ON THE DISTRIBUTED IMPLEMENTATION OF LOTOS

Gregor v. BOCHMANN, Qiang GAO, Cheng WU

Université de Montréal
Dept. I. R. O.
C.P. 6128, succ "A"
Montreal, Quebec
Canada

This paper presents an approach to the distributed implementation of multiple rendezvous, including dynamic process creation, as defined by the specification language LOTOS. The approach is based on a so-called activity tree which reflects the dynamic relationships between the active processes within the system, and a virtual ring algorithm for the distributed implementation of a set of rendezvous, which was originally developed for a static set of processes. A new dynamic ring establishment algorithm is presented which serves as a bridge between the activity tree and the virtual ring algorithm. When growing the activity tree, the algorithm establishes for each LOTOS gate a virtual ring over which the virtual ring algorithm is applied.

1.Introduction

LOTOS is a high level specification language for OSI communication protocols and services and other distributed systems [LOTOS 87]. At present, several LOTOS interpreters have been built [Logr 88, Eijk 89]. It would be desirable to have a methodology and possible tools for the development of efficient implementations from LOTOS specifications especially in a distributed environment. However, in general, this is a difficult topic.

In this paper we consider the distributed implementation of LOTOS specifications. We mean by this an implementation of a system specified in LOTOS involving a fixed number of sites each implementing part of the specified system. We assume that the different sites communicate by the exchange of the messages through an underlying reliable communication medium. A possible implementation would be a configuration of several computers interconnected through a local area network, each running a LOTOS interpreter.

One of the issues involved with such a distributed implementation is the allocation of the different parts of the specified system onto the different sites. The partition of the system may have a strong impact on the overall efficiency of the system implementation, due to the relatively low speed of inter-node communication, compared with the communication of the different parts allocated within a single site. In this paper we do not deal with this allocation problem. We assume that an allocation algorithm is given which determines for

each behavior expression the site on which it will be executed. However, some basic principles for this distribution are discussed in Section 3.3.

We address in this paper another issue which is fundamental to distributed computing, i.e. distributed implementation of multiple rendezvous. In LOTOS, a distributed system is described in terms of behavior expressions and processes. They can be imagined as a black box with gates which are capable of communicating with the environment by means of interactions. An interaction is an atomic event by which two or more processes will synchronize at a common gate.
There are two aspects which distinguish the rendezvous in LOTOS from other rendezvous mechanisms used in languages such as CSP [Hoare 78], Occam [Occam]. They are the following:

(a) In LOTOS, a gate is shared by two or more processes. There is no owner of the gate. In a distributed system a process allocated to one site may not know the other processes on the other sites which synchronize on the same gate. Therefore the process on one site does not know with whom on the other sites it can synchronize on a given gate.

(b) Executing a LOTOS specification can be seen as a sequence of sets of possible rendezvous dynamically changing after each execution of a rendezvous. There is also the dynamic creation of new processes and gates. The processes involved in the next set of rendezvous may not be known until the previous rendezvous is executed. In certain cases of non-well-guarded specifications, the number of participating processes is not even bounded.

The distributed implementation of rendezvous interactions is not trivial, and has been under study for a long time. A survey of this topic can be found in [Levy 88], and another algorithm can be found in [Ram 87]. In [Gao 89] an new algorithm for distributed implementation of multiple rendezvous is presented based on a multiple virtual ring structure, which is more efficient and has better fairness properties than other known algorithms.

However the above algorithms assume that for each rendezvous which may occur, there is a fixed number of associated processes (or behavior expressions) which participate in the rendezvous. It is assumed that the processes are statically allocated to physical nodes. As we will show in this paper, these assumptions can be removed by making the virtual ring algorithm applicable to the distributed implementation of LOTOS rendezvous.

The distributed implementation of LOTOS discussed in this paper is based on an execution model and an activity tree which represents the state of execution of the specified system. This tree reflects the dynamic relationships between the active behavior expressions or processes within the system. Each terminal node of the activity tree contains an active gate 'g' and the description of the behavior to be activated after a rendezvous happens at 'g'. An internal node of the activity tree represents the relation between its dependent nodes, i.e. one of the LOTOS operators [], ||, |||, and [> etc., and also contains the description of the behavior to be activated after the successful termination of its dependents as shown in Fig. 2.

The execution model has three phases: (1) growing of the activity tree, (2) rendezvous matching, (3) updating after rendezvous execution. The activity tree changes dynamically during the repetition of the three phases of the execution model. In the growing phase, the

system creates new dependent nodes in order to find terminal nodes with possible interactions. In the matching phase, the system tries to find possible rendezvous usually involving several terminal nodes of the tree. In the updating phase, after a rendezvous, the system prunes those subtrees of the activity tree which represent alternative behavior not possible any more.

The activity tree can be exploited to find out information about processes involved in the possible rendezvous and the relationships between the present set of possible rendezvous and the next set which is valid after the execution of a rendezvous. This execution model provides a general framework for the distributed implementation of LOTOS, and for the handling of non-well-guarded specifications [Wu 89].

Based on this execution model, we present in this paper an approach for the distributed implementation of LOTOS rendezvous, including the dynamic creating of new processes and non-well guarded expressions. It can be summarized as follows. When growing the tree, a ring establishment algorithm, described below, establishes the necessary virtual rings corresponding to the possible rendezvous that may occur according to the semantics of the LOTOS specification. The rendezvous matching phase is realized by the virtual ring algorithm described in [Gao 89].

In Section 2, the activity tree is presented. In Section 3, the distributed implementation of LOTOS rendezvous is fully discussed. It is shown how the virtual ring algorithm and the ring establishment algorithm can be embedded into the execution model.

2. An Execution Model for LOTOS Specifications

2.1. An Example

Figure 1. is a LOTOS specification of a Boy and VM 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'.

2.2. Activity Tree

An execution model is suggested in [Wu 89] for the interpretation of LOTOS specifications, which is based on a so-called activity tree. Several questions, namely completeness, fairness and distributed implementation, can be discussed based on this execution model. Here, we are only interested in the question of distributed implementation of LOTOS specifications, that is, the activity tree is divided into several subtrees which are executed in a distributed environment.

The activity tree is defined based on a subset of the syntax of LOTOS. Below is the context-free grammar of a simplified syntax for the temporal control part of LOTOS, where '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 $g \in \{i, g1, ..., gn\}$ denotes a gate in the system, each $r \in \{[], ||, [>, ;, >>\}$ 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 specification 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. A 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 participate in an interaction. It corresponds to a behavior expression 'B1#B2', where # is one of the operators ||, |||, [], [> and >>, and 'B1' and 'B2' are behavior expressions. A non-terminal node may be expanded during the growing phase (see below) 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.

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 in order to find terminal nodes with possible interactions. In the matching phase, the system tries 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 to reflect the state change implied by the rendezvous. For instance, sub-trees representing alternatives not taken are pruned, and the behavior expressions associated with the terminal nodes that participated in the rendezvous are changed by deleting the initial rendezvous interaction. An example 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. An example of pruning is shown in Figure 2(e) which shows a later stage of the activity tree. Here the node N1211 shown in Figure 2(d) has been pruned after a rendezvous on gate 'e' in which the N1212 and N1221 are involved (one of the devils eats the candy). If the matching phase does not lead to any rendezvous the growing phase is resumed.

2.3. Interaction Offers Involving Parameters

The discussion above of the matching phase is simplified in two respects. First, nothing was said about how to determine which nodes participate in a rendezvous on a given gate. In the general context of this execution model, this issue is discussed in more detail in [Wu 89]. For the context of distributed implementation, Section 3 below presents a distributed algorithm for establishing a virtual ring for each gate which includes all those nodes which have to participate in a rendezvous on that gate.

The second aspect not addressed above is the question of interaction parameters. In LOTOS, two or more processes may be involved in a rendezvous, where each of these processes will provide certain restrictions on the possible values of the interaction parameters. In the rendezvous execution model described below (as well as in the ring algorithm described in Section 3), these restrictions are imposed in a consecutive manner, considering the restrictions of each process in term. For example, in the case of three processes P1, P2 and P3, the rendezvous procedure begins with 'matching' the offers of P1 and P2. If it succeeds, it continues by 'matching' a hypothetical offer 'derived' from the offers of P1 and P2 with the offer of P3. In the following, we will define the concepts of 'match' and 'derive' in more details, based on the semantics of LOTOS.

In general, a LOTOS process offering a rendezvous on a gate 'g' offers it with a certain number of parameters. For each parameter i (i=1,...,n), its mode m_i indicates whether a fixed value e_i is expected (mi=!) or whether different values of a certain type t_i are acceptable (mi=?). In the latter case, a variable V_i represents the accepted parameter value, and a predicate condition c_i (depending on Vi) may further restrict the acceptable values. In addition, an overall guard G, possibly depending on all the variable parameters of the interaction, may further restrict the acceptable parameter values.
In general, we represent an offer o_g of a LOTOS process for an interaction on gate 'g' in the form o_g= <{<m_i, v_i, t_i, c_i>| i=1, 2, ..., n}, G> where in addition to the explanation above, v_i=V_i if m_i=?, and v_i=e_i, if m_i=!; t_i is the type of v_i, and c_i=True if m_i=!. It is important to note that the offer of a process depends on its 'state', that is, the value expressions e_i and the conditions c_i and G may depend on process parameters and other local variables within the process. However, all these process parameters and variables have fixed values for each node in the activity tree which may provide an interaction offer. Therefore the only free variables of the conditions in an interaction offer are the interaction parameters V_i. It is therefore possible, as discussed below and in Section 3, to send an interaction offer in a message to another site and have it evaluated in the remote environment independently from the originator.

We now assume that two nodes (processes) P1 and P2 are ready to participate in a rendezvous on gate 'g' and provide the offers $o1_g$=<{<$m1_i$, $v1_i$, $t1_i$, $c1_i$>| i=1, 2, ..., n}, G1> and $o2_g$=<{<$m2_i$, $v2_i$, $t2_i$, $c2_i$>| i=1, 2, ..., m}, G2> respectively. We say that the two offers match, written 'matched(o1g, o2g)=true', if a rendezvous satisfying both offers is possible. The necessary and sufficient conditions are :

(a) n=m

(b) $t1_i$=$t2_i$ for i=1, ..., n

(c) for each i=1,..., n one of the following conditions is satisfied:

(i) $m1_i$=$m2_i$=! and $e1_i$=$e2_i$

(ii) $m1_i$=! and $m2_i$=? and $c2_i$($e1_i$)

($V2_i$ is assigned the value $e1_i$)

(iii) $m2_i$=! and $m1_i$=? and $c1_i(e2_i)$

($V1_i$ is assigned the value $e2_i$)

(iv) $m1_i=m2_i$=?

(d) there exist values e_j assignable to the free variables $V1_j$ and $V2_j$ (for those j for which condition (iv) is satisfied) and the other variables V1i and V2i take the values defined by conditions (ii) and (iii), respectively, such that G1 and G2 and all $c1_i$ and all $c2_i$ become true.

If two interaction offers $o1_g$ and $o2_g$ match, their combined constraints can be represented by a single (hypothetical) interaction offer o'_g which is derived from the offers $o1_g$ and $o2_g$, written "o'_g=derive($o1_g$, $o2_g$)". The derived offer o'_g of the form o'_g=<{<m'_i, v'_i, $t1_i$, c'_i>| i=1, ..., n}, G'> satisfies the following conditions:

(a) for each i=1, ..., n, depending on which of the conditions above is satisfied:

case (i): m'_i=!, v'_i=$e1_i$, c'_i=true

case (ii): m'_i=!, v'_i=$e1_i$, c'_i=true

case (iii): m'_i=!, v'_i=e2i, c'_i=true

case (iv): m'_i=?, V'_i=$V1_i$, c'_i= $c1_i$ AND $c2_i$*

(b) G'=G1 AND G2*

where G2* (c2*) is the predicate obtained from G2(c2i) by replacing the free variables $V2_i$ in G2(c2i) by the corresponding variables $V1_i$.

It is clear that three offers $o1_g$, $o2_g$ and $o3_g$ allow for a joint rendezvous iff matched($o1_g$, $o2_g$) and matched($o3_g$, derived($o1_g$, $o2_g$)) are true.

2.4. Growing Strategies

As mentioned in Section 2.2, the LOTOS execution model foresees an alternation of the growing and matching phases, interrupted by an updating phase for each rendezvous executed during the matching phase. Which part, and how much, of the activity tree is grown in each growing phase is not specified. In fact, various growing strategies can be designed [Wu 89b] which provide different advantages, such as ease of implementation, completeness and/or fairness of the interpretation process, and efficiency for a subset of the LOTOS language. For instance, an existing LOTOS interpreter [Logr 88] uses depth-first growing, which is relatively efficient but leads to infinite loops without interactions for certain non-well-guarded recursive expressions.

We present here a simple random growth strategy which can be used in the distributed environment discussed in Section 3, and also presents the advantage that is can handle non-well-guarded expressions. 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 recursion. In the latter case, the interpretation process loops. It is interesting to note that infinite choices due to a CHOICE statement can also be handled by an appropriately adapted growing strategy [Wu 89].

Because of the random selection process, the completeness property mentioned above only holds statistically with probability one, that is, if a rendezvous exists according to the LOTOS semantics then the probability that it is not found by the algorithm is zero. In the special case of the algorithm where all non-terminal nodes are expanded in one growing phase, the growing strategy becomes breadth-first and the interpretation is deterministically complete.

3. Distributed Implementation

In this section we present the details of our approach to the distributed implementation of LOTOS rendezvous. First we introduce the virtual ring rendezvous matching algorithm, and then we define an algorithm for the establishment of these rings. We end the section with a discussion of distributing the activity tree.

3.1. Overview of the Virtual Ring Algorithm

The virtual ring algorithm for rendezvous execution described in [Gao 89] applies to a static configuration of processes and gates. Such configurations can be specified in LOTOS as following.

```
Process P:  =
 hide g1, g2, g3 in
 P3[g3]  |[g3]|   P1[g1, g2, g3]   |[g1,g2]|   P2[g1, g2]
 where
    Process P1 [g1, g2, g3]: noexit=
                          g1 ! 3 ?x : integer ? z: integer; stop
                        [] g2 ! 5; stop
                        [] g3 ! 8; stop
                        endproc
   Process P2 [g1, g2]: noexit =
                          g1 ? y : integer ! 5 ? z : integer; stop
                        [] g2 ? x : integer; stop
                        endproc
   Process P3 [g3]: noexit= g3 ? t : integer; stop endproc
  endproc
```

There are several possible multiple rendezvous and a nondeterminism choice between gates g_1, or g_2, or g_3. However, the number of processes involved in rendezvous and the number of gates are static. The virtual ring algorithm presented below can be used to implement such a set of rendezvous in a distributed environment.

A virtual ring is a software structure accessed by a set of processes distributed over several sites in a system, which communicate through a network. Each gate in a specification is implemented as a ring, which is named after the gate. The processes sharing a gate are

connected to the corresponding ring. The order of processes on a ring is not important. It is assumed that the ring has been established initially. In Section 3.2 a ring establishment algorithm is presented, based on the activity tree, to handle the dynamic creation of new processes and gates. The virtual ring algorithm, on each given ring, proceeds in three phases: the rendezvous detection phase, the negotiation phase, and the rendezvous phase.

In the first phase, a so-called rendezvous Detection message (D) goes around the ring to determine whether all connected processes are ready to participate in a rendezvous. Each process (Pi) on the ring can initiate a D<Pi, gatej, inf, Attr> message. Each D<Pi, gatej, inf, Attr> message has a certain priority which only depends on the initiator's index (i), and is circulated over the ring "gatej". Only the process which has initiated the highest priority D message on the ring will receive its D message after travelling completely around the ring. This process becomes the leader for the rendezvous on that ring.

If another process (Pk) receives the D message, and is already captured by a third process Pl for other port, Pk will add the tuple <Pk, Pl> to the information field (inf) of circulating D message.

The attribute field (Attr) of the D message contains an effective interaction offer. This field is initialized by the originator with its locally determined offer (see Section 2.3) and is updated by each node on the ring by the offer derived from the locally determined offer and the offer of the received D message. If there is no match, the offer is replaced by NIL, the D message is canceled, and a so-called no rendezvous message (NR) is circulated

When the leader receives the returning D message, it checks the two information fields mentioned above to determine whether all processes on the ring are ready for a rendezvous. If so, the leader will enter the rendezvous phase and start the execution of the rendezvous by sending a so-called Rendezvous message (R), which establishes the rendezvous.

If not all of the processes are ready, depending on the priorities associated with the D messages, the leader on a ring may enter a phase of negotiation in which it sends so-called ASK messages to other leaders of other rings indicated in the field "inf". The result of the negotiation phase is either a rendezvous on the given ring or an abandon of the attempted rendezvous, indicated by the circulation of a No Rendezvous (NR) message.

A more detailed explanation of the algorithm can be found in [Gao 89]. Its complexity, in terms of number of transmitted messages, is $O(n^2)$ for each successful rendezvous, where n is the maximum number of processes on a ring. The simplified version of the algorithm has linear message complexity [Gao 89].

3.2. The Ring Establishment Algorithm

In the following, we show how a virtual ring can be established for each gate during the growing of the activity tree. We assume that a ring is formed by a pointer structure where each node on the ring has a pointer to the next node of the ring. Thus each node of the activity tree has one pointer for each gate to which it is connected. In a distributed environment, these pointers are system-wide pointers including, in addition to the node identifier, also the network address of the site containing that node. Each node contains a table in which there is a list of entries. Each entry contains the name of a gate, a pointer to

the next node on the ring, and the local interaction offer for that gate, if any. A given site may contain several nodes.

The rings in the activity tree are established during the growing of the tree. Figure 3 shows the changes in the pointer structures which occur during the different growing steps. A new ring is created when a HIDE clause is processed, as show in Figure 3(a). Figure 3(b) shows the inclusion of two newly grown leaf nodes when a non-terminal node, including dependent parallelism for the gate in question, is expanded. The parent node remains in the ring, but does not actively participate in the virtual ring rendezvous algorithm; it forwards all messages received without change.

Figure 3(c) shows the expansion of a non-terminal node with independent parallelism, alternatives or disruption, where the gate in question is only involved in one of the subprocesses. The other subprocess will not be involved in the ring.

If both dependent subprocesses are involved in the gate in question the ring is partially duplicated, as shown in Figure 3(f). In this case, the parent node will duplicate all received messages and send them to both descendents, unless the message was originated by a direct or indirect descendent of the parent in which case it is only sent towards the originator. In order to distinguish the different branches of a duplicated ring, the gate field of the messages include a list of node names which identify the branches through which the message has travelled (or, in the case of Rendezvous message, should travel). In the case of a D message, if necessary, each node (e.g. N1211 if Figure 2(f)) will append its name to this list before forwarding the message to its successor in the ring. When one of the duplicated messages returns to the leader and a rendezvous is possible, the leader will direct the following Rendezvous message through those branches through which the received D message had travelled.

Figure 2 shows various stages of the activity tree of the Example system of Section 2.1, and the associated virtual rings. As shown in Figure 2(a), N1(system) node grows two dependents, N11(Boy) and N12(VM), it establishes two rings for the gates 'm' and 'c'. In Figure 2(b), the node N12(VM) has grown two dependents, N121(Machine) and N122. It has inserted the dependent N121 into the rings for the gates 'm' and 'c' by providing N121 with a pointer to its successor. It has also established a new ring for the gate 'e' between its two dependents, N121(Machine) and N122.

3.3. Distributing the Activity Tree

In the discussions above, nothing was assumed about the distribution of the nodes of the activity tree over the different sites of the distributed system in which the LOTOS specification is to be executed. As mentioned in the introduction, we do not address in this paper questions relating to performance and distribution. In this subsection, we describe some basic aspects of the growing of a distributed activity tree.

In the simplest case, the dependent nodes generated during the expansion of a non-terminal node will reside on the same site as their parent. In order to obtain distribution, one of the following distribution primitives can be used:

(a)Growing a remote descendant node. When a non-terminal node is expanded, it may decide that one of the descendants and its future subtree should evolve on a different site.

(b) Moving part of the existing activity tree to another site. In this case each moving node Nx should send a MOVE<Nx, new address> message around every ring to which it is connected in order to allow the other nodes to update their pointers.

While the growing and matching phases of the LOTOS execution model can be distributed, as discussed above, the updating phase also requires coordination between the different subtrees of parent nodes containing the [] or [> operators. Instead of developing a distributed algorithm for this purpose, we assume in this paper that all descendents of such nodes remain on the same site until one of the subtrees is pruned. This restriction also allows the manager of each site to enforce mutual exclusion between alternative subtrees, that are related by the [] or [> operator.

For example, if the initial expansion shown in Figure 2(a) places the node N12 in a remote site, the subtrees of the nodes N11 and N12 may remain at two different sites, respectively. When node N1221 and N1222 (two devils) are grown by node N122 (Figure 2(d)), they may be placed in a third and forth site. However, the nodes N111 and N112 must remain in the same site of node N11 because they related alternatives (Figure 2(d)). As soon as one alternative is pruned, the remaining one may be **moved** to a different site.

4. Conclusions

We have shown in this paper how a distributed implementation of multiple rendezvous, as required by general LOTOS specifications, can be obtained by combining a virtual ring algorithm designed for rendezvous with static processes and gate structures, with a general LOTOS execution model which allows the dynamic creation of the virtual rings which are the basis for the former algorithm.

The ring establishment algorithm does not add any further message complexity to the virtual ring algorithm described in Section 3.1. The messages passed in the system for a successful rendezvous matching on a ring are the same as in the original virtual ring algorithm.

By using a random or breadth-first growing strategy for the activity tree our approach can handle non-well-guarded specifications.

Further research is required to optimize the local organization at a given site for a coordinated and efficient processing for the interactions of the different nodes residing in that site. A prototype implementation of the here described approach is planned.

Acknowledgements: Special thank goes to Prof. Pierre Mondain-Monval for helpful discussions and comments.This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, and the Ministry of Education of Quebec.

References:

[Buck 83] G. N. Buckley, A. Silberschatz, "An Effective Implementation for the Generalized Input-Output Construct of CSP," ACM Transactions on Programming Languages and Systems, vol. 5, No. 2, April 1983. pp. 223-235

[Charl 87] Arthur Charlesworth, "The Multiway Rendezvous," ACM Transactions on Programming Languages and Systems, Vol. 9, No. 2, July 1987, pp. 350-366.

[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.

[Hoare 78] C. A. R. Hoare, "Communication Sequential Processes,"
Communications of the ACM, August 1978, Vol. 21, No. 8.

[Levy 88] E. Levy, "A Survey of Distributed Coordination Algorithms", MCC Technical Report Number STP-271-88, 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.

[Occam] " Occam Programing Manual," INMOS Limited, 1984.

[Ram 87] S. Ramesh, "A New and Efficient Implementation of Multiprocesses Synchronization," PARLE conf. Eindhoven, June 1987.

[Wu 89] Cheng Wu, G.v. Bochmann, "An Execution Model for LOTOS Specifications," submitted for publication.

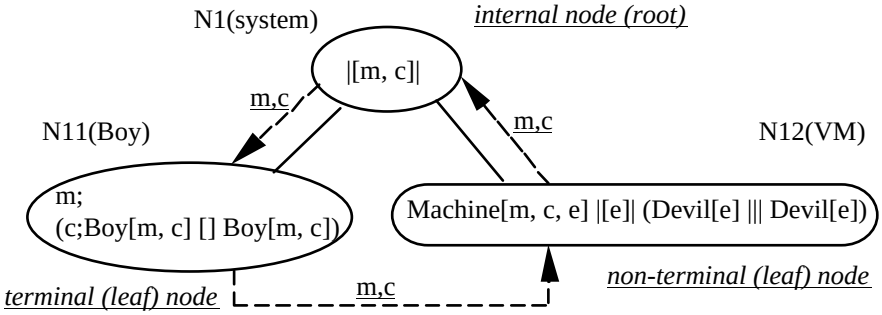
[Wu 89b] Cheng Wu, G.v. Bochmann, "Comparison of different interpretation algorithms for LOTOS", in preparation, 1989.

Figures

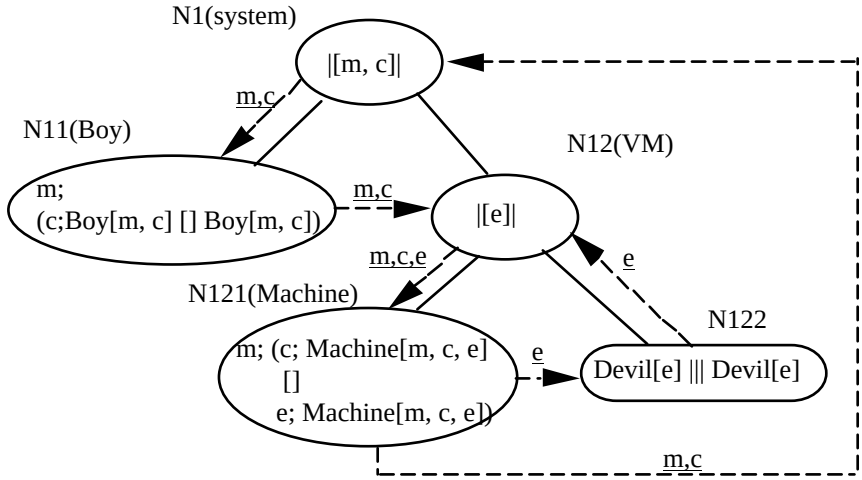
```
Specification Example:=
    hide m, c in
    Boy[m, c] || VM[m, c]
    where
    process Boy[m, c]:noexit:=
      m; ( c; Boy[m, c]
           []
           Boy[m, c])
     endproc
    process VM[m, c]:noexit:=
```

```
        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
```

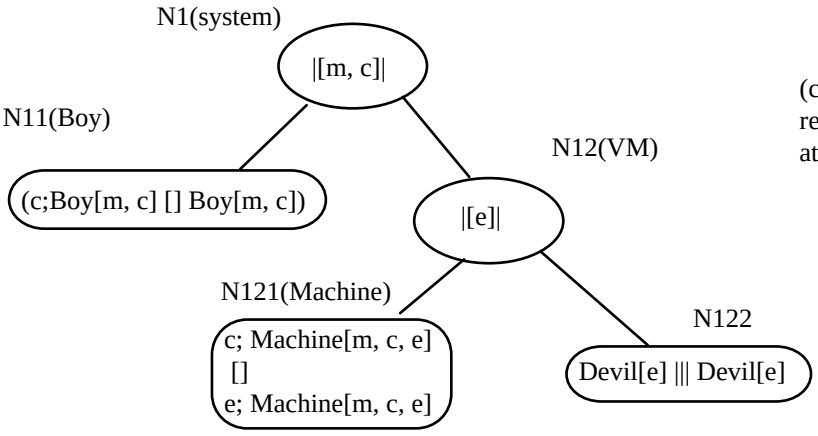
Figure 1: Boy and VM system



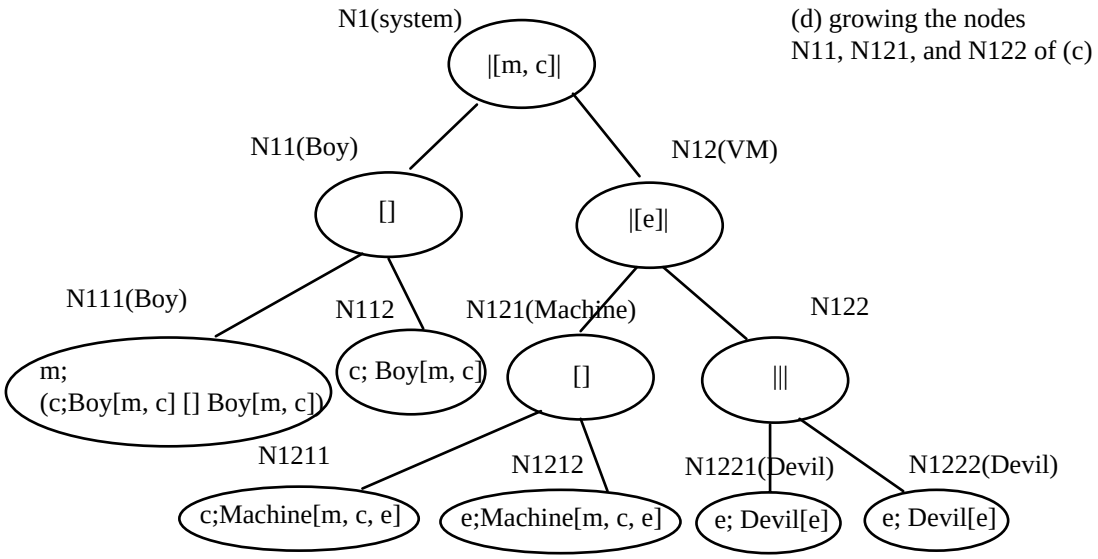
(a)

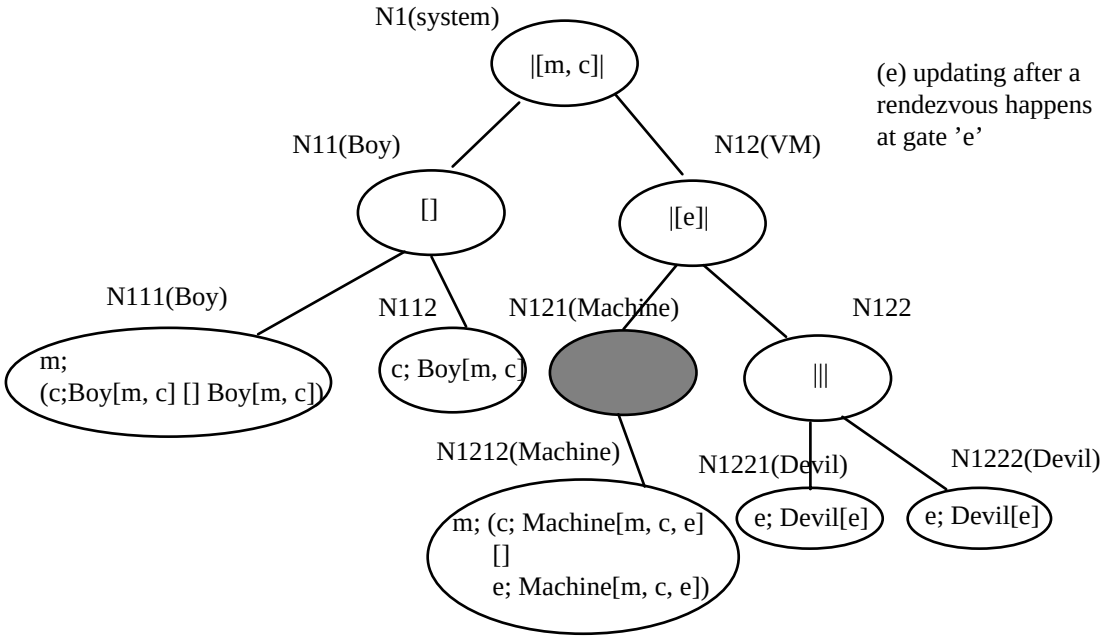


(b) growing node N12 of (a)

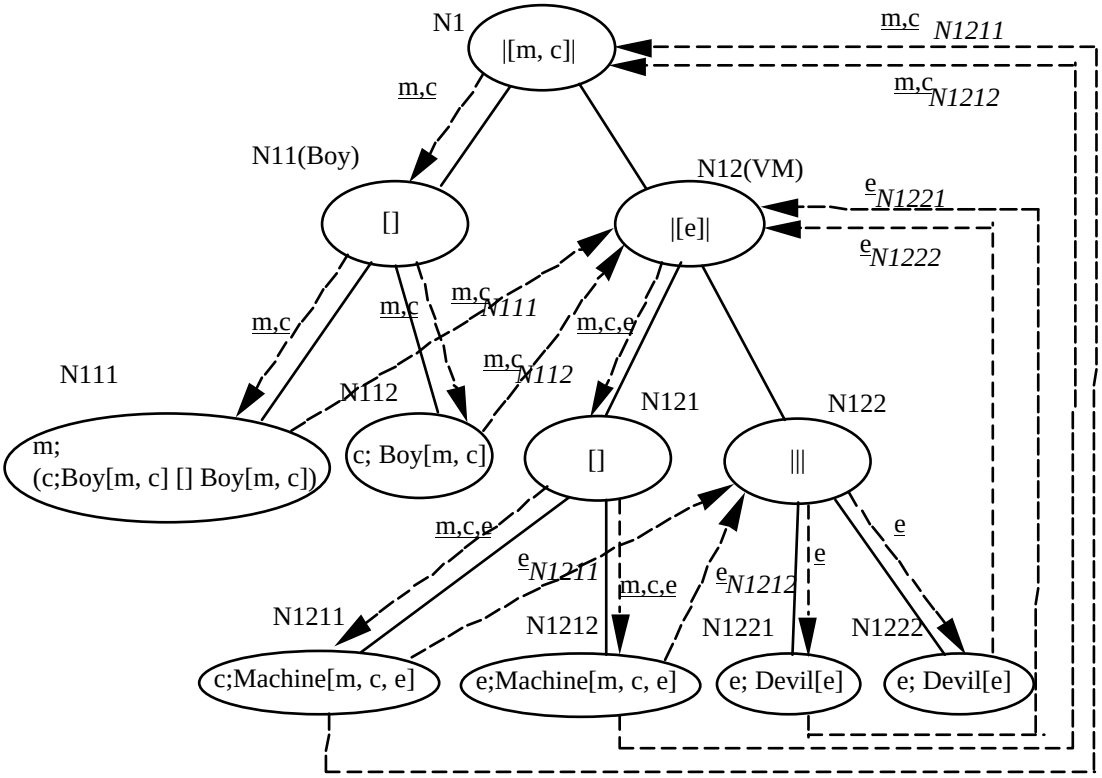


(c) updating after rendezvous happens at gate 'm'


N1(system)
|[m, c]|
(d) growing the nodes
N11, N121, and N122 of (c)
N11(Boy)
N12(VM)
[]
|[e]|
N111(Boy)
N112
N121(Machine)
N122
m;
(c;Boy[m, c] [] Boy[m, c])
c; Boy[m, c]
[]
|||
N1211
N1212
N1221(Devil)
N1222(Devil)
c;Machine[m, c, e]
e;Machine[m, c, e]
e; Devil[e]
e; Devil[e]


N1(system)
|[m, c]|
(e) updating after a
rendezvous happens
at gate ’e’
N11(Boy)
N12(VM)
[]
|[e]|
N111(Boy)
N112
N121(Machine)
N122
m;
(c;Boy[m, c] [] Boy[m, c])
c; Boy[m, c]
|||
N1212(Machine)
N1221(Devil)
N1222(Devil)
m; (c; Machine[m, c, e]
[]
e; Machine[m, c, e])
e; Devil[e]
e; Devil[e]

(f) Activity tree of (d)
including virtual rings



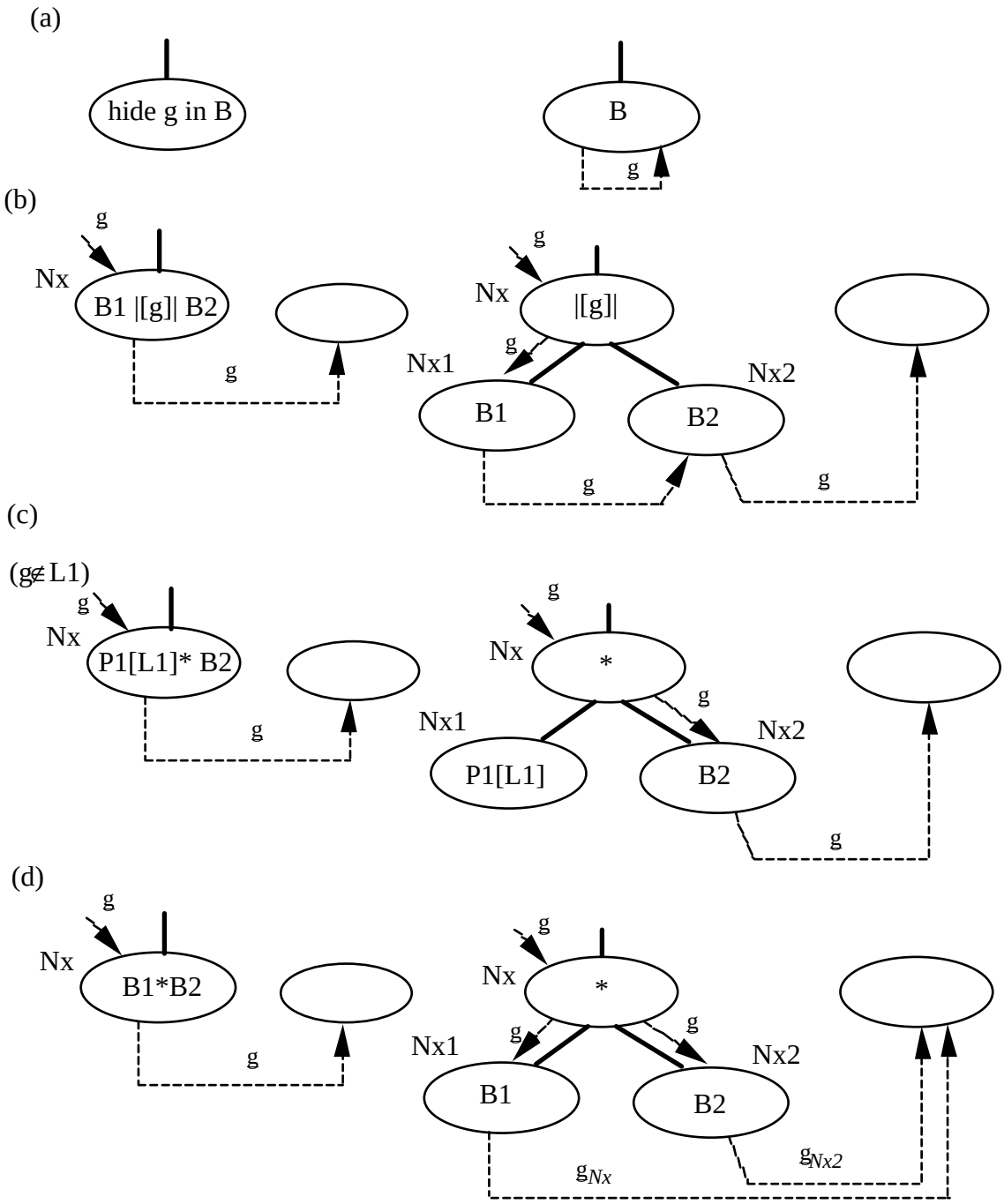
Notation:

node cannot be expanded (internal or terminal leaf node)

node can be expanded (non-terminal leaf node)

internal node which can be replaced by its son

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



Notation : the left side shows a non-terminal node Nx before expansion and
the right side shows the result of the expansion
* denotes one of the operators |||, [>, and [].

Figure 3: Growing steps including ring establishment