Protocol Specification, Testing, and Verification VIII
S. Aggarwal and K. Sabnani (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1988

EXECUTING LARGE LOTOS SPECIFICATIONS

R. Guillemot, M. Haj-Hussein and L.Logrippo

University of Ottawa
Protocols Research Group
Computer Science Department
Ottawa, Ont. Canada K1N 6N5
E-mail: LMLSL@UOTTAWA.BITNET

The features of the University of Ottawa LOTOS interpreter are discussed and it is shown how they can be used in order to simulate a full-sized constraint-oriented LOTOS specification, the specification of the OSI Transport Service Provider. Some problems related to the design of such interpreters are also discussed.

1. INTRODUCTION

In a number of previous papers [LP][LOG][LSU] we discussed the usefulness of the concept of prototyping complex distributed systems by means of executable specifications. In this paper, we report on our experiences towards prototyping a real system, the OSI Transport Service (TS) provider, using the University of Ottawa LOTOS interpreter. We discuss several difficulties that are encountered in this type of task, and some possible solutions, most of which have already been implemented in our interpreter.

2. THE UNIVERSITY OF OTTAWA LOTOS INTERPRETER

The current version of the interpreter has evolved from the structure described in [LOBF]. First, the LOTOS source is submitted to syntax and static semantics checks according to [ISO1]. If the source is found to be correct, it is translated into an 'internal' format which represents the source in the form of a Prolog list, in a 'flattened' name space. This representation is faithful to the source to the point that the latter can be recovered from the internal representation, with the exception of some details such as process functionality, comments, justification, spacing, etc. Therefore, the user never needs to be aware of the internal representation. During this phase, abstract data type equations are oriented as rewriting rules [FEH]. These functions, which constitute the "compiler" part of the interpreter, are programmed in C.

The interpreter runs on the internal representation. It implements in Prolog the LOTOS dynamic semantics specified in [ISO1]. Whenever a value expression has to be evaluated, the value expression evaluator is called. The latter evaluates value expressions by using the rewriting rules according to a "leftmost-innermost" strategy [FEH]. Our name for the interpreter is ISLA, or Interactive System for LOTOS Applications.

The main feature of the interpreter is to allow step-by-step execution of LOTOS specifications. At each step, the user is prompted with a menu of possible "next actions". Each action is displayed with the line number(s) that describe it in the specification (in square brackets in the following listings). The user choses the next action to be executed and, if this action involves a choice of data, he/she is prompted to provide the choice. Choice of data is required not only by actions involving the environment (in which case the user is asked to provide the role of the environment), but also by internal actions where data is to be generated nondeterministically.

At any point during simulation, the user can ask to see the current behavior expression, which describes the current state of the LOTOS machine, and the behaviors that will result by the execution of each one of the possible "next actions". This will be printed in external LOTOS format, rather than in the interpreter's internal representation.

At any time during execution, it is possible to ask the simulator to compute and print the tree of all possible "next actions" from the current point. We call this "symbolic execution tree" because variable values are represented by expressions that may involve other variables. Of course, the calculation of conditions cannot always be completed when it involves variables whose values depend on interactions with the environment. Such guards are assumed to be "true". Since such trees can have large depths and widths, at the time the user requests the calculation of a tree he/she is prompted to provide upper bounds (Annex 1.).

As a practical help for the user, "checkpoints" can also be set, that is, the current behavior expression and/or the sequence of actions executed so far can be saved in memory or on an external file, thereby giving the user the possibility of restarting the simulation some time later.

In order to allow the user to explore several alternative execution paths in the specification, at any point it is possible to obtain a display of the "history" by which the current execution point was reached (possibly including the guards that were evaluated and the instantiations encountered) and therefore to see where a certain variable acquired its value (Annex 2.A). This history will include all the actions performed so far and, depending on the format chosen, may include all the process instantiations and not only for the current path, but also for all paths attempted so far. It is possible to request that the execution go back to some point, where a different value can be chosen.

To help users in dealing with the complexity of LOTOS data representation, a command is provided by which they can ask for the valid syntax of constants to be entered. Constants are syntactically and semantically checked at the time they are entered, that is it is checked that the value entered is appropriate at the time it is entered. Furthermore, shorthands for commonly used constants can be declared. This is necessary, because extremely long constants may be required in specifications. When written in full, the definition of a TS primitive takes, in the average, about 400 characters. Furthermore, the sheer compactness of the LOTOS constraint-oriented specification style is by itself a source of problems. Most actions involve between six and ten processes and guards which, when written in full, are over 2000 characters long. Obviously such displays are impossible to understand. Our interpreter is able to look up the data base of user-defined shorthands and reduce the displays to a form which will be much shorter and much more understandable. Shorthands are distinguished by the fact that their name starts with the character $ (Annex 0.A).

When complex guards are involved in an action, it might be difficult to find values that satisfy them. To help the user in this respect, our interpreter has a feature by which one can ask that a set of guards be evaluated for a set of user-defined values placed in a file, rather than on only one value at the time. When an action requires a tuple of values, for any of these it is possible to provide sets of values instead. The guards are then evaluated on all possible combinations of values taken from the sets. This can be done for one or several next actions. The interpreter will then report for actions and for which tuples of value(s) the evaluation was successful, and all the tuples tried are numbered. Subsequently, the user can ask that action N be executed with tuple Y, where Y will presumably be one of the tuples for which the evaluation succeeded. It is also possible to obtain detailed step-by-step traces of the evaluation process to see why a guard failed. Sets of values have names starting by # (Annex 0.B and 2.B).

Yet another difficulty is presented by the "choice" construct, which does not generate a LOTOS action, however a choice is needed in order to determine the evolution of the simulation. This case is solved by presenting the user with a "pseudo-action" (marked by an asterisk) (Annex 2.C) which requires the input of a value just as regular actions do. The user must keep in mind that in this case the specification allows a range of values for the variable, while the interpretation process restricts this range to one choice only. A resulting deadlock, therefore, does not necessarily imply a deadlock in the specification, because a different choice of values may not lead to a deadlock. An example is provided by the following process, which is part of the specification considered below:

```
process TCAcceptance[t]: noexit :=
    choice AcceptTC: Tids []
        [AcceptTC ne {}] -> i;
            t ?ta:TAddress ?tcei:TCEI ?tsp:TSP
                [IsTCON1(tsp) implies (Tid(ta,tcei) Isin AcceptTC)];
            TCAcceptance[t]
endproc (* TCAcceptance *)
```

In the specification, this process must collaborate with other parallel processes by offering an action on gate t. It is ready to accept a TSDU whose address part is contained in a nondeterministically chosen nonempty set of addresses (ne stands for "not equal"). As it encounters this process, our interpreter will prompt the user to provide a choice for AcceptTC before reaching the next behavior. If the user provides an empty set {}, the guard in the next behavior is evaluated to "false" and deadlock follows because the process does not cooperate on t. This deadlock, however, is not specified according to proper LOTOS semantics because only nonempty sets were acceptable choices for AcceptTC, thus the user erred in providing an empty set. Therefore, the deadlock will simply have to be ignored and a correct choice will have to be provided instead. How to deal with this difficulty is an interesting problem in the design of LOTOS interpreters [VE]. In our system, the user can ask that all guards evaluated after the choice of a value be listed, together with the results of their evaluation. In this way, it is possible to see where the value failed, the interpretation can be backtracked to the choice point, and another value can be selected (Annex 2.D).

3. THE LOTOS OSI TRANSPORT SERVICE SPECIFICATION.

The OSI Transport Service [ISO2] specification was chosen as the basis for this experiment because of several reasons, the most important of which being the fact that it is is possibly the best developed LOTOS specification in existence today. It has a long history [BK][SPLB]. Its current version [ISO4] is based on a document produced by A. Tocher [ISO3] and recently modified and completed by G. Scollo, with the collaboration of an ad-hoc ISO committee. Other specifications, such as the LOTOS Transport Protocol Specification [ISO5] and the LOTOS Session Protocol and Service specifications [SVS] have been based on ideas first developed there.

The TS specification is modularized according to the constraint-oriented specification style [SVS][BB]. In this style, which has become the most currently used specification style in LOTOS, the various requirements of the specification are partitioned in independent processes, where each process is established as the "enforcer" of a constraint. Constraints can relate both to the order in which primitives can appear (this is expressed by the order of LOTOS behavior expressions) and to the possible values of parameters (this is expressed by input predicates). For example, the so-called "local" and "global" service constraints [LOG] are implemented by independent modules, and then the parallel composition of the modules is specified, meaning that the entity must satisfy both local and global constraints.

Unfortunately, because of the length of this specification (1074 lines of LOTOS, without the library or comments), it is not possible to describe it here in any detail, and the reader is assumed to have some previous knowledge of it.

Note that the full specification was used in our experiment, and not just parts of it. However, in order to better understand certain processes, it has often been found useful to disable others. In addition, cases of unguarded recursion that raise the possibility of infinitely branching execution trees [LOBF] were detected by our static semantics analyzer and were guarded by internal actions. Finally, sets of equations were found, that originate lengthy computations. This occurs in relation to the definition of the "Quality of Service" parameters, where an equation expects that a variable be instantiated in a premiss. Extensive backtracking may occur, because the evaluator may have to check a large number of possibilities. Happily, it is possible to rewrite such equations in a more efficiently computable format.

4. TRACING METHODOLOGY

The compiler part of our system performed the syntax and static semantics checks, which enabled us to correct several errors in the specification. For brevity we will not discuss this part of the work.

[ISO2] defines the TS characteristics with the help of the English language and some figures, where [ISO4] defines them as the composition of constraints represented by processes. We experimented in executing the [ISO4] specification with scenarios extracted from [ISO2]. Real "testing" or even systematic checking were not our immediate aims, rather we wanted to measure the usefulness of our tool with respect to these tasks, in order to find out how it could be enhanced to deal with them.

Three methods have appeared appropriate towards this goal.

- The first method involves examination of the symbolic execution trees. Since large trees are impractical to compute and read, and since they contain many uninteresting paths, we compute a tree up to a certain depth, then select a leaf node, compute a tree from that node, etc. The main problem with this approach is that many of the branches are unfeasible because they contain contradictory conditions such as "IsTCONreq(x) and IsTCONind(x)". These conditions cannot be evaluated because they contain variables, thus by our method they are assumed to be true.

- In the second method, which corresponds to the basic way of operating of our interpreter, the user exercises the specification by entering sequences of primitives by hand. In this way it can be found out which primitives are acceptable at a given point, etc. This methodology is very flexible and very useful for a user who is becoming acquainted with the specification but is obviously limited by the speed of the user in reacting to the prompts of the system and entering the data. This method can be very considerably speeded up by using shorthands and automatic evaluation of conditions on sets of values as mentioned above. Furthermore, at any point it is possible to request that the symbolic execution tree from that point be computed.

- Another methodology involves creating "testing processes" to be run in parallel with the specification. A very simple type of test process that we have found to be useful in our work is a process that exercises the specification on finite sequences of actions. Variable parameters can be entered by the user by using a specially designated gate. One can create a library of such processes according to several testing scenarios. Here we anticipated on the development of LOTOS, because we assumed the presence of constructs allowing external references in a specification. It is useful to combine this method with the symbolic execution tree method, in the sense of obtaining the execution tree of a process and its tester. Since the tester provides the values of all the variables, except those depending on choices, the tree is considerably reduced. By inspecting it one can check whether the sequence of data submitted by the tester is accepted.

5. SOME EXAMPLES

Reasons of space make it impossible to show complete examples. We will show only simplified or shortened examples, in edited form.

Example 1 shows a symbolic tree. Example 2 shows several of the major features of the interpreter, as described above. Example 3 shows the steps needed to achieve a connection. Fig. 1 shows a typical screen during a simulation session. Clockwise from top left we see: A list of shorthands; a portion of the execution tree; a menu; and a portion of the specification itself, containing (insert) a test process.

6. IMPLEMENTATION ISSUES AND PERFORMANCE CHARACTERISTICS

As mentioned above, LOTOS semantics was implemented by translating LOTOS inference rules into Prolog. Our first experiences with this method led to disappointing results on specifications of realistic size. The reasons for this were found to reside in the fact that a straightforward one-to-one translation of the inference rules of [ISO1] into Prolog statements causes repeated recursive calls of the inference rules in cases where more than one inference rule is used for the same LOTOS construct. This problem was solved by combining all such rules into one. For example, in [ISO1] three inference rules are used to define the parallel composition operator, and the recursive calls required by the first two are also required by the third. When the three rules are combined in one, the number of recursive calls is reduced by one half. When parallel composition is nested to n levels, this gain is raised to the power of n.

Currently, on the example discussed in this paper, calculation of the next menu takes an average of 5 seconds (as mentioned above, there are exception in cases of equations that were not defined in a computationally efficient way). The system used is a SUN 3/75 running under UNIX and Suntools, while the Prolog system used is Quintus Prolog in compilation mode. This performance appears to be quite adequate for use in the "one-stepper" mode. It is of course inadequate if the simulator is to be used for reachability analysis or generation of large amounts of data. For these purposes, we are envisaging more radical optimizations, namely in the value expression evaluator and in certain frequently used inference rules.

Prolog appeared to be a very appropriate language for this project [LOBF][PA]. Apart from the usefulness of the unification and backtracking mechanisms in this type of application, the high-level style of programming afforded by Prolog allowed a lot of design experimentation to take place, which otherwise would have been prohibitive in a small team such as ours.

Finally, the Suntools package contains several facilities that help in the use of these interactive features.

7. CONCLUSIONS

Executing and debugging constraint-oriented LOTOS specifications of realistic size is a task that involves considerable technical difficulties: after all, the most effective specification style is not necessarily the best for execution. Several closely intertwined processes are being executed at once, each process posing its own constraints on the next possible action. These constraints are expressed in terms of very complex value expressions, which need to be abbreviated in order to be understood. Even after abbreviation, they often remain difficult to read. A great number of them will turn out to be false, because of contradictory conditions. Frequent "choice" constructs force the choice of values that may later be discovered to be mutually incompatible. This leads to situations that appear to the interpreter as deadlocks, although they may simply be the consequence of inappropriate choices, hence the need for frequent backtracking.

As it happens in debugging ordinary programs, a thorough understanding of the logic of the specification is necessary in order to proceed productively, but exercising a specification is one way of gaining insight in its mechanism.

The result of our experience is that the technique is feasible, and that the necessary skills can be learned by good programmers. However much can be done in order to instrument existing interpreters with good interactive debugging aids, and the work reported here is only a first effort in this direction.

As a result of the availability of such tools, it can be expected that it will become customary to execute specifications as they are being written, which may lead to changes in the generally practiced specification style.

ACKNOWLEDGMENTS

Parts of the interpreter were written by J.P. Briand and M. Fehri. B. Stepien has also collaborated in the programming. We especially wish to thank A. Obaid for his continuing collaboration in this project, and for many useful ideas. The colleagues of the LOTOS group at Twente University, who themselves have been involved in the production of a LOTOS interpreter, have been generous in sharing their experiences with us. G. Scollo is to be credited for the time he spent discussing the details of the TS specification. Funding sources include the Natural Sciences and Engineering Research Council of Canada and Bell-Northern Research.

REFERENCES

[BB] Bolognesi, B., and Brinksma, E. Introduction to the ISO Specification Language LOTOS. To appear in Computer Networks and ISDN Systems.

[BK] Brinksma, E., and Karjoth, G. A Specification of the OSI Transport Servicein LOTOS. In: Yemini, Y., Strom, R., and Yemini, S. (eds.) Protocol Specification, Testing, and Verification, IV. North-Holland, 1985 227-251.

[FEH] Fehri, M.C. A System for Validating and Executing LOTOS Data Abstractions (SVELDA). MCS Thesis, University of Ottawa, 1987.

[ISO1] International Organisation for Standardization. Information Processing Systems. Open Systems Interconnection. LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior (ISO DIS 8807), 1987.

[ISO2] International Organisation for Standardization. Open Systems Interconnection. Transport Service Definition (ISO 8072), 1985.

[ISO3] International Organisation for Standardization. Formal Specification in LOTOS of ISO 8072 (ISO/TC 97/SC 6/N 4395), 1986.

[ISO4] International Organization for Standardization. Formal Description of ISO 8072 in LOTOS. (ISO/TC 97/SC 6/WG 4/N 317), 1987.

[ISO5] International Organisation for Standardization. Formal Description of the OSI Connection-Oriented Transport Protocol in LOTOS. (ISO/TC 97/SC 6/WG 4/N 318), 1987.

[LOBF] Logrippo, L., Obaid, A., Briand, J.P., and Fehri, M.C. An Interpreter for LOTOS: A Specification Language for Distributed Systems. To appear in Software - Practice & Experience.

[LOG] Logrippo, L. "Constructive" and "Executable" Specifications of Protocol Services by Using Abstract Data Types and Finite State Transducers. In: Rudin, H. and West, C.H. Protocol Specification, Testing, and Verification, III. North-Holland, 1983, 111-124.

[LP] Logrippo, L. and Probert, R.L. Protocol Specification-Level Validation. In: Sunshine, C. (ed.) Protocol Specification, Testing,, and Verification North-Holland, 1982, 303-304.

[LSU] Logrippo, L., Simon, D., and Ural, H. Executable Description of the OSI Transport Service in LOTOS. In: Yemini, Y., Strom, R., and Yemini, S. (eds.) Protocol Specification, Testing, and Verification, IV. North-Holland, 1985, 279-293.

[PA] Pappalardo, G. Experiences with a Verification and Simulation Tool for Behavioral Languages. In: Rudin, H., and West, C. Protocol Specification, Testing, and Verification, VII. North-Holland, 1987, 251-264.

[SPLB] Scollo, G, Pappalardo, G., Logrippo, L., and Brinksma, E. The OSI Transport Service and its Formal Description in LOTOS. In: Csaba, L. Tarnay, K, and Szentivanyi, T. Computer Network Usage: Recent Experiences North-Holland, 1986, 465-488.

[SVS] Scollo, G., and Van Sinderen, M. On the Architectural Design of the Formal Specification of the Session Standards in LOTOS. In: Sarikaya, B., and Bochmann, G.v. (eds) Protocol Specification, Testing, and Verification, VI. North-Holland, 1987, 3-14.

[VE] Van Eijk, P. Software Tools for the Specification Language LOTOS. Technische Hogeschool Twente, 1988.

ANNEX

0. Common definitions

```
/* These are comments

/* A: Defined constant values for all examples

$1      = Succ(0)          $2     = Succ(Succ(0))                        $P  = /($1,$2)
$adrA  = SomeTAddress    $adrB = AnotherTAddress(SomeTAddress)      $tcei = SomeTCEI
$NODATA = <>                         $DATA1     = Octet(Octet(0,0,0,0,0,0,0,1))
$TDTreq  = TDTreq($DATA1)           $TDTind    = TDTind($DATA1)
$TEXreq  = TEXreq($DATA1)           $TEXind    = TEXind($DATA1)
$TDISreq = TDISreq($DATA1)          $TDISindU = TDISind(User,$DATA1)
$Delays          = Delays(EstDelay($1),TransDelay($1,$1,$1,$1),RelDelay($1,$1))
$Failures        = Failures($P,$P,$P,$P)
$Performance     = Performance($Delays,$Failures,Throughput($1,$1,$1,$1),RER($P,$P))
$TQOS            = TQOS($Performance,Lowest,NoProtection)
$TCONreqAB       = TCONreq($adrB,$adrA,NoTEX,$TQOS,$NODATA)
$TCONindAB       = TCONind($adrB,$adrA,NoTEX,$TQOS,$NODATA)
$TCONrespAB      = TCONresp($adrB,NoTEX,$TQOS,$NODATA)
$TCONconfAB      = TCONconf($adrB,NoTEX,$TQOS,$NODATA)
$PviderGenInd  = TDISind(Provider,$NODATA)
   ...
/* remark: in the examples, identical constant names with suffixes
/*     - x  : define the same primitive with the option UseTEX instead of NoTEX
/*     - BA : define the same primitive with $adrA instead of $adrB

/* B: Defined Sets. U = set union

Set name => #adrs     = { $adrA , $adrB }
Set name => #tspABc   = { $TCONreqAB, $TCONindAB, $TCONrespAB, $TCONconfAB  }
Set name => #tspABcx  = { $TCONreqABx,$TCONindABx,$TCONrespABx,$TCONconfABx }
Set name => #tspdxi   = { $TDTreq,$TDTind,$TEXreq,$TEXind,$TDISreq,$TDISindU,$PviderGenInd }
Set name => #tspAB    =  #tspABc    U   #tspABcx    U   #tspdxi
```

1. A SIMPLE EXAMPLE OF SYMBOLIC EXECUTION

```
Enter a command 'h' for help -> tree TCEPOrdering
Maximum execution depth (default = 5) => 4

1 [role=CallingRole] t ?ta:TAddress ?tcei:TCEI ?tcr:TSP
                                   [and(IsTCONreq(tcr),IsCallingOf(ta,tcr))] [823]
| 1 t ?ta:TAddress ?tcei:TCEI ?tc2:TSP[IsValidTCON2For(tc2,tc1)] [825]
| | 1 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDT(tsp)] [836]
| | | 1 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDT(tsp)] [849]
| | | 2 [x=UseTEX] t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTEX(tsp)] [855]
| | | 3 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDIS(tsp)] [863]
| | 2 [x=UseTEX] t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTEX(tsp)] [836]
| | | 1 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDT(tsp)] [849]
| | | 2 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTEX(tsp)] [855]
| | | 3 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDIS(tsp)] [863]
| | 3 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDIS(tsp)] [863]
| | | 1 exit[864]
| 2 t ?ta:TAddress ?tcei:TCEI ?tsp:TSP[IsTDIS(tsp)] [825]
| | 1 exit[864]
```

2. A SIMPLE EXAMPLE OF STEP BY STEP EXECUTION

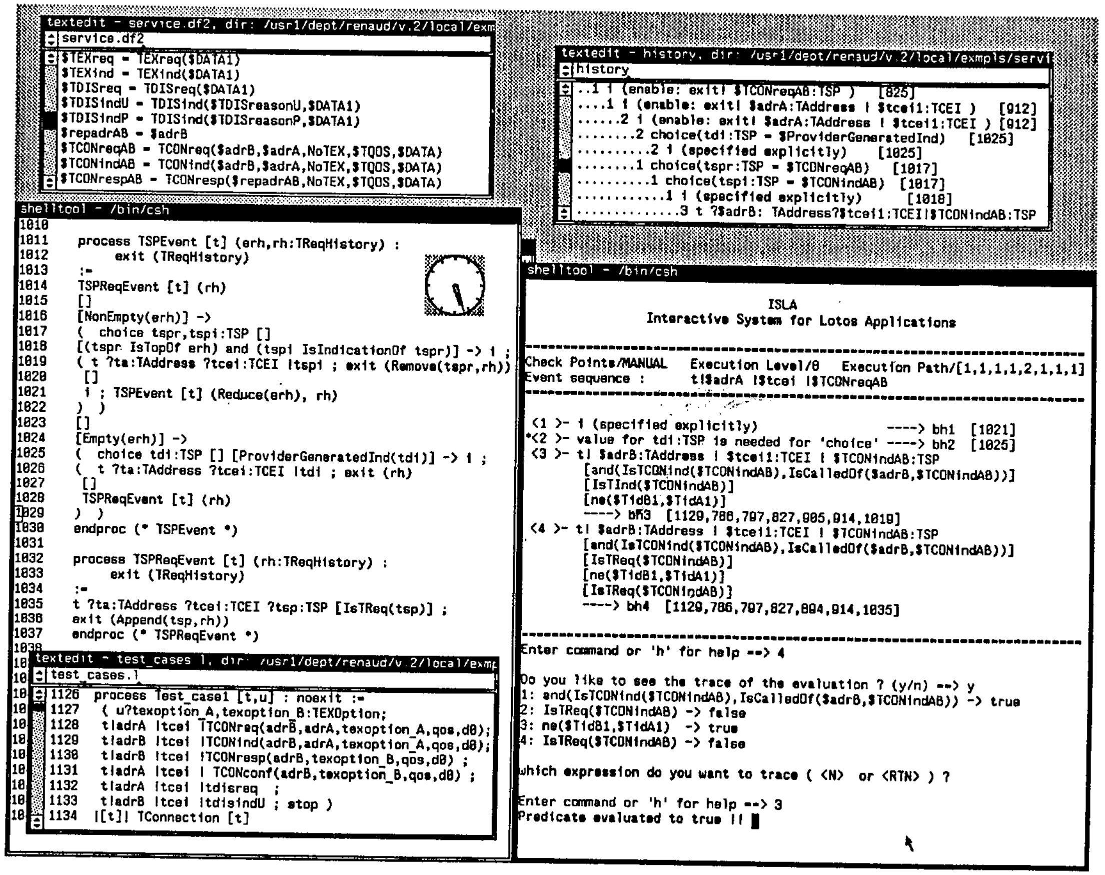
```
Enter a process to be executed ==> TCEP(CallingRole)

=========================================================================
Check Points/MANUAL        Execution Level/0     Execution Path/[]
=========================================================================
 <1 >- t? [ta,ta,ta]:TAddress ? [tcei,tcei,tcei]:TCEI ? [tsp,tsp,tcr]:TSP
          [and(IsTCONreq(tcr),IsCallingOf(ta,tcr))]    ---> bhl  [786,797,823]

=========================================================================
Enter a command or 'h' for help ==> 1

/* A: The user has selected action 1, the only possible one.
/* He is then prompted for the necessary tuple of values

Value for the Synchronization variables: [ta,ta,ta]:TAddress    ==> $adrA
Value for the Synchronization variables: [tcei,tcei,tcei]:TCEI  ==> $tcei
Value for the Synchronization variables: [tsp,tsp,tcr]:TSP      ==> $TCONreqBA


Predicate(s) of action 1 evaluated to false !!
Would you like to see the trace of the evaluation ? (y/n) ==> y
   1: IsTCONreq($TCONreqBA)          -> true
   2: IsCallingOf($adrA,$TCONreqBA) -> false
which expression do you want to trace ( <N>  or <RTN> ) ? 2
 EVAL: IsCallingOf($adrA,$TCONreqBA)
 | EVAL: $TCONreqBA = TCONreq($adrA,$adrB,NoTEX,$TQOS,$DATA)
 FOUND: [] ==> IsCallingOf($adrA,$TCONreqBA) = eq($adrA,$adrB)
 | EVAL: eq($adrA,$adrB)
 | FOUND: [] ==> eq($adrA,$adrB) = false
 | <- eq($adrA,$adrB) = false
 <- IsCallingOf($adrA,$TCONreqBA) = false

Enter command or 'h' for help ==> 1
Value for the Synchronization variables: [ta,ta,ta]:TAddress    ==> $adrA
Value for the Synchronization variables: [tcei,tcei,tcei]:TCEI  ==> $tcei
Value for the Synchronization variables: [tsp,tsp,tcr]:TSP      ==> $TCONreqAB

Predicate(s) of action 1 evaluated to true !!

/* B: After an intermediate action due to an enable we get:
=========================================================================
Check Points/MANUAL        Execution Level/2     Execution Path/[1,1]
Events sequence [1] t !$adrA  !$tcei  !$TCONreqAB
=========================================================================
 <1 >- t! $adrA:TAddress ! $tcei:TCEI ? [tsp,tsp,tc2]:TSP
          [IsValidTCON2For(tc2,$TCONreqAB)]   ---> bhl  [792,803,833]
 <2 >- t! $adrA:TAddress ! $tcei:TCEI ? [tsp,tsp,tsp]:TSP
          [IsTDIS(tsp)]                       ---> bh2  [792,803,863]

=========================================================================
Enter command or 'h' for help ==> 1,2
Value for the Synchronization variables: [tsp,tsp,tcr]:TSP     ==> #tspAB

The existing predicate(s) for action 1: [IsValidTCON2For(tc2,$TCONreqAB)]
1.1> $adrA,     $tcei,     $TCONconfAB  ==> true
1.2> $adrA,     $tcei,     $TCONconfABx ==> true

The existing predicate(s) for action 2: [IsTDIS(tsp)]
2.1> $adrA,     $tceil,     $PviderGenInd ==> true
2.2> $adrA,     $tceil,     $TDISindU ==> true

Enter Action number to execute or 'h' for help ==> 1.2
```

```
/* C:
=========================================================================
Check Points/MANUAL        Execution Level/3     Execution Path/[1,1,1]
Events sequence [1]    t !$adrA  !$tcei  !$TCONreqAB
              [111]  t !$adrA  !$tcei  !$TCONconfABx
=========================================================================
*<1 >- value for x:TEXOption is needed for 'choice'   ---> bh1  [836]
  <2 >- t! $adrA:TAddress ! $tceil:TCEI ? [tsp,tsp,tsp]:TSP
            [IsTDIS(tsp)]                               ---> bh2  [792,803,863]

=========================================================================
Enter command or 'h' for help ==> 1
For "choice", value for variable x:TEXOption must be entered ==>

/* D: If we try NoTEX the system will offer only a disconnection on the endpoint
/* ($adrA,$tcei)
/* The command le (ListEval) helps the user to find  why

                    Evaluated guards & expressions:
              1: IsTEXOptionOf(NoTEX,$TCONconfABx) -> false
              2: CalledRole=CallingRole -> false
              3: CalledRole=CalledRole  -> true

/* So, it's better to go back (command back) and retry with UseTEX
/* After a step due to an enable we get :
=========================================================================
Check Points/MANUAL        Execution Level/5     Execution Path/[1,1,1,1,1]
Event sequence [1]    t !$adrA  !$tcei  !$TCONreqAB
              [111]  t !$adrA  !$tcei  !$TCONconfABx
=========================================================================
  <1 >- t! $adrA:TAddress ! $tcei:TCEI ? [tsp,tsp,tsp]:TSP
            [IsTDT(tsp)]                           ---> bh1  [792,803,849]
  <2 >- t! $adrA:TAddress ! $tcei:TCEI ? [tsp,tsp,tsp]:TSP
            [IsTEX(tsp)]                           ---> bh2  [792,803,855]
  <3 >- t! $adrA:TAddress ! $tcei:TCEI ? [tsp,tsp,tsp]:TSP
            [IsTDIS(tsp)]                          ---> bh3  [792,803,863]

=========================================================================

/* We are now able to transmit normal or expedited data or to generate a
/* disconnection on the endpoint ($adrA,$tcei)
```

3. ESTABLISHMENT OF ONE CONNECTION

```
Enter a process to be executed ==> TConnection

/* generation of a TCONreq on endpoint ($adrA,$tcei)
=================================================================================
Check Points/MANUAL       Execution Level/0     Execution Path/[]
=================================================================================
 <1>- t?[ta,ta,ta,ta]:TAddress ?[tcei,tcei,tcei,tcei]:TCEI ?[tsp,tsp,tsp,tcr]:TSP
        [and(IsTCONreq(tcr),IsCallingOf(ta,tcr))]
        [IsTReq(tsp)]                          ---> bh1  [786,797,823,870]
 <2>- t?[ta,ta,ta,ta]:TAddress ?[tcei,tcei,tcei,tcei]:TCEI ?[tsp,tsp,tsp,tci]:TSP
        [and(IsTCONind(tci),IsCalledOf(ta,tci))]
        [IsTReq(tsp)]                          ---> bh2  [786,797,827,870]
=================================================================================
Enter Action number to execute or 'h' for help ==> 1
Value for the Synchronization variables: [ta,ta,ta,ta]:TAddress     ==> $adrA
Value for the Synchronization variables: [tcei,tcei,tcei,tcei]:TCEI ==> $tcei
Value for the Synchronization variables: [tsp,tsp,tsp,tcr]:TSP      ==> $TCONreqAB

Predicate evaluated to true !


/* then, after a few steps due to choice and enable constructs,
/* during which a TCONindication is generated,
/* we are able to generate the TCONresponse at the other end of the connection

=================================================================================
Check Points/MANUAL      Execution Level/7     Path/[1,3,3,3,6,1,1]
Events sequence [1] t ?$adrA:TAddress ?$tceil:TCEI ?$TCONreqAB:TSP
              [1,3,3,3,6] t ?$adrB:TAddress ?$tceil:TCEI !$TCONindAB:TSP
=================================================================================

   <1>- i (enable: exit !$adrA:TAddress !$tcei:TCEI)     ----> bh1  [912]
   <2>- i (enable: exit !$adrB:TAddress !$tcei:TCEI)     ----> bh2  [916]
   <3>- i (enable: exit !NoTReqs:TReqHistory)     -----> bh3  [1019]
   <4>- i (enable: exit !$adrA:TAddress !$tcei:TCEI)     ----> bh4  [912]
 * <5>- value for tdi:TSP is needed for 'choice'     ----> bh5  [1025]
   <6>- t !$adrA:TAddress !$tcei:TCEI ?[tsp,tsp,tc2,tsp,tsp,tsp]:TSP
          [IsValidTCON2For(tc2,$TCONreqAB)]
          [IsTReq(tsp)]
          [ne($TidA1,$TidA1)]
          [IsTReq(tsp)]     ----> bh6 **false**  [792,803,833,894,914,1034]
   <7>- t !$adrA:TAddress !$tcei:TCEI ?[tsp,tsp,tsp,tsp,tsp,tsp]:TSP
          [IsTDIS(tsp)]
          [IsTReq(tsp)]
          [ne($TidA1,$TidA1)]
          [IsTReq(tsp)]     ----> bh7 **false**  [792,803,863,894,914,1034]
   <8>- t !$adrB:TAddress !$tcei:TCEI ?[tsp,tsp,tc2,tsp,tsp,tsp]:TSP
          [IsValidTCON2For(tc2,$TCONindAB)]
          [IsTReq(tsp)]
          [ne($TidB1,$TidA1)]
          [IsTReq(tsp)]     ----> bh8  [792,803,833,894,914,1034]
   <9>- t !$adrB:TAddress !$tcei:TCEI ?[tsp,tsp,tsp,tsp,tsp,tsp]:TSP
          [IsTDIS(tsp)]
          [IsTReq(tsp)]
          [ne($TidB1,$TidA1)]
          [IsTReq(tsp)]     ----> bh9  [792,803,863,894,914,1034]

=================================================================================
/* Actions 8 and 9 can succeed and lead to the generation of a TCONresp or a TDISreq
/*   on endpoint ($adrB,$tcei)
/* Actions 6 and 5 are false (an instantiated predicate was evaluated to false)

Enter Action number to execute or 'h' for help ==> 8
Value for the Synchronization variables: [ta,ta,ta,ta]:TAddress     ==> $adrB
Value for the Synchronization variables: [tcei,tcei,tcei,tcei]:TCEI ==> $tcei
Value for the Synchronization variables: [tsp,tsp,tsp,tcr]:TSP      ==> $TCONrespAB

Predicate evaluated to true !
```

```
/* A: The history given below shows the steps done in order to obtain
/* a transfer of data between the two peer entities

cp[    TConnection[t]
---

 |1 t ?$adrA:TAddress ?$tceil:TCEI ?$TCONreqAB:TSP
      [and(IsTCONreq($TCONreqAB),IsCallingOf($adrA,$TCONreqAB))]
      [IsTReq($TCONreqAB)]    [786,797,823,870]
 |..3 choice(tspr:TSP = $TCONreqAB)  [1017]
 |....3 choice(tspi:TSP = $TCONindAB)  [1017]
 |......3 i (specified explicitly)  [1018]
1|........6 t ?$adrB:TAddress ?$tceil:TCEI !$TCONindAB:TSP
                [and(IsTCONind($TCONindAB),IsCalledOf($adrB,$TCONindAB))]
                [IsTInd($TCONindAB)]
                [ne($TidB1,$TidA1)]   [786,797,827,905,914,1019]
 |..........1 i (enable: exit !$TCONreqAB:TSP)  [825]
 |............1 i (enable: exit !$TCONindAB:TSP)  [829]
2|..............8 t !$adrB:TAddress !$tceil:TCEI ?$TCONrespAB:TSP
                          [IsValidTCON2For($TCONrespAB,$TCONindAB)]
                          [IsTReq($TCONrespAB)]
                          [ne($TidB1,$TidA1)]
                          [IsTReq($TCONrespAB)]   [792,803,833,894,914,1034]
 |................1 choice(x:TEXOption = NoTEX)  [836]
 |..................1 i (enable: exit !NoTEX:TEXOption)  [836]
 |....................1 i (enable: exit !$adrA:TAddress !$tceil:TCEI)  [912]
 |......................1 i (enable: exit !$adrB:TAddress !$tceil:TCEI)  [916]
 |........................1 i (enable: exit !NoTReqs:TReqHistory)  [1019]
 |..........................2 i (enable: exit !$adrB:TAddress !$tceil:TCEI)  [916]
 |............................2 i (enable: exit !$adrA:TAddress !$tceil:TCEI)  [912]
 |..............................2 i (enable: exit !Append($TCONrespAB,NoTReqs):TReqHistory) [1036]
3|................................2 choice(tspr:TSP = $TCONrespAB)  [1017]
 |..................................2 choice(tspi:TSP = $TCONconfAB)  [1017]
 |....................................2 i (specified explicitly)  [1018]
4|......................................4 t !$adrA:TAddress !$tceil:TCEI !$TCONconfAB:TSP
                                            [IsValidTCON2For($TCONconfAB,$TCONreqAB)]
                                            [IsTInd($TCONconfAB)]   [792,803,833,905,792,803,1019]
 |........................................3 i (enable: exit !NoTReqs:TReqHistory) [1019]
 |..........................................1 choice(x:TEXOption = NoTEX)  [836]
 |............................................1 i (enable: exit !NoTEX:TEXOption) [836]
5|..............................................3 t !$adrA:TAddress !$tceil:TCEI ?$TDTreq:TSP
                                                      [IsTDT($TDTreq)]
                                                      [IsTReq($TDTreq)]
                                                      [IsTReq($TDTreq)] [792,803,849,894,792,803,1034]
 |................................................1 i (enable:exit!Append($TDTreq,NoTReqs):TReqHistory)
 |..................................................1 choice(tspr:TSP = $TDTreq)  [1017]
 |....................................................1 choice(tspi:TSP = $TDTind)  [1017]
 |......................................................1 i (specified explicitly)  [1018]
6|********************************************************7 t!$adrB:TAddress!$tceil:TCEI!$TDTind:TSP
                                                                  [IsTDT($TDTind)] [IsTInd($TDTind)]
                                                                  [792,803,849,905,792,803,1019]
```



```
textedit - service.df2, dir: /usr1/dept/renaud/v.2/local/exm
service.df2
$TEXreq = TEXreq($DATA1)
$TEXind = TEXind($DATA1)
$TDISreq = TDISreq($DATA1)
$TDISindU = TDISind($TDISreasonU,$DATA1)
$TDISindP = TDISind($TDISreasonP,$DATA1)
$repadrAB = $adrB
$TCONreqAB = TCONreq($adrB,$adrA,NoTEX,$TQOS,$DATA)
$TCONindAB = TCONind($adrB,$adrA,NoTEX,$TQOS,$DATA)
$TCONrespAB = TCONresp($repadrAB,NoTEX,$TQOS,$DATA)
```

```
textedit - history, dir: /usr1/dept/renaud/v.2/local/exmpls/servi
history
..1 i (enable: exit! $TCONreqAB:TSP )   [825]
....1 i (enable: exit! $adrA:TAddress ! $tcei1:TCEI )   [912]
......2 i (enable: exit! $adrA:TAddress ! $tcei1:TCEI ) [912]
........2 choice(tdi:TSP = $ProviderGeneratedInd)   [1025]
..........2 i (specified explicitly)   [1025]
........1 choice(tspr:TSP = $TCONreqAB)   [1017]
..........1 choice(tspi:TSP = $TCONindAB)  [1017]
............1 i (specified explicitly)     [1018]
..............3 t ?$adrB: TAddress?$tcei1:TCEI!$TCONindAB:TSP
```

```
shelltool - /bin/csh
1010
1011   process TSPEvent [t] (erh,rh:TReqHistory) :
1012        exit (TReqHistory)
1013   :=
1014   TSPReqEvent [t] (rh)
1015   []
1016   [NonEmpty(erh)] ->
1017   (  choice tspr,tspi:TSP []
1018   [(tspr IsTopOf erh) and (tspi IsIndicationOf tspr)] -> i ;
1019   ( t ?ta:TAddress ?tcei:TCEI !tspi ; exit (Remove(tspr,rh))
1020    []
1021    i ; TSPEvent [t] (Reduce(erh), rh)
1022   )  )
1023   []
1024   [Empty(erh)] ->
1025   (  choice tdi:TSP [] [ProviderGeneratedInd(tdi)] -> i ;
1026   (  t ?ta:TAddress ?tcei:TCEI !tdi ; exit (rh)
1027    []
1028    TSPReqEvent [t] (rh)
1029   )  )
1030   endproc (* TSPEvent *)
1031
1032   process TSPReqEvent [t] (rh:TReqHistory) :
1033        exit (TReqHistory)
1034   :=
1035   t ?ta:TAddress ?tcei:TCEI ?tsp:TSP [IsTReq(tsp)] ;
1036   exit (Append(tsp,rh))
1037   endproc (* TSPReqEvent *)
```

```
textedit - test_cases 1, dir: /usr1/dept/renaud/v 2/local/exm
test_cases.1
1126  process Test_case1 [t,u] : noexit :=
1127    ( u?texoption_A,texoption_B:TEXOption;
1128    t!adrA !tcei !TCONreq(adrB,adrA,texoption_A,qos,d0);
1129    t!adrB !tcei !TCONind(adrB,adrA,texoption_A,qos,d0);
1130    t!adrB !tcei !TCONresp(adrB,texoption_B,qos,d0) ;
1131    t!adrA !tcei ! TCONconf(adrB,texoption_B,qos,d0) ;
1132    t!adrA !tcei !tdisreq  ;
1133    t!adrB !tcei !tdisindU ; stop )
1134  |[t]| TConnection [t]
```

```
shelltool - /bin/csh
                              ISLA
              Interactive System for Lotos Applications

---------------------------------------------------------------------------
Check Points/MANUAL   Execution Level/8   Execution Path/[1,1,1,1,2,1,1,1]
Event sequence :      t!$adrA !$tcei !$TCONreqAB
---------------------------------------------------------------------------

 <1 >- i (specified explicitly)                           ----> bh1  [1021]
*<2 >- value for tdi:TSP is needed for 'choice' ----> bh2  [1025]
 <3 >- t! $adrB:TAddress ! $tcei1:TCEI ! $TCONindAB:TSP
        [and(IsTCONind($TCONindAB),IsCalledOf($adrB,$TCONindAB))]
        [IsTInd($TCONindAB)]
        [ne($TidB1,$TidA1)]
        ----> bh3  [1129,786,797,827,905,914,1019]
 <4 >- t! $adrB:TAddress ! $tcei1:TCEI ! $TCONindAB:TSP
        [and(IsTCONind($TCONindAB),IsCalledOf($adrB,$TCONindAB))]
        [IsTReq($TCONindAB)]
        [ne($TidB1,$TidA1)]
        [IsTReq($TCONindAB)]
        ----> bh4  [1129,786,797,827,894,914,1035]

---------------------------------------------------------------------------
Enter command or 'h' for help ==> 4

Do you like to see the trace of the evaluation ? (y/n) ==> y
1: and(IsTCONind($TCONindAB),IsCalledOf($adrB,$TCONindAB)) -> true
2: IsTReq($TCONindAB) -> false
3: ne($TidB1,$TidA1)  -> true
4: IsTReq($TCONindAB) -> false

which expression do you want to trace ( <N>  or <RTN> ) ?

Enter command or 'h' for help ==> 3
Predicate evaluated to true !!
```

Figure 1