AN EXECUTION MODEL FOR LOTOS SPECIFICATIONS

Cheng WU and Gregor v. BOCHMANN

Dept. I.R.O., Université de Montréal, Québec, Canada

Abstract:

The paper describes a model for executing LOTOS specifications (temporal control part). The execution model is based on an *activity tree* with *attributes*. The activity tree reflects the dynamic relations between the process invocations and activations of behavior expressions in the specified system, while functions related to the attributes control the execution of interactions and the growing and updating of the tree. The problem of *infinite branching*, which is caused by non-well guarded specifications or specifications containing generalized choices, is discussed based on the strategies for growing the activity tree.

1. Introduction

LOTOS [LOTOS 87, Bolo 87] is an FDT which is standardized within ISO for formally specifying communication protocols and services of Open System Interworking (OSI). It is also applicable to distributed systems. LOTOS consists of two parts: Abstract Data Types and the Temporal Control part (related to CCS [Miln 80]). This paper concerns implementation issues of the latter part.

LOTOS is designed as an executable specification language. Execution of a specification plays an important role in development process of communication software [Turn 89]. Some interpreters (or simulators) have been or is being developed based on high level programming languages [Bria 86] [Logr 88] [Mana 89] [Gilb 89]. They allow users to simulate the execution of a LOTOS specification and check whether it behaves correctly. Some work also has been done on translating LOTOS to state machines and restricted form of LOTOS to efficiently execute LOTOS specifications [Karj 88] [Dubu 89] [Boga 89] [Quem 89]. But there are still several questions which need further attention, such as distributed implementation and infinite branching. Concerning these questions, there are certain difficulties for LOTOS specifications which contain 'non-well-guarded' recursion, generalized choices, or dynamic rendezvous matching. A specification containing 'non-well-guarded' recursion or generalized choices may cause an implementation into an infinite loop, and the dynamic rendezvous matching of LOTOS makes the distributed implementation a challenge.

In LOTOS, distributed systems are described in terms of processes. A system as a whole is a single process, in the following called 'system process', which may consist of several interacting sub-processes. These sub-processes may in turn be refined into sub-sub-processes etc., so that a specification of a system in LOTOS is essentially a hierarchy of process definition. In LOTOS, the realtion between two processes is defined by operators ';' '>>', '[]', '||', '|||', or '[>', which represent sequential, enabling, alternative choice, dependent parallelism, independent parallelism and disabling respectively[LOTOS 87].

In a LOTOS specification, a process is said to be active if it is the system process or it is called by its super-process which is active. For the specification P:=P1 * P2 (where * is one of operators [], ||, ||| and [>), for example, if P is active, the sub-processes P1 and P2 are also active. However, for P:= g;P1 or P:= P2>>P1, P1 is not active when P is active, since it has to wait for the interaction at gate 'g' or the successful termination of P2. According to the LOTOS semantics, all active processes must be considered at each given time for possible execution of interactions.

There are two cases in which a LOTOS specification will result in an infinite number of active processes. One is a specification which contains 'non-well-guarded' recursion, for example in the case of a specification of the form P[a](x:int):=a!x; stop * P[a](x+1) (where * is one of the operators [], ||, ||| or [> etc). Another case is a specification which contains a generalized choice, such as P:=choice x1:t1, ..., xn:tn [] B(x1, ..., xn) where one of the types t_i (i=1, ..., n) defines an infinite set of possible values. A specification with one of the two cases above defines a behavior allowing for an infinite number of active processes, and when executed, the implementation or interpreter may create infinitely many process instances, which practically means that the system loops or uses up all system resources.

LOTOS uses multi-way rendezvous for communication among processes, which is useful for system specification [Char 87]. However, multi-way rendezvous is more complex than the two-way rendezvous used in many other languages. In the case of LOTOS, it is difficult to tell in gerneral, by static analysis, how many processes, and which processes take part in a LOTOS rendezvous. As an example, we consider the following specification:

```
P:= hide a, b in
 P1[a,b] || P2[a,b]
 where
 P2[a,b]:= hide c in
  (a; stop) [] ((a; stop [] c; stop) || P4[a,b,c])
   ...
```

For an interaction at gate 'a', there are two choices within P2:
(1) P1 may rendezvous with a; stop.
(2) P1 may rendezvous with (a; stop [] c; stop) || P4.
In case (1), there are two processes involved in the rendezvous, while in case (2), there are at least three processes, or more, if P4 contains several subprocesses participating in the interaction.

In order to provide a framework for discussing problems of distributed implementation and infinite branching, an execution model is suggested in this paper . The execution model is

CH2827-4/90/0000-1890 $1.00 © 1990 IEEE

based on an activity tree with attributes. The activity tree reflects the dynamic relation between the process invocations and activations of behavior expressions in the specified system, while the functions related to the attributes control the execution of interactions and the growing and updating of the activity tree. In this paper, we only discuss the problem of infinite branching by presenting different growing strategies based on the execution model. The growing strategies have different properties concerning the handling of non-well-guarded specifications. In [Boch 89?] there is a detailed discussion of the problem of distributed implementation also using this execution model.

In Section 2, we will present the execution model. In Section 3, we will discuss the problem of infinite branching based on different growing strategies. Section 4, finally, contains the conclusions.

2. Execution Model

The execution model is based on an activity tree with attributes. In Section 2.1, we present the activity tree and its growing and updating rules. In Section 2.2, we define the evaluation rules for attributes. In Section 2.3, we present the three phases for the execution of interactions of the execution model. Finally, in Section 2.4, we discuss several aspects of the execution model which are particularly interesting.

Figure 1 shows a LOTOS specification of the 'Example' system, which will be used as an example in rest of this paper. After dropping money into a VM (vending machine), a boy may obtain a candy, if the latter is not be consumed by one of the two little devils that are included in the VM. In Figure 1, 'm' denotes 'money', 'c' denotes 'candy', and 'e' denotes 'eat_candy'.

```
Specification Example:=
        hide m, c in
        Boy[m, c] || VM[m, c]
        where
        process Boy[m, c]:noexit:=
         m; ( c; Boy[m, c]
              []
              Boy[m, c])
         endproc
        process VM[m, c]:noexit:=
         hide e in
         Machine[m, c, e]
         |[e]|
         (Devil[e] ||| Devil[e])
         where
        process Machine[m, c, e]:noexit:=
         m; ( c; Machine[m, c, e]
              []
              e; Machine[m, c, e] )
        endproc
        process Devil[e]:noexit:=
         e; Devil[e]
        endproc
        endproc
endspec
```

Figure 1: Example system - Boy and VM

2.1. Activity tree

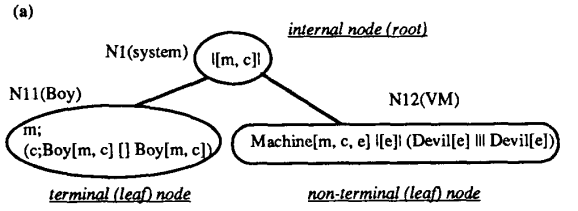
Trees have been used to present LOTOS specifications. For example, a so-called action tree in [LOTOS 87] (also called behavior tree in [Logr 88]) is used to show all possible action sequences defined by a LOTOS specification. A syntax tree can also be used to show the structure of a LOTOS specification and the relations among its actions. Below is the context-free grammar of a simplified syntax for the temporal control part of LOTOS, which is the basis on which we will define our activity tree. In the next section, we will define 'attributes' to deal with interaction offers involving parameters.

In the following, 'B' is a non terminal (and also the starting) symbol of the grammar, and {[], ||, [>, ;, >>, stop, exit, i, g1, ..., gn} is the set of terminal symbols. Each g∈ {i, g1, ..., gn} denotes a gate in the system, each r∈ {[], ||, [>, ;, >>} denotes a LOTOS operator, and 'stop' and 'exit' denote the STOP and EXIT processes respectively.

(1) B→ t for each t∈ { stop, exit}
(2) B → g;B for each gate g∈ {i, g1, g2, ..., gn}
(3) B → B >>B
(4) B → B [] B
(5) B → B || B
(6) B → B [> B

In contrast to the syntax tree of a LOTOS specifiction which represents the static structure of the text of the specification, the activity tree represents a dynamic changing system state during the execution of the specification. Nevertheless, it has certain similarities with the syntax tree in so far as the production rules of the activity tree correspond to the above syntax rules. The major difference is that the activity tree is normally not completely expanded. It is grown in a top-down fashion, as explained below, starting with the root node which represents the system specification.

The activity tree reflects the possible activities and the dynamic relationships between the active behavior expressions during the execution of the specified system. An activity tree consists of leaf nodes and internal nodes. An internal node represents the relation between its descendent nodes, i.e. one of the LOTOS operators [], ||, |||, and [> etc., or contains the description of the behavior to be activated after the successful termination of its descendants, i.e. >>B (where B is a behavior expression). There are two kinds of leaf nodes: terminal and non-terminal. A terminal node corresponds to a behavior expression 'g;B', where 'g' is called an active gate and 'B' is the behavior expression which will be activated after a rendezvous happens at 'g'. A non-terminal node cannot directly participate in an interaction, it must first be expanded. A non-terminal node corresponds to a behavior expression 'B1#B2', where # is one of the operators ||, |||, [], [> and >>, and 'B1' and 'B2' are behavior expressions. During the growing phase, a non-terminal node may be expanded and may thus lead to new terminal nodes that may participate in interactions. Figure 2(a) shows the activity tree of the Example system before any money is dropped in. Figure 2(b) shows the tree after the expansion of the node N12 representing the vending machine (VM). Note that in the node N121, the invocation of the 'Machine[m, c, e]' process is replaced by its definition, as given in Figure 1.



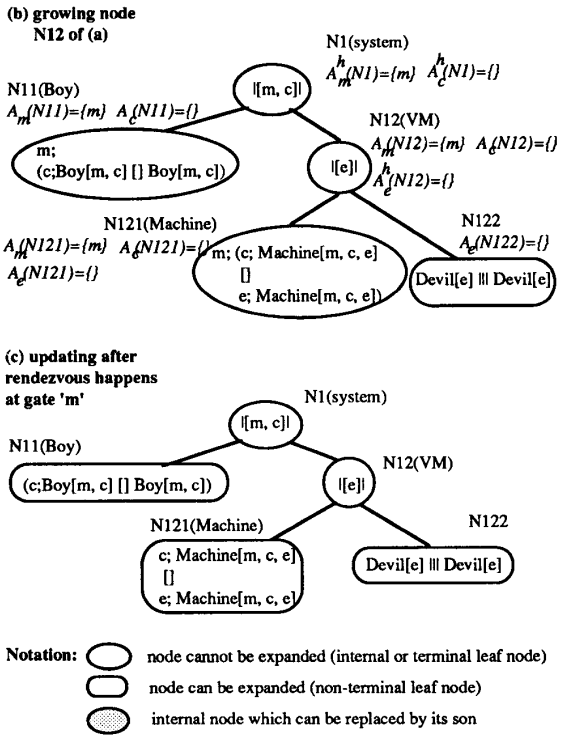


Figure 2: Different stages of the activity tree for the system of Figure 1.

When executing a LOTOS specification, the system behavior changes dynamically. So does the activity tree. The activity tree can be grown and updated. By growing, we mean that the system expands non-terminal nodes in order to find terminal nodes with possible interactions. By updating, we mean that, after a rendezvous, the system prunes those sub-trees of the activity tree which represent alternative behavior not possible any more and let some behaviors (next behaviors) be active. Figure 3 shows the rules for growing. A non-terminal (leaf) node 'B1*B2' (* is one of [], ||, |||, and [>) can be expanded to become an internal node '*' and with two son (terminal or non-terminal nodes) 'B1' and 'B2', as shown in Figure 3(a). Figure 3(b) shows a non-terminal node 'B1>>B2' can be expanded to become an internal node '<<B2' with a son (terminal or non-terminal) node 'B1'. We note that there is no growing rule corresponding to the syntax rule of process invocation. If the LOTOS behavior expression of a non-terminal node contains a process invocation, this invocation will be replaced by the behavior of the corresponding process definition with a substitution of its parameters, as defined by the LOTOS semantics. The so obtained behavior is then the basis for further expansion of the node.

Figure 4 shows two of the rules of updating. After participation a rendezvous at gate 'g', a terminal node 'g;B' become (terminal or non-terminal) node 'B', as shown in Figure 4(a). Figure 4(b) shows a tree with root node '[]' and two sub-trees 'B1' and 'B2'. When a rendezvous happens in 'B2', 'B1' is pruned and 'B2' is updated to "B2'". That is, the original tree become one with empty root node (which can be replaced by its son) and a sub-tree "B2'". The full updating rules are given in [Wu 89] by comparing them with LOTOS semantics as defined by the transition system given in [LOTOS 87]. The growing and updating of the activity tree will be discussed in more detail in Section 2.3.

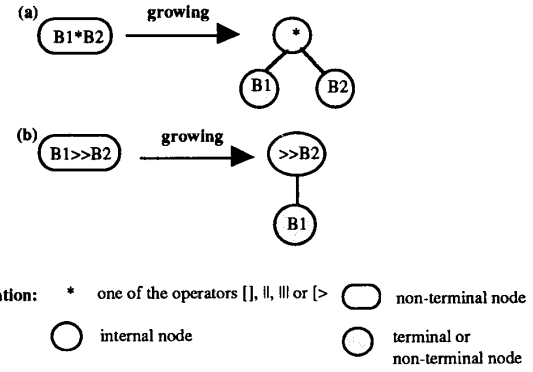


Figure 3: The growing rules

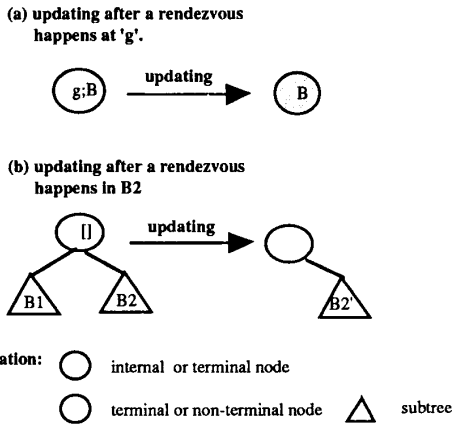


Figure 4: Two updating rules

2.2.Attributes

Attributes are defined in the activity tree. Their functions are to determine which nodes participate in a rendezvous on a given gate. Similar as in the case of attribute grammars [Boch 76c], the attributes are associated with the nodes of the tree. In contrast to attribute grammars, however, where the values of the attributes are evaluated once and for all for each given syntax tree, the values of the attributes associated with a node in the activity tree may change over time, as the structure of the activity tree changes.

Without restriction of generality, we may assume that each node 'B' of an activity tree corresponds to a specification with the general structure 'P[S1]:= hide S2 in < expression>, where S1 and S2 are the gate lists. Here all free gates of <expression> must either be in S1 or S2. In most cases S2 will be empty, for instance, a node representing the behavior 'g1;B1 [] g2;B2' will be written as 'P[g1, g2] := hide in g1;G1 [] g2;B2'. An attribute Ag is defined in node 'B' for each $g \in S1 \cup S2$. An attribute Ag is also called a 'hide attribute' (denoted as A^h_g) if $g \in S2$. The value of attribute Ag is a set of interaction offers concerning the gate 'g'.

The attributes of the activity tree are 'synthesized', that is, they are evaluated by applying the evaluation rules from the bottom of the tree towards the top. The precise definition of these evaluation rules is given in the following table. In the table, 'Ag(B)' denotes the attributes of gate 'g' in node 'B', 'o_g' denotes the interaction offer of gate 'g', 'S' denotes a gate list, and 'B→ B1#B2' denotes an internal node 'B' which has two son nodes, the left son 'B1' and the right son 'B2', where '#' is one of operators [], ||, [>, and >>. In the table, there are two functions **matched** and **derived**. Their formal definitions are given in [Wu 89]. **Matched**(o1,o2) is true if the two offers 'o1' and 'o2' are compatible for a rendezvous, and **derived** (o1,o2) is a single offer including the constraints imposed by 'o1' and 'o2'. For instance, **matched**('g?x:int!3?z:int', 'g?x:int?y:int!5') = **true** and **derived**('g?x:int!3?z:int', 'g?x:int?y:int!5') = 'g?x:int!3!5'.

Attribute evaluation rule	
For leaf nodes:	
$Ag(g;B) = \{o_g\}$	
$Ag(B) = \phi$	if B is non-terminal node
For internal nodes:	
Ag(B) = Ag(B1)	if B→ B1>>B2
$Ag(B) = Ag(B1) \cup Ag(B2)$	if B→ B1 [] B2
$Ag(B) = Ag(B1) \cup Ag(B2)$	if B→ B1 |S| B2 and g∉ S
$Ag(B) = \{o'_g \mid o'_g = derived(o1_g, o2_g),$ $o1_g \in Ag(B1), o2_g \in Ag(B2)$ and $matched(o1_g, o2_g) = true\}$	if B→ B1 |S| B2 and g∈ S
Ag(B) = Ag(B1)∪Ag(B2)	if B→ B1 [> B2
(Note: Ag(B) =ϕ if gate 'g' is not defined in node 'B')	

It is clear that a rendezvous is possible at gate 'g' if the attribute A^h_g at the node where 'g' is hidden contains an offer 'o_g'. All nodes that participate in the derivation of 'o_g' will involve in the rendezvous. For example in Figure 2(b), a rendezvous can only happen at gate 'm' because $A^h_m(N1)=\{m\}$, $A^h_c(N1)=\{\}$, and $A^h_e(N12)=\{\}$. Nodes N11 and node N121 will be involved in the rendezvous.

2.3 Three phases for the execution of interactions

The activity tree changes dynamically during the execution of LOTOS specifications through the repetition of the following three phases: growing, matching and updating. In the growing phase, the system expands non-terminal nodes until all or some terminal nodes with possible interactions are reached. After that, the system goes into the matching phase, by evaluating attributes, to find possible rendezvous usually involving several terminal nodes of the tree. If a possible rendezvous is found and executed, the matching phase is followed by the updating phase during which the system updates the tree, according to the rules discussed in Section 2.1 to reflect the state change implied by the rendezvous. If the matching phase does not lead to any rendezvous, the growing phase is resumed.

An example of growing is given by Figure 2(b) which shows the activity tree obtained by expanding the non-terminal node N12 of the Example tree of Figure 2(a). Figure 2(b) also shows the values of attributes in each node of the tree, which are obtained in the matching phase. An example of updating is given by Figure 2(c) which shows the Example tree of Figure 2(b) after a rendezvous at gate 'm' in which the terminal nodes N11 and N121 participated.

2.4. Interesting aspects of the model

The execution model describe above can support parallel processing and selective, possibly time-dependent, creation processes (activities). It provides also a framework for discussing the problems of distributed implementation and infinite branching.

The three phases of growing, matching and updating in different parts of the activity tree could be processed largely in parallel. For example, when a rendezvous happens in one sub-tree, the system may update the sub-tree while the other sub-trees may continue the 'growing ' or 'matching' activities. This feature allows us to discuss the problem of distributed implementation of LOTOS specifications based on the execution model. By distributed implementation, we mean an implementation of a system specified in LOTOS involving a given number of sites, communicating with one another by the exchange of messages through an underlying reliable communication medium. At present, some work has been done about the implementation of multi-way rendezvous in distributed environment [Gao 89] and distributed implementation of LOTOS [Boch 89?]. In [Boch 89?], the different sub-trees of the activity tree reside on different physical sites and do 'growing' and 'updating' independently, and the procedure of evaluation of attributes is replaced by a so-called virtual ring algorithm [Gao 89] which deals with the implementation of distributed rendezvous interactions.

As discussed in Section 1, all active processes are considered as candidates for participating in interactions at the same time (time independence) in LOTOS semantics. However, in the execution model, we could specialize the LOTOS semantics by considering only selective creation of active processes, and possibly time dependent activation by designing different growing strategies for the activity tree (see Section 3).

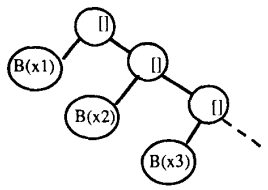


Figure 5: The activity tree of P:=choice x:t [] B(x)

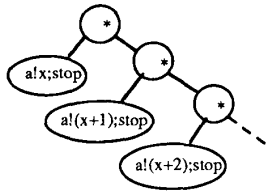


Figure 6:
The activity tree of
P[a](x:int):=(a!x;stop) * P[a](x+1),
where * is one of the opertors [], ||, ||| or [> etc.

The two cases of infinite branching mentioned in Section 1 can be modeled by the activity tree as follow: in the case of a specification of the form P:=choice x: t [] B(x) where 't' is an infinite (enumeratable) set, we consider an activity tree of the form shown in Figure 5, which results in a finite number of nodes for any finite depth of the activity tree. For the case of a specification of the form P[a](x:int):= a!x;stop * P[a](x+1) (where * is one of the operators [], || or [>), we obtain an activity tree of the form shown in Figure 6. In Section 3, we will discuss the infinite branching problem in the context of different growing strategies for the activity tree.

Time dependent process creation can be used as a basis for defining the semantics of performance parameters, such as proposed in [Boch 88b]. For example, the behavior expression 'a; suite_a[x,y] [] b; wait 50; suite_b[x,y]' defines a process which may participate in actions 'a' or 'b' (depending on its environment). A delay is introduced if action b is executed, such that suite_b can only start 50 time units later. This can be modelled by delaying the creation of the process representing suite_b by 50 units.

3. Growing Strategies

We discuss in this section several strategies for the growing phase of the LOTOS execution model. Some of these strategies are only suitable for the restricted class of well-guarded specifications. We are also interested to know whether the execution model interprets the LOTOS specifications correctly. For this purpose we give in the first subsection some definitions concerning desirable properties of LOTOS interpreters.

3.1. Desirable properties of interpreters

Let S be any LOTOS specification and M[S] be an execution model (interpretation) of S. Let σ denote a state of M[S], and σ_{int} denote the initial state. Let G be a set of observable actions, g1, g2,... ranging over G, **i** be an invisible action. Let δ range over $G \cup \{\mathbf{i}\}$, and α denote a string $\delta_1\delta_2...\delta_n$ of actions.

A transition relation - δ -> is defined as: σ - δ -> σ' iff after an action δ happened in σ, the system state changes to σ'. An extension transition relation = α => is defined as : $\sigma=\alpha=> \sigma'$ iff there exist σ_i, $0 \le i \le n$, such that $\sigma=\sigma_0\text{-}\delta_1\text{->}\sigma_1...\sigma_{n-1}\text{-}\delta_n\text{->}\sigma_n=\sigma'$.

Let σ - δ -> denote that there exists a σ' with σ - δ -> σ'; $\sigma=\alpha=>$ is defined analogously. Let σ - δ ->* and $\sigma=\alpha=>*$ be the negations of σ - δ -> and $\sigma=\alpha=>$ respectively.

Let $Tr(M[S]) = \{\alpha | \sigma_{int}=\alpha=> \}$ denote the set of possible traces of M[S].

Definition of soundness:
Let M_1 and M_2 be execution models. M_1 is said sound based on M_2 iff $\forall S\ (Tr(M_2[S]) \supseteq Tr(M_1[S]))$. M_1 is sound if M_1 is sound based on the ideal execution model as defined by the LOTOS semantics.

Definition of completeness:
Let M_1 and M_2 be execution models. M_1 is said complete based on M_2 iff $\forall S \forall \alpha \forall g(\exists \sigma 1'(\sigma 1_{int}=\alpha=>\sigma 1'\text{-}g\text{->}*) \rightarrow \exists \sigma 2'(\sigma 2_{int}=\alpha=>\sigma 2'\text{-}g\text{->}*))$. M_1 is complete if M_1 is complete based on the ideal execution model as defined by the LOTOS semantics.

It is noted that the soundness of the execution model described in this paper is based on the characteristics described in Section 2. In this respect, we note the similarity of the LOTOS semantics with the updating rules for the activity tree (see [Wu 89]) and the rules for rendezvous matching described in Section 2.2. This similarity suggest that our execution model is sound. In the next sections, we will only discuss the completeness of the execution model in the case of different growing strategies.

3.2. Interactive interpretation

A typical LOTOS Interpreter is described in [Bria 86, Logr 88]. When executing a LOTOS specification, the interpreter creates, during the growing phase, all possible nodes and, in the corresponding matching phase, makes a list of all possible rendezvous. An interactive user must select one of these interaction for execution. Then, the system prunes the nodes which are out of date and does the growing and matching again. In this case, the non-determinism of a LOTOS specification is implemented by the user choosing one of the possible interaction.

The LOTOS interpreters can execute all well-guarded LOTOS specification as long as enough memory space is available for all active nodes. However, the interpreter can not handle all non-well-guarded LOTOS specifications because the growing step may loop indefinitely. To deal with such cases, an interpretation parameter N may be introduced which limits the number of nodes in a system. However, this may lead to blocking in cases where a possible rendezvous could have been found if a larger value had been chosen for N; this means, the interpretation algorithm is not complete.

An interactive LOTOS interpreter can be transformed into an automatic interpreter by making an automatic random choice among the possible rendezvous at each step of the interpretation process(see for instance mechanism M_0 in [Hori 88]).

3.3. Breadth first and random growing

M1, another interpretation mechanism presented in [Hori 88], controls the growing by 'random choice'. It improves the interpreters described above by reducing the space, but it can not model a non-well-guarded specification such as P[a](x:int):= a!x ||| P[a](x+1), because the growing phase will loop. As a matter as fact, in [Hori 88], both M_0 and M_1 are presented for executing only well-guarded specifications.

We present here a simple random growth strategy which can handle infinite branching caused by non-well-guarded expressions or general CHOICE statements. The strategy is as follows: in each growing phase, a subset of all the non-terminal leaf nodes of the activity tree is selected randomly. All these selected nodes will be expanded. The non-terminal descendent nodes resulting in that expansion are not further expanded.

Together with this growing strategy, the LOTOS execution model described here is complete, that is, it will find a possible rendezvous if there is one according to the LOTOS semantics.

If there is no possible rendezvous in the interpreted specification, there is either a detected deadlock, that is, all leaf nodes of the activity tree are terminal (non expandable) and no rendezvous is possible, or the activity tree always allows for further expansion, which implies that the specification contains some non-well-guarded recursions. In the latter case, the interpretation process loops.

Because of the random selection process, the completeness property mentioned above only holds statistically, however, with probability one, which means it is satisfied for all practical purposes. In the case that all non-terminal nodes are expanded in one growing phase, the growing strategy becomes breadth-first and the interpretation is deterministically complete. A more detailed discussion of these issues is given in [Wu 90], where additional growing strategies are presented with considerations of both completeness and fairness.

4. Conclusions

We have described an execution model for simulated execution of LOTOS specifications which supports the selective creation (possibly time dependent) of LOTOS processes, and parallel processing. It also provides a framework for discussion various execution strategies. In this context, strategies which can handle non-well-guarded specifications are discussed with considerations of completeness.

The general execution model described here can also be used as a basis for designing LOTOS implementation strategies for distributed environments or for systems with parallel processors. In this context, it is important to limit the growing of the activity tree in order to reduce the number of messages to be transmitted between different sites. This is still an area for further study.

Acknowledgements:

The authors would like to thank Qiang Gao for helpful discussions. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, and the Ministry of Education of Quebec.

References

[Boch 76c] Gregor v. Bochmann, "Semantic evaluation from left to right", Comm, ACM19, pp. 55-62, 1976.

[Boch 88b] Gregor v. Bochmann and Jean Vaucher, "Adding performance aspects to specification languages", IFIP Symposium on Protocol Specification, Testing and Verification, Atlantic City, June 1988.

[Boch 89?] Gregor v. Bochmann, Qiang Gao, and Cheng Wu, "On the Distributed Implementation of LOTOS", submitted to FORTE89, 1989.

[Boga 89] Kees Bogarads, " LOTOS supported system development ", in K.J. Turner (editor) "Formal Description Techniques", Elsevier Science Publishers B. V. (North-Holland), 1989.

[Bolo 87] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Network and ISDN Systems, vol. 14, no. 1, pp. 3- , 1987.

[Bria 86] J.P. Briand, M.C. Fehri, L. Logrippo, A. Obaid, "Executing LOTOS Specifications", in Protocol Specification, testing and verification, B. Sarikaya and G. v. Bochmann (eds), North Holland, 1986.

[Char 87] A. Charlesworth, "The Multi-way Rendezvous", ACM Tran. on Programming Languages and Systems, Vol. 9, No. 2, July 1987, pp. 350-366

[Dubu 89] Eric Dubuis, " An algorithm for translating LOTOS behavior expressions into automata and ports ", FORTE'89, Vancouver, Dec 5-8 1989.

[Eijk 89] P.H.J. van Eijk, et al., "The formal description technique LOTOS", North Horlland Publ., 1989.

[Gao 89] Qiang Gao, G.v. Bochmann, " A Virtual Ring Algorithm for the Distributed Implementation of Multi-Rendezvous," Technical Report #675, University of Montreal, Dept. I.R.O.

[Gilb 89] D. R. Gilbert, " A LOTOS to PARLOG translator ", in K.J. Turner (editor) "Formal Description Techniques", Elsevier Science Publishers B. V. (North-Holland), 1989.

[Hoar 78] C. A. R. Hoare, "Communication Sequential Processes", 1978.

[Hori 88] Eiichi Horita, "Formulation of CCS with Typed Lambda Calculus and Its Efficient Interpretation Mechanism", 1988.

[Karj 88] G. Karjoth, " Implementation process algebra specifications by state machines", in " Protocol specification, testing , and verification", VIII, North-Holland, 1988.

[Logr 88] L. Logrippo, et al., "An interpreter for LOTOS: A specification language for distributed systems", Software Practice and Experience, to appear.

[LOTOS 87] " LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior," ISO, DIS 8807, 1987

[Mana 87] Mànas, "LIW An Implementation Wordbench for the Specification Language LOTOS", SEDOS/CS/WP/13/M, E.T.S.I., Telecomunication, Madrid, 1987.

[Mana 89] J. A. Manas, T. de Miguel-More, " From LOTOS to C", in K.J. Turner (editor) "Formal Description Techniques", Elsevier Science Publishers B. V. (North-Holland), 1989.

[Miln 80] R. Milner, "A Calculus of Communicating Systems", Secture Notes in CS, No. 92, Springer Verlag, 1980.

[Obai 88] Abdellatif Obaid, "Specification of Implementation Choices for LOTOS Descriptions", April, 1988.

[Quem 89] Juan Quemada, Santiago Pavon, Angel Fernandez, " Transforming LOTOS specifications with LOAL: The Parameterized Expansion ", in K.J. Turner (editor) "Formal Description Techniques", Elsevier Science Publishers B. V. (North-Holland), 1989.

[Turn 89] K. Turner, " A LOTOS - based development strategy ", FORTE'89, Vancouver, Dec 5-8 1989.

[Wu 89] Cheng Wu, G.v. Bochmann, "An execution model for LOTOS specifications", PUBLICATION #701, Dept. I.R.O., Universite de Montreal, Oct. 1989.

[Wu 90] Cheng Wu, G.v. Bochmann, " Fairness in LOTOS ", in preparation, 1990.