Extension of the Karp and Miller Procedure to Lotos Specifications [1]

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

Abstract

In a companion paper [Barb 90a,b], we proposed a Place/Transition-net (P/T-net) semantics for a subset of Lotos. This subset is such that Lotos specifications can be translated into finite structure Petri nets. It is therefore possible to apply P/T-net verification techniques since they require finite structure. It this paper, we demonstrate that it is possible to apply P/T-net verification methods without an a priori construction of the P/T-net associated to the Lotos specification to be analysed. In particular we consider a well known reachability analysis technique for P/T-nets, namely, the Karp and Miller procedure.

1 Introduction

The goal of this work is to investigate verification techniques for basic Lotos specifications [Bolo 87a]. Basic Lotos is presented in § 4. It can be demonstrated that this language has the computational power of Turing machines. Therefore, generally non-trivial properties are undecidable.

Our verification method is based on Petri nets. Petri net verification techniques are transfered to Lotos. So far, two transfer approaches have been proposed. A first approach consists of translating Lotos specifications into Petri nets and evaluating the properties on the equivalent Petri net models. Presently, there are at least two tools based on this approach ([March 89] and [Gara 90]).

We found that the Lotos to Petri nets translation step is, in many cases, as complex, in terms of time and space, as the verification step itself. In this paper, we propose a second approach which involves no translation from one formalism to another. We adapt Place/Transition-net (P/T-net) verification algorithms to Lotos. We consider a well known P/T-net reachability analysis technique, the Karp and Miller procedure [Karp 69].

The main difficulty in these approaches is to make shure that the P/T-net, that models the Lotos specification, has a finite structure. That is, a bounded number of places and transitions. We model Lotos local process states as Petri net places. We put syntactical constraints so that the number of local process states is bounded. Moreover, the number of alternatives to make a transition, from a given local state, is also bounded. Consequently, the Petri net has a finitre structure. However, these constraints do not bound the number of identical parallel processes. Therefore, finite structure Petri nets does not necessarily mean finite state systems. Consequently, the "Lotos" that can be analyzed with our method is computationaly more powerful than the "Lotos" that can be verified with finite state/transition system based methods (e.g. [Bolo 87b]).

Petri nets can represent, with finite structures, infinite state systems. This means that classical finite state system reachability analysis [Boch 78] is not applicable. Karp and Miller trees and graphs are finite and partial representations of, in general infinite, Petri net reachability trees and graphs. P/T-nets are introduced in § 2. Reachability analysis for P/T-nets is discussed in § 3. Modelling of Lotos by P/T-nets is presented in § 5. The extension of Petri net reachability analysis to Lotos is discussed in § 6.

[1] This work has been funded by the Natural Sciences and Engineering Research Council of Canada, the Centre de recherche informatique de Montréal and Bell Northen Research.

2 P/T-nets

We slightly deviate from the usual notation for P/T-nets [Pete 81]. We represent a P/T-net 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 has a **finite structure** 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. A multi-set is a set that can contain multiple instances of the same element. An element $t = (X, a, Y) \in T$ is also 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 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.

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. A marking M is also denoted as a n-tuple $(M(p_1), ..., M(p_n))$. Instances of the element p_i are also called tokens inside place p_i. The operators $\leq$, $+$ and $-$ denote respectively multi-set inclusion, summation and difference.

$pre(t)(p)$ 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 $M(t > M'$.

A P/T-net is illustrated in Fig. 1, places are shown as circles, transitions as bars and tokens as dots inside places. There is a directed edge from place p_i (transition t_j) to transition t_j (place p_i) iff $p_i \in pre(t_j)$ $(p_i \in post(t_j))$. If $pre(t)(p) > 1$ $(post(t)(p) > 1)$ we may label the corresponding edge with the value of $pre(t)(p)$ $(post(t)(p))$. The initial marking of this particular net can be denoted as the multi-set $\{p_1, p_2\}$ or as the 6-tuple $(1, 1, 0, 0, 0, 0)$.

3 Reachability Analysis for P/T-nets

With respect to Holzmann's classification [Holz 89], the Karp and Miller tree construction procedure is a *stack search strategy*. It is a depth-first technique that minimizes memory usage at the expense of run time. The whole state space does not have to be maintained in memory, only the path starting from the initial marking to the current marking. In general, Petri nets are not finite state systems. The Karp and Miller tree is also called the **coverability tree** because for every reachable marking M of a Petri net, there exists a marking M' in the Karp and Miller tree such that $M \leq M'$.

Definition 1 *The* **coverability tree** *(CT) associated to a P/T-net $N = (P, T, Act, M_0)$ is a tree, where vertices are labelled with markings of N and edges are labelled with elements in T. The CT can be recursively defined as follows:*

1. *The root is labelled with M_0;*

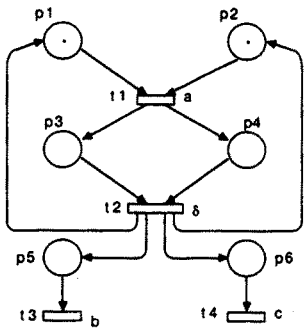


Figure 1: A P/T-net

2. *Let x denote a vertex with label M,*

   (a) if there exists a vertex y such that $y \prec x$ and $label(y) = M$, then x is a leaf;

   (b) else, the successors of x are in one-to-one correspondence with the elements of the set:

$$S = \{(t, M') : t \in T \wedge M(t > M'\}$$

   Let $(t, M') \in S$, we create the successor vertex z and the edge (x, t, z). The label of z is determined as follows: For $i = 1, ..., n$ (n is the number of places),

   i. if there exists a vertex y such that $y \prec z$, $label(y) \leq M'$ and $label(y)(p_i) < M'(p_i)$ then $label(z)(p_i) = \omega$;

   ii. else, $label(z)(p_i) = M'(p_i)$;

The symbol ω denotes infinity. If the whole CT is maintained in memory, the **coverability graph** (CG) can be obtained by merging vertices with identical labels. In that case, more memory is required but full connectivity information is stored and can be used to analyse loops.

Example 1: For the net of Fig. 1, the CG is shown in Fig. 2.

With the CT and the CG, the following six problems [Fink 90] become decidable:

1. *Termination* Is the reachability tree finite?
2. *Finiteness* Is the reachability set finite?
3. *Coverability* Given a marking M, is there a reachable marking M' such that $M \leq M'$?
4. *Quasi-liveness* Given an action a, is there a reachable marking M such that a is executed from M ?
5. *Boundedness* Is the number of tokens in a given place bounded?
6. *Regularity* Is the Petri net language recognizable by a finite state automaton?

The language of a Petri net is the set of transition sequences starting from the initial marking. If the language is regular the Petri net can be simulated by a finite state automaton. The Karp and Miller tree and graph constructions do not necessarily detect every deadlock in a nonfinite state system.

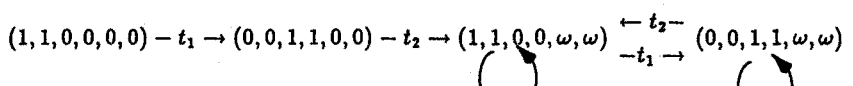


Figure 2: Coverability Graph

4 Lotos

A basic 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	$B_1 \|\|\| B_2$
General parallel composition	$B_1 \|[g_1, ..., g_n]\| B_2$
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 semantics of Lotos is given in [Bolo 87a].

This subset of Lotos has the computational power of Turing machines (proved in [Barb 90a]). We conclude that nontrivial properties are generally undecidable. P/T-nets (with finite structures) do not have the computational power of Turing machines as Lotos does. In the rest of this section we define a subset of Lotos, PLotos, that can be modelled by finite structure P/T-nets, and conversely into which P/T-nets can be simulated. The mapping from PLotos to P/T-nets is introduced in the next section whereas P/T-nets simulation in PLotos is discussed in [Barb 90a,b].

We assume that PLotos specifications satisfy the following constraints:

1. *Guarded recursive processes.* A process instantiation term is guarded if it is in the scope of a prefixing operator ";" or in the right sub-expression B_2 of a sequential composition $B_1 >> B_2$ or of a disabling $B_1 [> B_2$.

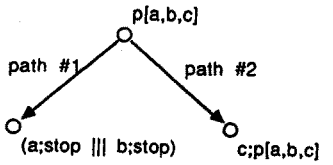
2. *No combination of recursion and general parallel composition.* The general parallel operator "$|[g_1, ..., g_n]|$" is not allowed on recursive paths.

3. *Tail recursion.* The process in which $B_1 >> B_2$ (or $B_1 [> B_2$) is defined may not be called from sub-expression B_1. And, in B_1 combination of recursion and parallelism is not allowed.

4. *"Noexit" functionality in pure interleaving.* Operands B_1 and B_2 in parallel composition $B_1 |||B_2$ must have the *noexit* functionality.

Paths are defined by choice alternatives and can be illustrated as a tree with behavior expressions labelling nodes, as in the following example:

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



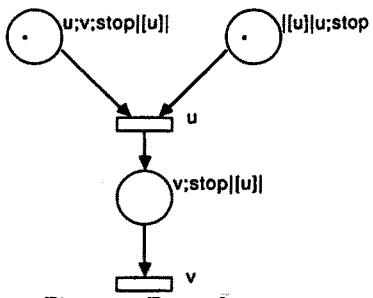


Figure 3: Example

The path number 1 is not recursive whereas the path number 2 is because the process p is recursively called at node labelled "$c; p[a, b, c]$". A process that does not contain parallel operator "$|||$" and "$|[g_1, ..., g_n]|$" on recursive paths can be modelled by a finite state system. If the operator "$|[g_1, ..., g_n]|$" is disallowed on recursive paths, whereas "$|||$" is allowed with functionality *noexit* operands, the system is not finite state but can still be represented by a finite structure P/T-net. The **functionality** of a behavior B is equal to *exit iff* it terminates with the successful termination action δ, otherwise it is equal to *noexit*, [Bolo 87a].

PLotos specifications are rewritten into simpler forms. Non-recursive paths 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 $|[g_1, ..., g_n]|_k$.

5 Modelling of Lotos with P/T-nets

The mapping from Lotos to P/T-nets is based on the work of Olderog [Olde 87] for CCSP. In general, a Lotos behavior expression B represents the composition of several concurrent components. The expression B is explicitly decomposed into its parallel components that become tokens when this behavior is activated. 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 component state. Lotos gates are modelled by Petri net transitions. Tokens, contained in transition input places, represent components synchronized on this gate. Tokens deposited into 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 structure 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 components is denoted as the set $\{u; v; stop|[u]|, |[u]|u; stop\}$. In this syntax, we represent explicitly the fact that components are coupled on gate u by concatenating the symbol $|[u]|$ to the right of $u; v; stop$ and to the left of $u; stop$.

Fig. 3 depicts the corresponding P/T-net model. Places modelling states of components are labelled with the corresponding expression components. Transitions are labelled with gate names. The "stop" expression represents inaction and does not appear in the P/T-net. In our construction,

edges from place to transition are always one valued and every place has a distinct label. We unambiguously denote a place by its label. The above "u" labelled transition is 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 expressions, and ii) a system of inference rules. The head of each rule matches a term of the form:

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

A rule can be used to infer, as a function of place label structures, 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 $M_1.|[S]|_k - a \rightarrow M_1'.|[S]|_k$

has been 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 first introduce the decomposition function in §5.1, then we present in §5.2 the inference rules.

5.1 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 multi-sets of place labels.

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,

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

where

- B_p represents the body of process definition p,
- in (d4), $g_1, ..., g_n$ 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. The restriction to guarded recursive processes 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. We show in [Barb 90a] 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.

5.2 Inference Rules

This section presents the inference rules of the mapping from PLotos to P/T-nets. The P/T-net $N = (P, T, Act, M_0)$ associated to a PLotos behavior B is such that:

1. $M_0 = dec(B)$, $(\forall p)[M_0(p) > 0 \Rightarrow p \in P]$,
2. if $X \subseteq P$ and $X - a \rightarrow Y$ then $(\forall p)[Y(p) > 0 \Rightarrow p \in P]$, $(X, a, Y) \in T$, $a \in Act$, and
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 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 place 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 $\{B_1 >> B_2\} - \delta \rightarrow dec(B_2)$

(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 $\{B_1[> B_2\} - \delta \rightarrow dec(B_1)$

(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 $hide\ S\ in.M_1 - \mathrm{i} \rightarrow hide\ S\ in.M_1'$
(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 Lotos semantics in [Bolo 87a]. Consistency of this P/T-net interpretation of Lotos is formally proved in [Barb 90a].

Example 2: Consider the following specification[2]:

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

The P/T-net derived from this specification is identical to the net depicted in Fig. 1 with:

$p_1 = a; exit >> (b; stop|||p2[a,b])|[a]|$ $p_2 = |[a]|a; exit >> (c; stop|||p2[a,c])$
$p_3 = exit >> (b; stop|||p2[a,b])|[a]|$ $p_4 = |[a]|exit >> (c; stop|||p2[a,c])$
$p_5 = b; stop|[a]|$ $p_6 = |[a]|c; stop$

6 Coverability Graphs for Lotos

Given a Lotos specification, it is possible to construct an equivalent P/T-net model by successive applications of the above inference rules. This P/T-net then becomes the input of the reachability analysis algorithm to evaluate the properties. In the worst case, the P/T-net can have more vertices and edges than the coverability graph. In our approach we skip the intermediate Lotos to P/T-nets translation step. We derive the coverability graph directly from the Lotos specification then properties are evaluated.

The syntax of Lotos coverability graphs slightly deviates from the usual syntax for Karp and Miller graphs. Markings are multi-sets of Lotos behavior expression components. We label the root of the graph with the decomposition of the Lotos expression that represents the initial behavior. For example, the decomposition of the initial behavior in Example 2 yields a state represented as the following box:

$$\boxed{\begin{array}{l} 1/a; exit >> (b; stop|||p2[a,b])|[a]| \\ 1/|[a]|a; exit >> (c; stop|||p2[a,c]) \end{array}}$$

Every line in the box defines the number of instances of one behavior expression component type in the current state. In case there is an infinite number of occurrences, the expression component is paired with the ω symbol.

We go from one marking to another by application of the inference rules. An inference rule is applicable from one marking if a finite subset of the expression component multi-set matches the preset of the transition in the head of the rule. The successor state is obtained by removing this

[2]For the sake of simplicity, labelling of the $|[a]|$ operator is omitted in this example.

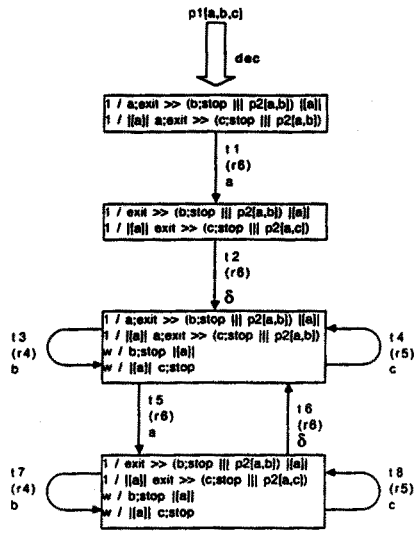


Figure 4: Coverability graph of Example 2

preset from the current state and adding the postset defined by the transition (reformulation of the usual P/T-net firing rule). Every edge is labelled with the number of the inference rule which has been applied to derive the transition and the action name of the transition. The coverability graph of Example 2 is illustrated in Fig. 4.

The six problems stated in § 3 can be solved as follows:

1. *Termination* The reachability tree is infinite if there is at least one circuit in the CG.

2. *Finiteness* The reachability set is infinite if the CG contains one marking and one process p paired with ω.

3. *Coverability* Given a marking M, there exists a reachable marking M' such that $M \leq M'$, if there exists in the CG a marking M'' with $M \leq M''$.

4. *Quasi-liveness* The action a is quasi-live if there exists an edge in the CG labelled with a.

5. *Boundedness* Instantiation of process p is unbounded if there exists a marking M in which $dec(p)$ is paired with ω.

6. *Regularity* The language is regular if every elementary circuit of the CG is labelled by a sequence of transitions $t_1, t_2, ..., t_n$ such that for every place p:

$$post(t_1)(p) - pre(t_1)(p) + post(t_2)(p) - pre(t_2)(p) + \cdots post(t_n)(p) - pre(t_n)(p) \geq 0$$

In general, conclusions can be easily drawn from visual inspection of the CT and the CG. For instance, the language of Example 3 is not regular since:

$$post(t_3)(b; stop|[a]|) - pre(t_3)(b; stop|[a]|) = -1 < 0$$

7 Conclusion

We have presented a reasonable subset of Lotos that can be verified using Petri net reachability analysis techniques. Our method does not require explicit translation from Lotos to Petri nets. Analysis is performed in the Lotos world to which the Karp and Miller procedure is extended. To cope with state space explosion, MCT and MCG can be computed to solve the aforementioned six problems. We experimented the MCT construction procedure and obtained satisfactory results. The MCTs were several times less complex than the nonminimal CTs.

8 References

[Barb 90a] Barbeau, M., Bochmann, G. V. *Deriving Analysable Petri Nets from Lotos Specifications*, Research report no. 707, Dept. d'IRO, Université de Montréal, 1990.

[Barb 90b] Barbeau, M., Bochmann, G. V. *Verification of Lotos Specifications: A Petri Net Based Approach*, Proc. of Canadian Conf. on Elec. and Computer Eng., Ottawa, 1990.

[Barb 89] Barbeau, M., Bochmann, G. V. *Experiences with Automated Verification Tools: Application to Discrete Event Systems*, Proc. of Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, 1989.

[Boch 78] Bochmann, G. V. *Finite State Description of Communication Protocols*, Computer Network 2, (361-372), 1978.

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

[Bolo 87b] Bolognesi, T., Smolka, S. A. *Fundamental Results for the Verification of Observational Equivalence: A Survey*, Proc. of PSTV VII, Zurich, 1987.

[Fink 90] Finkel, A. *A Minimal Coverability Graph for Petri Nets*, Proc. 11th Int. Conf. on Application and Theory of Petri Nets, Paris, 1990.

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

[Holz 89] Holzmann, G. J. [1989]. *Algorithms for Automated Protocol Validation*, Proc. of Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, 1989.

[Karp 69] Karp, R. M., Miller, R. E. *Parallel Program Schemata*, J. Computer and System Sciences 3, (147-195), 1969.

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

[Olde 87] Olderog, E.-R. *Operational Petri Net Semantics for CCSP*, LNCS 266, Springer-Verlag, 1987.

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