THE LOTOS MODEL OF A FAULT PROTECTED SYSTEM AND ITS VERIFICATION USING A PETRI NET BASED APPROACH

Michel Barbeau
Département de mathématiques et d'informatique
Université de Sherbrooke, Sherbrooke, Canada J1K 2R1
and
Gregor v. Bochmann
Département d'IRO, Université de Montréal
C.P. 128, Succ. "A", Montréal, Canada H3C 3J7

Abstract

Having introduced a novel Petri net based method for the verification of Lotos specifications [Barb 90a], this paper demonstrates its practical interest. Contrary to other similar Petri net based techniques, our approach avoids to build the whole Petri net from the Lotos specification before verification. In contrast to finite automata based methods, our method can analyse Lotos systems with unbounded state spaces. Our method is founded on a Place/Transition-net Lotos semantics. The method is applied to the verification of the Lotos model of fault protected system.

1. Introduction

Lotos [ISO 88] is a specification language for protocols and distributed applications. In this paper, we consider a Lotos specification which models the fault protection aspect of a small system. This system consists of two unreliable pieces of equipment and a standby equipment. Initially, the system is working and protected. When a failure occurs, the standby is substituted for the failed piece of equipment and the whole system moves to the "working-unprotected" state. If a second equipment failure occurs, the whole system moves to the "failed" state. To describe this model, only a subset of Lotos called *Basic Lotos* is required. Basic Lotos is introduced in § 2 whereas the Lotos model of the fault protected system is presented in § 3.

The dynamic semantics of Lotos is defined formally. This means that formal verification is possible. Our verification method is based on Petri net theory. Petri net verification techniques are transferred 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 [Gara 90, and Marc 89].

We proposed a second approach which involves no translation from one formalism to another [Barb 90a]. We adapted to Lotos the well known Place/Transition-net (P/T-net) reachability analysis technique, namely, the Karp and Miller procedure [Karp 69]. In § 4 of this paper, we apply this technique for the verification of the Lotos specification of a fault protected system.

2. Basic Lotos

Lotos consists of two sub-languages. First, there is a sub-language, based on Act One [Ehri 85], dealing with the description of data structures. Second, there is a sub-language, based on CCS [Miln 89], concerned with the description of dynamic discrete event processes. Basic Lotos is a core subset of the second sub-language. 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 formal semantics of Lotos is given in [ISO 88].

3. The Lotos Specification

To present the specification of the fault protected system, we adapted to Lotos Knuth's literate programming style [Knut 84].

1. The purpose of this Lotos specification is to define the model of a system which is composed of two pieces of equipment that can fail. The system is protected by one piece of standby equipment that can fail as well. When either two pieces of equipment or one piece of equipment and the standby are failed, the whole system is failed. The Lotos specification describes the failure as well as the repair of pieces of equipment.

2. For readability, we define the following lists of gates:

$L_0 \equiv fl1, fl2, rp1, rp2, sfl, srp1, srp2, swsucc, swfl$

$L_1 \equiv res, fl1, fl2, rp1, rp2, sfl, srp1, srp2, swsucc, swfl$

$L_2 \equiv fl1, rp1, sfl, srp1, srp2, swsucc, swfl$

$L_3 \equiv fl1, fl2, rp1, rp2, swsucc, swfl$

Lotos gates are synchronization points where atomic events occur. We interpret the above gates as follows:

res (restart): The system had a breakdown and is restarted in the protected state.

*fl*1 (failure1): A piece of equipment fails while the standby is available.

*fl*2 (failure2): A piece of equipment fails while the standby is not available.

*rp*1 (repair1): A piece of equipment is repaired while the system is working but unprotected.

*rp*2 (repair2): A piece of equipment is repaired while the system is failed.

sfl (standby failure): The standby fails, the system moves to the failed state.

*srp*1 (standby repair): The standby is repaired while the system is unprotected.

*srp*2 (standby repair): The standby is repaired while the system is failed.

swsucc (switch successful): This event models the successful substitution of the standby for a failed piece of equipment.

swfl (switch failure): This event models the unsuccessful substitution of the standby for a failed piece of equipment.

3. Here is an outline of the Lotos specification:

specification ProtectedSystem [res]:**noexit**
behavior
< Initial system behavior 4 >
where
< Definition of the system component 5 >
< Definition of the piece of standby 6 >
< Definition of a piece of equipment 7 >
endspe

In accordance with the classification of Vissers et al. [Viss 87], the style of this Lotos specification is *resource-state* oriented. In the resource oriented style, the behavior of the system is defined as the composition of interacting components. Actually, we define the components "system", "piece of standby" and "piece of equipment". The components are specified using a state oriented approach. Every entity internal state is modelled as one Lotos process.

4. The fault protected system initial behavior is defined as the parallel composition of four process instances. The instances correspond to the initial states of one system component, one piece of standby component and two pieces equipment components. Internal interactions are hidden.

< Initial system behavior 4> ≡
hide L_0 **in** ((protected [L_1]
|[L_2]|
available [L_2])
|[L_3]|
(equip_work [L_3] ||| equip_work [L_3]))

5. The initial state of the system component is protected and may change to switching (i.e., sys_switch) if a piece of equipment detects a failure. The standby is switched for the failed piece of equipment. The standby then does the work of the original piece of equipment. Next, there are two alternatives. If the standby does not detect a problem, the original piece of equipment is declared

as failed and the switching phase is complete. However, if the standby also detects a failure, the conclusion is that the malfunction origin is not the piece of equipment. And, the system moves to the breakdown state. In the latter situation, the system requires service and may then be restarted in the protected state.

The system status may change from unprotected to failed (sys_failed) if either another piece of equipment fails or the standby fails. Two other alternatives are repair of a piece of equipment or repair of the standby. The system remains in the failed state until either a piece of equipment or the standby is repaired. Five processes are defined to model the five internal states of a "system" entity.

< Definition of the system component 5 > ≡
process protected [L_1]:**noexit**:=
fl1;sys_switch [L_1]
where
process breakdown [L_1]:**noexit**:=
res;protected [L_1]
endproc
process sys_switch [L_1]:**noexit**:=
swsucc;unprotected [L_1] [] swfl;breakdown [L_1]
endproc
process unprotected [L_1]:**noexit**:=
fl2;sys_failed [L_1] [] sfl;sys_failed [L_1] [] rp1;protected [L_1] [] srp1;protected [L_1]
endproc
process sys_failed [L_1]:**noexit**:=
rp2;unprotected [L_1] [] srp2;unprotected [L_1]
endproc
endproc

6. The standby entity is initially inactive and available. When a piece of equipment failure occurs, it becomes active. If the switching phase succeeds, it behaves as a piece of equipment. Four processes are defined to model the four internal states of the standby.

< Definition of the piece of standby 6 > ≡
process available [L_2]:**noexit**:=
fl1;stan_switch [L_2]
where
process stan_switch [L_2]:**noexit**:=
swsucc;stan_work [L_2] [] swfl;available[L_2]
endproc
process stan_work [L_2]:**noexit**:=
sfl;stan_failed [L_2] [] rp1;available [L_2]
endproc
process stan_failed [L_2]:**noexit**:=

srp1;available [L_2] [] srp2;stan_work [L_2]
endproc
endproc

7. A piece of equipment is initially working and has four internal states modelled by four Lotos processes.

< Definition of a piece of equipment 7 > ≡
process equip_work [L_3]:**noexit**:=
fl1;equip_switch [L_3] [] fl2;equip_failed [L_3]
where
process equip_switch [L_3]:**noexit**:=
swsucc;equip_failed [L_3] [] swfl;equip_work [L_3]
endproc
process equip_failed [L_3]:**noexit**:=
rp1;equip_work [L_3] [] rp2;equip_work [L_3]
endproc
endproc

8. The goal of this model is to mimic the behavior of a real fault protected system. We expect some properties from this model which are listed below:

P1: If the system is in the protected state then the standby is available and two pieces of equipment are working.

P2: If the system is in the unprotected state then either: (1) the standby and one piece of equipment are working, or (2) two pieces of equipment are working.

P3: If the system is in the failed state then either: (1) two pieces of equipment are failed, or (2) the standby and one piece of equipment are failed.

P4: If the system is in the breakdown state then the standby is available and two pieces of equipment are working.

Correctness of the specification means that the aforementioned properties are satisfied. The assessment of correctness is the topic of next section.

4. Verification Based on a P/T-net Semantics

Our approach is based on a P/T-net semantics for Lotos. That is, the execution of Lotos specifications is modelled by P/T-nets. We first outline in § 4.1 the P/T-net semantics for Lotos. The verification method itself is discussed in § 4.2 and § 4.3.

4.1. Overview of the P/T-net Semantics

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** is X, its **postset** is Y and **action** is a. The operators $\leq$, $+$ and $-$ denote respectively multi-set inclusion, summation and difference. A Petri net marking is also a multi-set. We denote by $M(p_i)$ $(X(p_i)/Y(p_i))$ the number of instances of the element p_i in the multi-set M (preset/ postset). Instances of the element p_i are also called tokens inside place p_i.

Full Basic Lotos cannot be modelled by finite structure P/T-nets [Barb 90b]. There is a theoretical limitation. Basic Lotos has the computational power of Turing machines, this is not the case of P/T-nets. Furthermore, data structures and their operations are difficult to model concisely into P/T-nets. We identified a subset of Basic Lotos, called PLotos, that can be modelled by P/T-nets, with bisimulation equivalence [Miln 83]. It consists of Basic Lotos plus easy to verify syntactical constraints.

Definition of PLotos

Let p_1 be a process and B_{p_1} the body of the definition of p_1. We say that p_1 **calls** p_2 if instantiation of p_2 is a subterm of B_{p_1}. This relation is denoted as:

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

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

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

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*. The **execution paths** of behavior expressions are defined, as usual, by selective statements, namely, choice terms for Basic-Lotos.

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

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

2. *Noexit functionality in independent parallelism:* Operands B_1 and B_2 in a parallel composition $B_1 ||| B_2$ must have the *noexit* functionality.

3. For every pair $(p_1, p_2) \in M$ we must have:

   3.1. The general parallel operator $||[g_1, ..., g_n]||$ does not occur on the execution path of B_{p_1} which leads to instantiation of p_2.

   3.2. If $B_1 >> B_2$ (or $B_1[> B_2$) is a sub-term of B_{p_1} then instantiation of p_2 is not a sub-term of B_1, also B_1 must have the *exit* functionality.

Stronger constraints, than constraints 2 and 3.1, are stated in [Gara 90] which disallow mutual recursion in sub-terms of the form "$B_1|||B_2$" and "$B_1|[g_1, ..., g_n]|B_2$". They are such that the control can be modelled by a finite state automaton. In our case the operator "$|[g_1, ..., g_n]|$" is disallowed on recursive paths, but mutual recursion is possible in sub-terms of the form "$B_1|||B_2$", with functionality *noexit* operands, the control is not finite state but can still be represented by a finite structure P/T-net.

It is possible to simulate an arbitrarily large stack if the constraint 3.2 is not satisfied. Arbitrarily large stacks cannot be simulated by finite structure P/T-nets.

The Lotos to P/T-nets mapping is based on the work of Olderog [Olde 87] and has two aspects i) a decomposition function, and ii) a set of inference rules.

A Lotos behavior expression B generally represents the composition of several concurrent components. B is decomposed into a multi-set of behavior expressions which, when B is activated, may be interpreted as P/T-net tokens. P/T-net tokens are unstructured elements. The place in which a token is contained is named after the concurrent component that it denotes. For instance, let $B = u; v; stop|[u]|v; stop$, then:

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

This means that instantiation of B is modelled as tokens deposited into places labelled $u; v; stop|[u]|$ and $|[u]|v; stop$.

Inference rules are used to infer, from subsets of concurrent components, executable transitions. For instance, by application of appropriate inference rules we can infer that the following transition is executable from $dec(B)$:

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

Moreover, from the component set $\{u; stop|[u]|\}$ the following transition is executable:

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

The head of each rule is a term of the form:

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

Each rule can be used to infer, as a function of component 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 second above transition: We substituted $\{u; stop\}$, 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. The decomposition function and the inference rules are defined in [Barb 90a, Barb 90b].

Finally, note that we have also shown that conversely P/T-nets can be simulated in our Lotos subset, with language equality equivalence [Barb 90b]. This means that our Lotos subset and P/T-nets have the same computational power. Proofs of correctness can also be found in [Barb 90b].

4.2. Karp and Miller Graphs for Lotos

Given a Lotos specification, it is possible to construct an equivalent P/T-net model by successive applications of the inference rules discussed above. This P/T-net then becomes the input of the reachability analysis algorithm to evaluate the properties. In our approach, we skip the intermediate Lotos to P/T-nets translation step. We derive the reachability graph directly from the Lotos specification, then properties are evaluated. The syntax of the reachability graph slightly deviates from the usual syntax for Karp and Miller graphs. We first discuss the derivation of Karp and Miller graphs for Lotos.

P/T-nets as well as PLotos are not finite state systems. This means that classical finite state/transition system reachability analysis [Boch 78] does not work for P/T-nets. Karp and Miller graphs are finite representations of in general infinite reachability graphs. As other reachability graphs, vertices are labelled with states and edges with transitions. However, a single state in the coverability graph may represent an unbounded number of "equivalent" reachable states.

In our case, states 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 system behavior. For example, the initial behavior B_0 of the protected system is[1]:

$$(protected[L_1]||[L_2]||available[L_2])||[L_3]|(equip\_work[L_3]|||equip\_work[L_3])$$

The decomposition of B_0, $dec(B_0)$, yields the state represented as the following box:

$1/protected[L_1]\|[L_2]\|\|[L_3]\|$
$1/\|[L_2]\|available[L_2]\|[L_3]\|$
$2/\|[L_3]\|equip\_work[L_3]$

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

We go from one state to another by application of the inference rules. An inference rule is applicable from one state 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 preset from the current state and adding the postset defined by the transition (reformulation of the usual P/T-net transition 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. A Karp and Miller graph is also called a coverability graph because for every reachable state s of the

[1] We choose the unhidden version of the behavior in order to obtain meaningful transition labels in the graph.

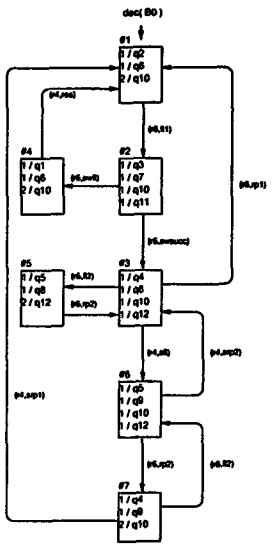

Figure 1: Fault protected system coverability graph

model, there exists a state s' in the Karp and Miller graph such that $s \leq s'$ (the operator $\leq$ denotes multi-set inclusion). The Karp and Miller procedure adapted to Lotos is precisely defined in [Barb 90a].

The coverability graph of the fault protected system is shown if Fig 1. For conciseness, Lotos expression components are shown as $q_1, ..., q_{12}$ which are equivalent to:

$q_1 \equiv \mathit{breakdown}[L_1]|[L_2]|||[L_3]|$, system in the breakdown state.

$q_2 \equiv \mathit{protected}[L_1]|[L_2]|||[L_3]|$, system in the protected state.

$q_3 \equiv \mathit{sys\_switch}[L_1]|[L_2]|||[L_3]|$, system in the switching phase.

$q_4 \equiv \mathit{unprotected}[L_1]|[L_2]|||[L_3]|$, system in the unprotected state.

$q_5 \equiv \mathit{sys\_failed}[L_1]|[L_2]|||[L_3]|$, system in the failed state.

$q_6 \equiv |[L_2]|\mathit{available}[L_2]|[L_3]|$, standby is available.

$q_7 \equiv |[L_2]|\mathit{stan\_switch}[L_2]|[L_3]|$, standby in the switching phase.

$q_8 \equiv |[L_2]|\mathit{stan\_work}[L_2]|[L_3]|$, standby in the working state.

$q_9 \equiv |[L_2]|\mathit{stan\_failed}[L_2]|[L_3]|$, standby in the failed state.

$q_{10} \equiv |[L_3]|\mathit{equip\_working}[L_3]$, a working piece of equipment.

$q_{11} \equiv |[L_3]|\mathit{equip\_switch}[L_3]$, a piece of equipment in the switching phase.

$q_{12} \equiv |[L_3]|\mathit{equip\_failed}[L_3]$, a failed piece of equipment.

4.3. Verification of Properties

Verification consists of determining if the model is "correct" and satisfies certain properties. The properties can be classified into two groups: (1) general properties, and (2) specific properties. General properties are independent of the goal of the modelled system whereas specific properties are strongly related to the system function. In the field of communication protocols, deadlock freeness and conformity with the service are examples of, respectively, general and specific properties.

By examination of Fig. 1, we can identify the following general properties: (1) The fault protected system is *finite state* because there is no marking that contains a component paired with the ω symbol, (2) for the same reason the *bounded process instantiation* property is satisfied, and (3) every state has successors, consequently the model contains *no deadlock*.

The goal of our model is to mimic the behavior of a real system, with respect to the fault protection aspect. The real system properties are listed under item 8 in § 3, they are evaluated with the help of the coverability graph. First, these specific properties have to be expressed formally. We use logical formulae, i.e. assertions, to formalize the properties. The expression "$s(q_x)$" denotes the number of occurrences of the Lotos expression component q_x in state s. The expression "$s(q_x) = n$" is true if state s contains n occurrences of q_x, otherwise the expression evaluates to false. We sometimes consider $s(q_x)$ as a predicate which is true if "$s(q_x) > 0$" and false if "$s(q_x) = 0$". We denote as RS the set of reachable states. The above four properties are formally stated as follows:

P1: $\forall s \in RS.s(q_2) \Rightarrow s(q_6) \wedge s(q_{10}) = 2$

P2: $\forall s \in RS.s(q_4) \Rightarrow [(s(q_8) \wedge s(q_{10})) \vee s(q_{10}) = 2]$

P3: $\forall s \in RS.s(q_5) \Rightarrow [s(q_{12}) = 2 \vee (s(q_9) \wedge s(q_{12}))]$

P4: $\forall s \in RS.s(q_1) \Rightarrow s(q_6) \wedge s(q_{10}) = 2$

To verify the properties, we must check that every reachable state satisfies the assertions. A quick visual inspection of the coverability graph of Fig. 1 reveals that the above assertions are satisfied by the model.

5. Conclusion

We introduced a novel verification method for Basic Lotos specifications based on Petri net theory. The method can handle non-finite state systems although the example presented in this paper has the finiteness property.

The verification method consists of building a coverability graph. We use a notation slightly different from the usual syntax for Karp and Miller graphs. Properties are stated by logical formulae and evaluated by visual inspection of the coverability graph associated with the Lotos specification. In contrast to other similar approaches, no intermediate Petri net coding is required. That means: (1) a verification procedure of lower complexity, and (2) reachability graphs that are easier to interpret since states contain Lotos expressions nearly in their original form.

Acknowledgements. This work was performed within a research project on object-oriented specifications funded by Bell Northern Research and the Computer Research Institute of Montréal. The authors thank the members of this project for many fruitful discussions.

References

[Barb 90a] 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.

[Barb 90b] M. Barbeau, G. v. Bochmann, *Verification of Lotos Specifications: A Petri Net Based Approach*, Proc. of Canadian Conference on Electrical and Computer Engineering, Ottawa, September 1990 (Full paper: *Deriving Analysable Petri Nets from Lotos Specifications*, Research Report No. 707, Dept. d'IRO, Université de Montréal, 1990).

[Boch 78] G. v. Bochmann, *Finite State Description of Communication Protocols*, Computer Networks, Vol. 2, October 1978, pp. 361-372.

[Ehri 85] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specifications 1*, Springer-Verlag, Berlin, 1985.

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

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

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

[Knut 84] D. Knuth, *Literate Programming*, Computer Journal, Vol. 27, No. 2, May 1984, pp. 97-111.

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

[Miln 83] R. Milner, *Calculi for Synchrony and Asynchrony*, TCS 25, 1983, pp. 267-310.

[Miln 89] R. Milner, *Calculus for Communication and Concurrency*, Prentice-Hall, 1989.

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

[Viss 87] C. A. Vissers, G. Scollo, M. van Sinderen, *Architecture and Specification Style in Formal Description of Distributed Systems*, Proc. of PSTV VIII, Atlantic City, 1987.