net representation of the process-oriented language is finite. It is well known that an unbounded number of Petri net places and transitions is required to represent a process-oriented language when recursion is combined with parallel composition, sequential composition, hiding and disabling operators. This difficulty means that it is impossible to transfer to the process-oriented language several important verification techniques elaborated for Petri nets, since they require finite nets. Note that in our mind, finite nets does not mean finite state systems. Finite representations can be obtained by restricting the process-oriented language or using high-level Petri net models. Finite representations for subsets of CCS are proposed in [Golt 88], using P/T-nets, and in [Taub 89], using Predicate/Transition-nets, which is a high-level model. Finite extended Petri nets are generated from Lotos, with the finite control property, in [Gara 90], this work is also interesting because the data part of Lotos is also handled. In this paper we define a subset of Basic Lotos, with syntactical constraints, that can be modelled by finite P/T-nets.

Non distinction of concurrency and nondeterminism means that Lotos expressions such as $a; stop|||b; stop$ and $a; b; stop[]b; a; stop$ have the same semantic interpretation. Distinction of concurrency and nondeterminism allows accurate representation of behaviors by partial orders. It is a representation that shows just natural dependencies between actions. Multi-sets of actions are possible in a single transition. This has an impact on treatment of fairness problems [Reis 84]. Our Place/Transition-net semantics is less attractive, than definitions described in Refs. [Dega 88], [Golt 88], [Niel 86] and [Olde 91], with respect to distinction of concurrency and nondeterminism.

An important feature in our approach is that we show that P/T-nets can be simulated in our Lotos subset. Other authors have proposed simulations of Petri nets in languages such as Prolog, Azema et al. [Azem 84], or Meije, Boudol et al. [Boud 85]. These simulations are not in languages that have been shown translatable into finite Peo nets. The goal of Azema et al. is to use Prolog as a simulation tool for Petri nets whereas the aim of Boudol et al. is to provide a textual representation for Petri nets. Translation into Lotos of another graphical representation for behaviors, called *Process-Gate Network*, is described [Bolo 90].

In Section 2, we introduce the P/T-net model. Our Basic Lotos subset that can be translated into finite P/T-nets is called PLotos and is defined in Section

3. In Section 4, we discuss modelling of PLotos by P/T-nets. The converse simulation is presented in Section 5. We conclude in Section 6.

2. P/T-nets

We represent a P/T-net [Pete 81] as a tuple (P, T, Act, M_0) where:

- P, is a set of places $\{p_1, ..., p_n\}$,
- $T \subseteq \mathcal{N}^P \times Act \times \mathcal{N}^P$, is a transition relation,
- Act, is a set of transition labels, and
- $M_0 \in \mathcal{N}^P$, is the initial marking.

A P/T-net is **finite** if the sets P, T and Act are finite.

$\mathcal{N}$ is the set of non-negative integers. $\mathcal{N}^P$ denotes the set of multi-sets over the set P. An element $t = (X, a, Y) \in T$ is also denoted as $X - a \rightarrow Y$. Its **preset** $pre(t)$ is X, **postset** $post(t)$ is Y and **action** $act(t)$ is a. The multi-sets X and Y are also called respectively the input and output places of t. We denote as $pre(t)(p)$ ($post(t)(p)$) the number of instances of the element p in the preset (postset) of t.

The operators $\leq$, $+$ and $-$ denote respectively multi-set inclusion, summation and difference. A multi-set X can also be seen as the formal sum: $X = \sum_{p \in P} pre(t)(p)p$.

A Petri net marking is also a multi-set. We denote by $M(p_i)$ the number of instances of the element p_i in the multi-set M. Instances of the element p_i are also called tokens inside place p_i.

$pre(t)(p)$ is the number of tokens that place p must contain to enable transition t. 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 represented as $M(t > M'$.

We define the **reachability graph** of a P/T-net $N = (P, T, Act, M_0)$ as a graph $RG(N) = (RS, E, M_0)$ where:

1. RS is the reachability set, i.e. a set of markings of N,
2. $E \subseteq RS \times Act \times RS$, is a transition relation, and
3. for all $M \in RS, t \in T$, if $M(t > M'$ then $M' \in RS$ and $(M, act(t), M') \in E$.

3. Definition of the Syntax of PLotos

In this section we define the syntax of a subset of Basic Lotos, namely PLotos which is equivalent, in terms of computational power, to finite P/T-nets (to be shown formally in Section 4). First, we discuss Basic Lotos. Then, we define PLotos as Basic Lotos along with syntactical constraints. The syntax of Basic Lotos is given in Ref. [Bolo 87] and in Appendix A.

It is well known that Basic Lotos has the computational power of Turing machines. Our aim is to reduce the power of Basic Lotos to the one of P/T-nets. Before we state the syntactical constraints that make PLotos equivalent to P/T-nets, we define preliminary concepts.

The "calls" relation

Let p_1 be a process and B_{p_1} its defining behavior-expression. We say that p_1 **calls** p_2 if B_{p_1} has one or more occurrences of p_2. This relation is denoted as:

$$C = \{(p_1, p_2) : p_1 \text{ calls } p_2\}$$

The mutual recursion relation

Let C^+ be the transitive closure of C. We define in terms of C^+ the **mutual recursion** relation Φ as follows:

$$\Phi = \{(p_1, p_2) : (p_1, p_2) \in C^+ \wedge (p_2, p_1) \in C^+\}$$

Recursive process

The process p is recursive if $(p, p) \in \Phi$.

Functionality

The **functionality** of a behavior B is equal to *exit iff* every alternative in B terminates with the successful termination action δ, otherwise it is equal to *noexit* [Bolo 87].

Context

A Lotos **context** $C[\,]$ is a Lotos behavior-expression with a formal "behavior-expression" parameter denoted as "[ ]". If $C[\,]$ is a context and B is a behavior-expression then $C[B]$ is the behavior-expression that is the result of replacing all occurrences of "[ ]" in $C[\,]$ by B. For example, let $C[\,]$ be the Lotos context $g;[\,]$. The behavior-expression $C[stop]$ is defined as $g; stop$.

Guarded process

A process instantiation term p is guarded if it occurs in any of the following forms:

- $C_1[a; C_2[p]]$
- $C_1[B >> C_2[p]]$
- $C_1[B[> C_2[p]]$

where $C_1[\ ]$ and $C_2[\ ]$ are any contexts, a is any gate identifier and B is any behavior-expression.

PLotos

PLotos is defined as the subset of Basic Lotos that satisfies the following syntactical constraints:

1. Terms that instantiate recursive processes must be guarded.
2. Operands B_1 and B_2 in a parallel composition $B_1|||B_2$ must have the *noexit* functionality.
3. Let $C_1[\ ]$ and $C_2[\ ]$ denote two contexts and B denote a behavior-expression. For any pair $(p_1, p_2) \in \Phi$ the defining behavior-expression of p_1 may not have the following patterns:

   3.1 $C_1[C_2[p_2] * B]$, where the operator "$*$" is either "$|[g_1, ..., g_n]|$" or "$>>$" or "$[>$"

   3.2. $C_1[B|[g_1, ..., g_n]|C_2[p_2]]$

   3.3 $C_1[hide\ g_1, ..., g_n\ in\ C_2[p_2]]$
4. The behavior-expression B_1 must have the *exit* functionality in behavior-expressions of the forms: $B_1 >> B_2$ or $B_1[> B_2$.

Mutual recursion is possible in sub-terms of the form "$B_1|||B_2$", with operands of functionality *noexit* (i.e. constraint 2). The control is not finite state but can be represented by a finite P/T-net. It is possible to simulate an arbitrarily large stack if the constraint 3.1 is unsatisfied (e.g. [Gotz 86]). Arbitrarily large stacks cannot be simulated by finite P/T-nets.

PLotos has the computational power of finite P/T-nets. That is, every PLotos specification can be modelled by an equivalent finite P/T-net. Conversely, every finite P/T-net can be modelled by an equivalent PLotos specification. In Section 4, we show how a PLotos specification can be modelled by a finite P/T-net. The converse is demonstrated in Section 5.

4. P/T-net Semantics for PLotos

4.1. General Idea

Our PLotos to P/T-nets mapping is inspired by the work of Olderog [Olde 91] for CSP. In general, a Lotos behavior-expression B represents the composition

of several concurrent components. In our simulation of PLotos by P/T-nets, the expression B is explicitly decomposed into its components which become tokens when this behavior is activated. More precisely, parallel components and states of parallel components are respectively modelled by Petri net tokens and places. The place in which a token is contained denotes the state of the corresponding component. Every Lotos gate occurrence is modelled by a Petri net transition. Tokens, contained in the transition input places, represent components synchronized on the gate. Tokens deposited into the transition output places represent the successor components after the transition has occurred. Several tokens, contained in the same place, represent several identical components. This models unbounded process instantiation with finite P/T-nets.

For example, the Lotos expression $u; v; stop|[u]|u; stop$ represents two concurrent components. The first component executes actions u and v and then stops. The second component executes action u and becomes inactive. Both components are coupled on gate u and are therefore dependent on each other with respect to the occurrence of u. The decomposition of $u; v; stop|[u]|u; stop$ into its components is denoted as the multi-set $\{u; v; stop|[u]|, |[u]|u; stop\}$. In this syntax, we represent explicitly the fact that the components are coupled on gate u by concatenating the symbol $|[u]|$ to the right of $u; v; stop$ and to the left of $u; stop$.

Places modelling states of components are labelled by the corresponding component-expressions. Transitions are labelled by gate names. The "**stop**" expression represents inaction and does not appear in the P/T-net. In our construction, edges from places to transitions are always one valued (i.e. $(\forall t, p)[pre(t)(p)$ equals 0 or 1]) and every place has a distinct label. **We unambiguously denote a place by its label.** ¿From the above multi-set of components, it is possible to derive the transition represented as the triple:

$$\{u; v; stop|[u]|, |[u]|u; stop\} - u \rightarrow \{v; stop|[u]|\}$$

To derive such triples, we define: i) a function decomposing PLotos behavior-expressions into component-expressions, and ii) a system of inference rules. The head of each rule matches a term of the form:

$$\{p_1, ..., p_m\} - a \rightarrow \{q_1, ..., q_n\}$$

Such a rule can be applied to infer, as a function of the component-expressions, a transition with preset $\{p_1, ..., p_m\}$, action a and postset $\{q_1, ..., q_n\}$. For instance, the rule:

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

is used to infer the transition:

$$\{v; stop|[u]|\} - v \rightarrow \{\}$$

We substituted $\{v; stop\}$, u and v to respectively M_1, S and a. M_1' is empty because the decomposition of "**stop**" is defined as the empty set.

We introduce the decomposition function in Section 4.3 then we present, in Section 4.4, the inference rules. But first, in Section 4.2 we translate Lotos specifications in a form that makes easier development of consistency proofs.

4.2. Normal Form Specifications

PLotos specifications are rewritten into simpler forms, called normal form specifications. Sub-terms in which mutual recursion does not occur are expanded, that is, process definitions are substituted for process calls. Then we distinguish every parallel composition $B_1|[g_1,...,g_n]|B_2$ by labelling the operator with an unique value k. This is represented as $|[g_1,...,g_n]|_k$. For example:

process $p_1[a,b,c]$: $noexit$:=
$(p_2[a,b]|||p_2[a,b])[]c;p_1[a,b,c]$
endproc
process $p_2[a,b]$: $noexit$:=
$a;b;stop|[a]|a;stop$
endproc

is rewritten as:

process $p_1[a,b,c]$: $noexit$:=
$((a;b;stop\ |[a]|_1\ a;stop)|||(a;b;stop\ |[a]|_2\ a;stop))[]c;p_1[a,b,c]$
endproc

Static relabelling instead of dynamic relabelling is performed when process instantiation terms are substituted by the corresponding defining behavior-expressions. This issue is further discussed in Section 4.3.

As discussed in Section 4.1, with the small example: $\{u;v;stop|[u]|,|[u]|u;stop\}$, every general parallel composition is decomposed into two or more component-expressions during the PLotos to the P/T-net modelling process. Labelling of general parallel operators with a unique value is required to preserve important contextual information of component-expressions. This information is required to unambiguously determine which component-expressions need to be synchronized together.

4.3. 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 finite multi-sets of component-expressions.

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, the de-

composition function *dec* is defined as follows:

$$
\begin{array}{lrl}
(d1) & dec(stop) & := \{\} \\
(d2) & dec(a; B_1) & := \{a; B_1\} \\
(d3) & dec(B_1 [] B_2) & := \{B_1 [] B_2\} \\
(d4) & dec(p[g_1, ..., g_n]) & := dec(B_p[g_1/h_1, ..., g_n/h_n]) \\
(d5) & dec(B_1 ||| B_2) & := dec(B_1) + dec(B_2) \\
(d6) & dec(B_1 |[S]|_k B_2) & := dec(B_1).|[S]|_k + |[S]|_k.dec(B_2) \\
(d7) & dec(B_1 >> B_2) & := \{B_1 >> B_2\} \\
(d8) & dec(B_1 [> B_2) & := \{B_1 [> B_2\} \\
(d9) & dec(hide\ S\ in\ B_1) & := hide\ S\ in.dec(B_1) \\
(d10) & dec(exit) & := \{exit\}
\end{array}
$$

where:

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

The *dec* function is deterministic, taking into account operator precedences given in [ISO 88]. The restriction to guarded recursive processes (see Section 3) is required to stop recursion in the *dec* function.

The relabelling operator is not user accessible and exists for the semantic description of process instantiation. In Lotos, relabelling is dynamic; gates are renamed at the execution time. For instance, let us consider this process definition:

process $p[a, b]$: **noexit** :=
$a; stop|[a, b]|b; stop$
endproc

Instantiating p with $p[a, a]$ yields an inactive process with dynamic relabelling, since the expression $a; stop|[a, b]|b; stop$ is inactive. Nevertheless, with static renaming $p[a, a]$ yields the expression $a; stop|[a, a]|a; stop$ which may perform the action a and becomes inactive.

It can be shown easily that for injective relabelling operators, static and dynamic relabelling are equivalent. For the sake of simplicity, hereafter we consider solely injective relabellings and perform static renaming, that is syntactical substitution. We believe that this restriction is not significant, at least from a computational point of view, and it is fulfilled in many applications.

4.4. Inference Rules

This section exposes the inference rules of our PLotos to P/T-nets mapping. The P/T-net $N = (P, T, Act, M_0)$, with reachability set RS, associated to a PLotos behavior-expression B is defined as:

1.
   - $M_0 = dec(B)$
   - $M_0 \in RS$
   - $(\forall p)[M_0(p) > 0 \Rightarrow p \in P]$
2. if $M \in RS$ and $X \leq M$ and $X - a \rightarrow Y$ then
   - $(\forall p)[Y(p) > 0 \Rightarrow p \in P]$
   - $(X, a, Y) \in T$
   - $a \in Act$
   - $M' = M - X + Y$
   - $M' \in RS$
3. only the elements that can be obtained from items 1 or 2 are in P, T and Act

The transition instances are inferred from the rules below. For all PLotos behavior-expressions B_1, B_1', B_2, B_2', action name a, list $S = g_1, ..., g_n$ of synchronization gates and component-expression multi-sets 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 $M_1.|[S]|_k - a \rightarrow M_1'.|[S]|_k$
(r5) if $M_2 - a \rightarrow M_2'$ and $a \notin \{S, \delta\}$
then $|[S]|_k.M_2 - a \rightarrow |[S]|_k.M_2'$
(r6) if $M_1 - a \rightarrow M_1'$ and $M_2 - a \rightarrow M_2'$ and $a \in \{S, \delta\}$
then $M_1.|[S]|_k + |[S]|_k.M_2 - a \rightarrow M_1'.|[S]|_k + |[S]|_k.M_2'$
(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 $\{B_1 >> B_2\} - i \rightarrow dec(B_2)$
(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 $\{B_1 [> B_2\} - \delta \rightarrow dec(B_1')$
(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 $hide\ S\ in.M_1 - i \rightarrow hide\ S\ in.M_1'$
(r14) $\{exit\} - \delta \rightarrow \{stop\}$

In the "if part" of inference rules (r2), (r3), (r7) (r8), (r9), (r10) and (r11) behavior B_1 (B_2) makes a transition to behavior B_1' (B_2') on action a or δ in accordance with the original Basic Lotos semantics.

Theorem 1 *(Boundedness theorem) PLotos can be modelled by a finite P/T-net.*

We must show [3] that any PLotos normal form specification:

$$\textbf{specification} \ldots \textbf{behavior} B_0 \ldots \textbf{endspec}$$

can be modelled by a P/T-net $N = (P, T, Act, M_0)$ whose sets P, T and Act are finite (note that the associated reachability set RS is not necessarily finite).

In the sequel, the operators in component-expressions are classified as follows:

- **stop**, **exit**, and $p[g_1, ..., g_n]$ are nullary operators.
- $|[S]|_k$, and **hide** S **in** are unary operators.
- ";", ">>" and "[>" are binary operators.

Note that the operator "|||" never appears in a component-expression.

(*The set Act is finite*). In a normal form PLotos specification there is a finite number of gates. Lotos gates are translated to P/T-net transition labels, i.e. elements of Act. Consequently, the set Act is finite.

(*The set P is finite*). The statement "The set P is finite" is equivalent to the statement:

> *S1*: There exists a K such that for all $p \in P$, the number of nullary operators in p is less than K.

This equivalence is a consequence of the conjunction of the following facts (let us suppose that we distinguish, in the normal form specification, every operator from the others): i) The set of gates and nullary, unary and binary operators, that can possibly be used in a component-expression is finite. ii) Every unary or binary operator is used at most once in a component-expression. iii) Using a finite number of gates and nullary, unary and binary operators, and zero or one occurrence of every unary or binary operator, there is a finite number of syntactically different component-expressions that can be constructed.

The negation of statement *S1* is the following statement:

[3] The proof technique is similar to the one used in [Gara 89].

S2: It is possible to infer from $dec(B_0)$ a component-expression p in which there is an unbounded number of nullary operators.

Statement *S2* implies that there exist processes p_1, p_2 with:

$$(p_1, p_2) \in \Phi$$

a marking:

$$M_n \in RS$$

and a component-expression p with:

$$p \in M_n$$

where there is a nullary operator who occurs an unbounded number of times in p. This unbounded number of occurrences is due to the substitution of recursive instantiation terms of p_1 by its defining behavior-expression B_{p_1} of p_1. Nonetheless, this true solely if B_{p_1} has one of the following patterns:

- $C_1[C_2[p_2] * B]$, where the operator "$*$" is either "$|[g_1, ..., g_n]|$" or "$>>$" or "$[>$".
- $C_1[B|[g_1, ..., g_n]|C_2[p_2]]$.
- $C_1[hide\ g_1, ..., g_n\ in\ C_2[p_2]]$

where $C_1[\ ]$ and $C_2[\ ]$ denote two contexts and B denote any behavior-expression. However, these patterns ar disallowed in PLotos (see Section 3).

(*The set T is finite*). This follows from the fact that from a finite set of syntactically different component-expressions, application of the inference rules can derive a finite number of transitions.

The next theorem states that the P/T-net semantics is in accordance with the original semantics of Lotos.

Theorem 2 *(Consistency theorem) The Petri net semantics of Lotos is consistent with the standard Lotos semantics. That is, for all PLotos behavior expression B, marking M with $dec(B) := M$:*

1. $[B - a \rightarrow B'] \Rightarrow (\exists M')(\exists t)[M(t > M' \wedge act(t) = a \wedge dec(B') := M']$
2. $[M(t > M'] \Rightarrow (\exists B')[B - act(t) \rightarrow B' \wedge dec(B') := M']$

The proof is by induction on the number of operators in a behavior-expression B and refers to the standard Lotos semantics in Refs. [Bolo87] and [ISO88].

Definition 1 *Two graphs $A_1 = (S_1, E_1, n_1)$ and $A_2 = (S_2, E_2, n_2)$ are **bisimular** [Park 81] if there exists a relation $R \subseteq S_1 \times S_2$, called a bisimulation relation, with:*

1. $(n_1, n_2) \in R$, *and for all* $(n, m) \in R$

2. $[(n, a, n') \in E_1] \Rightarrow (\exists m')[(m, a, m') \in E_2 \wedge (n', m') \in R]$, *and*

3. $[(m, a, m') \in E_2] \Rightarrow (\exists n')[(n, a, n') \in E_1 \wedge (n', m') \in R]$.

Corollary 1 *Let B be a PLotos behavior-expression, with transition graph TG, and let $N = (P, T, Act, M_0)$ be the associated P/T-net with reachability graph $RG(N)$. The dec function is a homomorphism from TG reachability set to $RG(N)$ reachability set. TG **and** $RG(N)$ **are bisimular** under the bisimulation relation R defined as:*

1. $(B, M_0) \in R$, *and*

2. *For all B' in the TG reachability set and for all M in $RG(N)$ reachability set:*

$$(B', M) \in R \Longleftrightarrow dec(B') := M$$

The *dec* function is a graph homomorphism because it identifies equivalent Lotos behavior-expressions $B_1|||B_2$ with $B_2|||B_1$, and $B_1|||(B_2|||B_3)$ with $(B_1|||B_2)|||B_3$. These equivalences are in accordance with the commutativity and the associativity laws in [ISO 88]. Solely syntactic nature information is lost, "*dec*" preserves all semantic properties. This can be illustrated by the following commutative diagram:

$$\begin{array}{ccc} B & -dec \rightarrow & M \\ | & & | \\ a & & a \\ \downarrow & & \downarrow \\ B' & -dec \rightarrow & M' \end{array}$$

5. Simulation of P/T-nets in PLotos

In Section 4, we identified a subset of Lotos, PLotos, that can be modelled by finite P/T-nets. In this section, we show that conversely P/T-nets can be simulated by PLotos. These two facts lead to the conclusion that PLotos and P/T-nets are equivalent models, that is models with equivalent computational power.

We make two reasonable hypotheses. First, we simulate in PLotos, P/T-nets whose place to transition edges are one valued, i.e.:

$$(\forall t, p)[pre(t)(p) \text{ equals 0 or 1}]$$

This restriction is not a handicap because it has been proved [Kasa 82] that P/T-nets of arbitrary edge valuation can be simulated by P/T-nets whose edges are all valued to one, with language equality equivalence.

Second, we assume that no place is simultaneously in the preset and the postset of a single transition. This restriction is not significant. P/T-nets with circuits made of one place and one transition can be simulated by circuit free (*pure*) P/T-nets [Bram 83].

Before we go into detail, we give a brief overview of the simulation. Given a P/T-net $N = (P, T, Act, M)$ with set of places $P = \{p_1, ..., p_n\}$, we define a PLotos process $N_{1,n}$ with equivalent behavior. That is, the reachability graph of N and the transition graph of $N_{1,n}$ are bisimular. The process $N_{1,n}$ is defined inductively on the number n of places.

Every transition $t \in T$ is mapped to a Lotos gate, also named t. Every place $p_i \in P$ is mapped to three PLotos processes, namely $token_i$, p_i and $P_i(k)$. The process $token_i$ models a token inside the place p_i. It participates in actions that occur at gates corresponding to outgoing transitions of the place p_i. Instances of $token_i$ are created by the process p_i when the place p_i incoming transitions are fired.

The process $P_i(k)$ models place p_i containing k tokens and is defined as the independent parallel composition of one instance of process p_i and k instances of process $token_i$. Simulation of a place p_i in PLotos is further discussed in Section 5.1 (with an example in App. B).

The whole PLotos model of the P/T-net N, with current (or initial) marking M is defined inductively. The PLotos model $N_{1,1}$ of N, restricted to place p_1, is defined as the process $P_1(M(1))$.

The model $N_{1,i}$ of N, restricted to places $p_1, ..., p_i$, is defined as the parallel composition of the process $N_{1,i-1}$ that models N restricted to places $p_1, ..., p_{i-1}$ and the process $P_i(M(i))$. These two processes are synchronized on the set of transitions that place p_i shares with places $p_1, ..., p_{i-1}$. This construction is presented formally in Section 5.2 (with an example in App. B).

5.1 Modelling of Places

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

Given place $p_i \in P$, let:

- $\Gamma^{-1}(p_i) = \{t_I : post(t_I)(p_i) > 0\}$, transitions that deposit tokens into place p_i
- $\Gamma(p_i) = \{t_O : pre(t_O)(p_i) > 0\}$, transitions that extract tokens from place p_i
- $T(X) = \bigcup_{p_i \in X}(\Gamma(p_i) \cup \Gamma^{-1}(p_i))$, 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 $\Gamma(p_i) = \{t_{O1}, ..., t_{Ov}\}$, a token inside place p_i is represented as the following PLotos process:*

process $token_i[t_{O1}, ..., t_{Ov}]$***:noexit:=***
$t_{O1}; stop[]...[]t_{Ov}; stop$
endproc

If $\Gamma(p_i)$ is empty, then the body of $token_i$ is stop. Let $\Gamma^{-1}(p_i) = \{t_{I1}, ..., t_{Iu}\}$, the place p_i is modelled by the following process:

$$
\begin{array}{l}
\textbf{\textit{process}}\ p_i[t_{I1},...,t_{Iu},t_{O1},...,t_{Ov}]\textbf{\textit{:noexit:=}} \\
\quad t_{I1};\ \underbrace{((token_i[t_{O1},...,t_{Ov}]|||\cdots|||token_i[t_{O1},...,t_{Ov}])}_{(*\ post(t_{I1})(p_i)\ times\ *)} \\
\quad\quad |||\ p_i[t_{I1},...,t_{Iu},t_{O1},...,t_{Ov}]) \\
\quad [] \cdots [] \\
\quad t_{Iu};\ \underbrace{((token_i[t_{O1},...,t_{Ov}]|||\cdots|||token_i[t_{O1},...,t_{Ov}])}_{(*\ post(t_{Iu})(p_i)\ times\ *)} \\
\quad\quad |||\ p_i[t_{I1},...,t_{Iu},t_{O1},...,t_{Ov}]) \\
\textbf{\textit{endproc}}
\end{array}
$$

Note that recursive calls to p_i are guarded and allowed in a pure interleaving. Informally, this says that when an input transition of place p_i is fired, either t_{I1} or ... or t_{Iu}, then tokens are deposited inside place p_i (instances of process $token_i$ are created). These new tokens can enable and fire transitions in $\Gamma(p_i)$. If $\Gamma^{-1}(p_i)$ is empty, then the body of p_i is stop.

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

Lemma 1 *Let, for $k \in \mathcal{N}$, $P_i(k)$ denote the place p_i containing k tokens, modelled as the PLotos processes* [4]*:*

$$
\begin{array}{ll}
P_i(0) & = p_i \\
P_i(k) & = token_i|||P_i(k-1) \quad \textit{if } k > 0
\end{array}
$$

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

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

The proof is by induction on k.

5.2 Modelling of P/T-nets

The model of a P/T-net in PLotos is also defined inductively. We first consider unlabelled P/T-nets. For $1 \leq i \leq n$, we denote by:

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

the subnet of $N = (P, T, M)$ restricted to places $\{p_1, ..., p_i\}$ where:

- $P_{1,i} = \{p_1, ..., p_i\}$
- $T_{1,i} = \{(X, act(t), Y) : t \in T \wedge X = \sum_{p \in P_{1,i}} pre(t)(p)p$
  $\wedge\ Y = \sum_{p \in P_{1,i}} post(t)(p)p \wedge (X \neq \{\} \vee Y \neq \{\})\}$

[4] *For the sake or readability, we omit gate-tuples.*

- $M_{1,i}$, 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 modelled by a PLotos process named $N_{1,i}(M_{1,i})$ defined as:*

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

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

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

Note that, for $i = 1, ..., n$, $N_{1,i}(M_{1,i})$ is not recursive (i.e constraint 3.1 and 3.2 are not violated).

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

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

The proof is by induction on i.

Let $N = (P, T, Act, M)$ be a labelled P/T-net and let $N_{1,n}(M_{1,n})$ be the PLotos model of the corresponding unlabelled P/T-net. We may add labels $a_1, ..., a_l$ in Act, of transitions in T, to the PLotos model as follows:

process $LabelledN_{1,n}(M_{1,n})[a_1, ..., a_l]$:**noexit**:=
    hide $t_1, ..., t_m$ **in** $A[a_1, ..., a_l, t_1, ..., t_m]$ $|[t_1, ..., t_m]|$ $N_{1,n}(M_{1,n})[t_1, ..., t_m]$
where
process $A[a_1, ..., a_l, t_1, ..., t_m]$:**noexit**:=
    $t_1; act(t_1); A[a_1, ..., a_l, t_1, ..., t_m]$ $[] \cdots []$ $t_m; act(t_m); A[a_1, ..., a_l, t_1, ..., t_m]$
endproc
endproc

Note that in the process $LabelledN_{1,n}$, the constraint 3.3 is not violated.

6. Conclusion

The fact that PLotos has the computational power of P/T-nets, with bisimulation equivalence, means that:

1. properties that are decidable for P/T-nets are decidable as well for PLotos, and

2. algorithms for deciding properties of P/T-nets can be adapted to PLotos.

Furthermore, the aforementioned items are obtained by minimally restricting Lotos, since P/T-nets can be modelled by PLotos. We have investigated adaptation of P/T-nets verification techniques to PLotos in Refs. [Barb 91a] and [Barb 91b].

Acknowledgements

The authors thank the members of the CRIM/BNR project for many fruitful discussions. We wish to thank Prof. Alain Finkel from École normale supérieure de Cachan who contributed to the proofs in Section 5.

References

[Azem 84] P. Azema, G. Juanole, E. Sanchis, M. Montbernard, *Specification and Verification of Distributed Systems Using Prolog Interpreted Petri Nets*, 7th International Conference on Software Engineering, 1984.

[Barb 91a] M. Barbeau, G. v. Bochmann, *Extension of the Karp and Miller Procedure to Lotos Specifications*, Computer Aided Verification'90, ACM/AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 3, 1991, pp. 103-119; and Springer- Verlag, LNCS 531, pp. 333-342.

[Barb 91b] M. Barbeau, G. v. Bochmann, *The Lotos Model of a Fault Protected System and its Verification Using a Petri Net Based Approach*, Workshop on Computer-aided verification, Aalborg, Danemark, 1991; and Springer-Verlag, LNCS 575.

[Bolo 87] T. Bolognesi, E. Brinksma, *Introduction to the ISO Specification Language Lotos*, Computer Networks and ISDN Systems, Vol. 14, No. 1, 1987, pp. 25-59.

[Bolo 90] T. Bolognesi, *A Graphical Composition Theorem for Networks of Lotos Processes*, Proceedings of Distributed Computing Systems, Paris, May-June 1990, pp. 88-95.

[Boud 85] G. Boudol, G. Roucairol, R. de Simone, *Petri Nets and Algebraic Calculi of Processes*, Advances in Petri Nets, 1985, pp. 41-58.

[Bram 83] G. W. Brams, *Réseaux de Petri: Théorie et Pratique - T.1. Théorie et analyse*, Masson, Paris, 1983.

[Cind 83] F. de Cindio, G. de Michelis, L. Pomello, C. Simone, *Milner's Communicating Systems and Petri Nets*, in: A. Pagnoni, G. Rozenberg (Eds.), Application and Theory of Petri Nets, Springer-Verlag, IFB 66, 1983, pp. 40-59.

[Dega 88] P. Degano, R. de Nicola, U. Montanari, *A Distributed Operational Semantics for CCS Based on Condition/Event Systems*, Acta Informatica, Vol. 26, 1988, pp. 59-91.

[Gara 89] H. Garavel, E. Najm, *Tilt: From Lotos to Labelled Transition Systems*, in: P. H. J. van Eijk, C. A. Vissers, M. Diaz (Eds.), The Formal Description Technique Lotos, North-Holland, 1989, pp. 327-336.

[Gara 90] H. Garavel, J. Sifakis, *Compilation and Verification of Lotos Specifications*, PSTV X, Ottawa, 1990, pp. 359-376.

[Glab 87] R. J. van Glabbeek, F. W. Vaandrager, *Petri Net Models for Algebraic Theories of Concurrency*, Proceedings of PARLE, Vol. II, LNCS 259, Springer-Verlag, 1987.

[Golt 84a] U. Goltz, A. Mycroft, *On the Relationship of CCS and Petri Nets*, in: J. Paredaens (Ed.), Proceedings of ICALP 84, LNCS 172, Springer-Verlag, 1984, pp. 196-208.

[Golt 84b] U. Goltz, W. Reisig, *CSP-Programs as Nets with Individual Tokens*, in: G. Rozenberg (Ed.), Advances in Petri Nets 1984, LNCS 188, Springer-Verlag, 1985, pp. 169-196.

[Golt 88] U. Goltz, *On Representing CCS Programs by Finite Petri Nets*, in: M. Chytil et al. (Eds.), Mathematical Foundations of Computer Science 1988, LNCS 324, Springer-Verlag, 1988, pp. 339-350.

[Gotz 86] R. Gotzhein, *Specifying Abstract Data Types with Lotos*, Proc. of PSTV VI, Montréal, 1986.

[ISO 88] ISO, *Lotos - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*, IS 8807, E. Brinksma (Ed.), 1988.

[Kasa 82] T. Kasai, R. E. Miller, *Homomorphisms Between Models of Parallel Computation*, J.C.S.S., Vol. 25, 1982, pp. 285-331.

[Marc 89] S. Marchena, G. Leon, *Transformation from Lotos Specs to Galileo Nets*, in: K. J. Turner (Ed.), Formal Description Techniques, North-Holland, 1989.

[Niel 86] M. Nielsen, *CCS and its Relationship to Net Theory*, in: W. Brauer, Advances in Petri Nets 1986, Part II, LNCS 255, Springer-Verlag, 1986.

[Olde 91] E.-R. Olderog, *Nets, Terms and Formulas: Three Views of Concurrent Processes and their Relationships*, Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press, 1991.

[Park 81] D. M. R. Park, *Concurrency and Automata on Infinite Sequences*, Proceedings of 5th GI Conf. on Theoretical Computer Science, LNCS 104, Springer-Verlag, 1981, pp. 167-183.

[Pete 81] J. L. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice Hall, 1981.

[Reis 84] W. Reisig, *Partial Order Semantics Versus Interleaving Semantics for CSP-like Languages and Its Impact on Fairness*, in: G. Goos, J. Hartmanis, 11th Colloquium on Automata, Languages and Programming, LNCS 172, Springer-Verlag, 1984, pp. 403-413.

[Taub 89] D. Taubner, *Finite Representation of CCS and TCSP Programs by Automata and Petri Nets*, LNCS 369, Springer-Verlag, 1989.

Appendix A: Basic Lotos

A.1. Syntax of Basic Lotos

We assume that Basic Lotos specifications are constructed as follows:

specification ::= **specification** specification-identifier formal-parameter-list
behavior
behavior-expression
[local-definitions]
endspec

formal-parameter-list ::= [gate-tuple] ":" functionality

gate-tuple ::= "[" gate-identifier-list "]"

gate-identifier-list ::= gate-identifier { "," gate-identifier }

functionality ::= **exit** | **noexit**

behavior-expression ::=

stop
gate-identifier ";" behavior-expression
behavior-expression "[]" behavior-expression
process-identifier [gate-tuple]
behavior-expression "|||" behavior-expression
behavior-expression "|[" gate-identifier-list "]|" behavior-expression
exit
behavior-expression ">>" behavior-expression
behavior-expression "[>" behavior-expression
hide gate-identifier-list **in** behavior-expression

local-definitions ::= **where** process-definition { process-definition }

process-definition ::=
process process-identifier formal-parameter-list ":="
behavior-expression
endproc

specification-identifier ::= identifier

process-identifier ::= identifier

gate-identifier ::= identifier

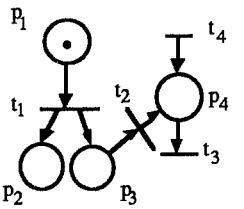
identifier ::= letter [{ normal-character | "_" } normal-character]

normal-character ::= letter | digit

In a "process-definition", the term "behavior-expression" is called the defining behavior-expression of the process named "process-identifier".

Appendix B: The simulation of a P/T-net in PLotos

P/T-net



Translation of places

process $p_1[t_1]$:noexit:=
    stop
endproc

process $p_2[t_1]$:noexit:=
    t_1; token$_2$[ ]|||$p_2[t_1]$
endproc

process $p_3[t_1, t_2]$:noexit:=
    t_1; token$_3[t_2]$|||$p_3[t_1, t_2]$
endproc

process $p_4[t_2, t_3, t_4]$:noexit:=
    t_2; token$_4[t_3]$|||$p_4[t_2, t_3, t_4]$
[ ]
    t_4;token$_4[t_3]$|||$p_4[t_2, t_3, t_4]$
endproc

process token$_1[t_1]$:noexit:=
    t_1; stop
endproc

process token$_2$[ ]:noexit:=
    stop
endproc

process token$_3[t_2]$:noexit:=
    t_2;stop
endproc

process token$_4[t_3]$:noexit:=
    t_3;stop
endproc

Lotos model of the P/T-net

process $N_{1,1}((1))[t_1, t_2, t_3, t_4]$:noexit:= $P_1(1)$ endproc

process $N_{1,2}((1,0))[t_1, t_2, t_3, t_4]$:noexit:= $P_2(0)|[t_1]|N_{1,1}((1))[t_1, t_2, t_3, t_4]$ endproc

process $N_{1,3}((1,0,0))[t_1, t_2, t_3, t_4]$:noexit:= $P_3(0)|[t_1]|N_{1,2}((1,0))[t_1, t_2, t_3, t_4]$
endproc

process $N_{1,4}((1,0,0,0))[t_1, t_2, t_3, t_4]$:noexit:= $P_4(0)|[t_2]|N_{1,3}((1,0,0))[t_1, t_2, t_3, t_4]$
endproc

where

$P_1(1) =$ token$_1[t_1]$|||$p_1[t_1]$
$P_2(0) = p_2[t_1]$
$P_3(0) = p_3[t_1, t_2]$
$p_4(0) = p_4[t_2, t_3, t_4]$