Automatic Analysis and Test Case Derivation for a Restricted Class of LOTOS Expressions with Data Parameters *

Teruo Higashino† and Gregor v. Bochmann††

30th November 1992

†: Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University,
Machikaneyama 1-1, Toyonaka, Osaka, 560, Japan
Phone : +81-6-850-3069 Fax : +81-6-850-3059
E-mail : higashino@ics.osaka-u.ac.jp

††: Département d'IRO, Université de Montréal,
C.P. 6128, Succursale A, Montréal, Québec, H3C 3J7, Canada
Phone : +1-514-343-7484 Fax : +1-514-343-5834
E-mail : bochmann@iro.umontreal.ca

* : This work was started during 1990 when T. Higashino was on leave at the University of Montreal.

Abstract

In this paper, we propose an automatic analysis and test case derivation method for LOTOS expressions with data values. We introduce the class of P-LOTOS expressions where the data types are restricted to the integer and boolean types and the operators of the integers are restricted to addition, subtraction and comparison. For this class, we give an algorithm for deriving a set of test cases (a test suite). The algorithm is carried out by using a decision procedure for integer linear programming problems. We also give solutions for the deadlock detection problem, the detection of non-executable branches and the detection of non-deterministic behaviours. We have implemented a tool for the analysis and test selection based on our techniques. The derivation of a test suite for a simplified Session protocol is described as an example.

Index Terms

Automatic Test Case Derivation, Communication Protocols, Conformance Testing,
Formal Description Techniques, LOTOS, Protocol Models

1. Introduction

Precise specifications are essential for the design and implementation of distributed systems and communication networks. They are important during the validation of the system design, the implementation development and conformance testing phase [Boch 90]. The use of formal description techniques (FDT's) allows the automation of certain aspects of these activities. For the description of OSI communication protocols and services, the standardized languages Estelle [ISO 89b], LOTOS [ISO 89a] and SDL [CCITT 88] are proposed. In this paper, we are particularly concerned with the following questions:

(1) Are all branches of the specification executable ?

(2) Does a given specification contain a deadlock ?

(3) Does the specification contain non-deterministic behaviors ?

(4) Can one derive a test suite that covers all branches of the specification ?

While the first three questions relate mainly to the system design phase, the last question is important for conformance testing, which is of large interest in the context of standardized communication protocols, in particular OSI.

While the above questions can be solved for the simple specification model of finite state machines, they become usually undecidable for more complex languages, such as the FDT's mentioned above, because of the data parameters. Already the inclusion of integer parameters and their related operations makes many questions undecidable.

The selection of appropriate test cases is an important issue for conformance testing of communication protocols, as well as for software engineering. In the software engineering context, it is well known that the problem of deciding whether a given branch of a program is executable, and if yes, which test inputs would lead to the execution of this branch, is undecidable. Therefore it is understandable that most work on test suite development for communication protocols assumes that the protocol is specified in a state transition model without interaction parameters. Many methods exist for test selection for finite state machine (FSM) specifications (for on overview, see for instance [Fuji 91a]). Certain authors have considered extended finite state machine (EFSM) specifications

which include interaction parameters and additional state variables. Usually, the data flow relations between input/output parameters and state variables are considered in the test selection process [Sari 87], however, it is generally assumed that the transitions do not contain enabling conditions depending on the additional state variables, or such dependencies are treated in an informal manner. The situation is similar for the work on test selection based on LOTOS specifications. Several authors limit their attention to basic LOTOS, ignoring interaction parameters [Brin 88, Weze 89, Lang 89]. The approach of [TrSa 89] considers the data flow relation automatically, while [Tret 89] leaves certain aspects to be carried out interactively, as certain problems can not be solved automatically in the general case.

In this paper, we show that the problems mentioned above, including the problem of test suite development with interaction parameters, can be completely automated if the power of the underlying specification language is sufficiently restricted. We use in this paper the notation of LOTOS and consider a subset of the language where the data parameters are restricted to values belonging to the integer and boolean types, and the operations of the integers are restricted to the operations addition, subtraction and comparison. Integers with these restricted operations were first considered by Presburger [Pres 29]; therefore we call our LOTOS subset "P-LOTOS". It is known that the theory of integers with addition is decidable [Oppe 78]. This is the reason that many questions related to specifications written in P-LOTOS are also decidable, as further discussed in this paper.

In [HiBo 92], we introduced a method to generate test cases for finite P-LOTOS expressions. In this paper, we generalize the method so that we can treat not only the test case generation problem but also the analysis problems such as the above (1), (2) and (3). The class is also extended to infinite P-LOTOS expressions. The paper is structured as follows. The definition of P-LOTOS expressions and the corresponding extended labeled transition systems (ELTS's) are given in Section 2. In Section 3, we introduce the problems (1) through (4) above in more details by discussing some simple examples in the context of LOTOS. In Sections 4, we provide the basic idea to solve the above four problems. If the control flow part of a given P-LOTOS expression is finite, then all problems are decidable. We show them in Section 5. Although it is shown that the problems are undecidable in general for infinite control flows, they can be solved algorithmically if the control flow is restricted to the equivalent of finite state machines (FSM's). Some solutions for infinite

systems are described in Section 6. As an example, the test suite derivation for a simplified Session protocol is described in Section 7. In Section 8, a tool for the analysis and test selection based on our techniques is explained.

2. P-LOTOS expressions and corresponding extended LTS's

2.1 P-LOTOS expressions

In this section, we define the class of LOTOS expressions considered in this paper. In general, we assume that a LOTOS expression "t" consists of a tuple "<P_0,P_1(..),..., P_k(..)>" of one main process P_0 and some sub-processes P_1(..),...,P_{k-1}(..) and P_k(..). We use a slightly simplified syntax and do not write the gate declarations in the process definitions. We assume that the gates are globally defined.

[Example 2.1]

t_1 = <R,D(w)>

R := f?x:int[-8≤x≤8] ; g!x ; h?y:int[-8≤y≤8] ;

(([y≥0 and x=y] -> k!x ; **stop**) [] ([y≤-1] -> k!y ; D(x-1)))

D(w:int) := a?z:int ; (b!z ; **stop** [] ([w≥7] -> c!z ; D(w-1))) []

In Example 2.1, the process R is the main process of t_1 and the process D(w) is a sub-process. Here, "f?x" and "h?y" are input events, and "g!x" is an output event. The process R is finished when it executes "**stop**". Some LOTOS operators are used for specifying the temporal ordering of execution of the events. Let B, B_1 and B_2 denote behaviour expressions. A behavior expression "a ; B" represents that "B" is executable after the event "a" is executed. A behavior expression "B_1 [] B_2" represents that either "B_1" or "B_2" is executed. A behavior expression "B_1 |[g_1,..,g_n]| B_2" represents that both "B_1" and "B_2" are executable in parallel. The events in "B_1" and "B_2" communicating via the gates in the set {g_1,..,g_n} must be executed as rendezvous interactions. A behavior expression "B_1 >> B_2" represents that "B_2" is executable after the execution of "B_1" is finished successfully. An internal event "**exit**" represents the *successful* termination of a behavior expression. The expressions appearing in input/output events are called "input/output parameters". The predicates, such as [-8≤x≤8] and [y≥0 and x=y], are called "guards". The behaviour expression "[y≥0 and x=y] -> k!x ; **stop**" represents that the event "k!x" is executable if and only if the

predicate [y≥0 and x=y] holds. The variable "w" in D(w:int) is called a formal process parameter of the process D, and the expression "x-1" in D(x-1) is called an actual process parameter of the process D. Although the main process is not required to have its formal process parameters, all sub-processes may have their formal process parameters. We say that a LOTOS expression "t" is finite if and only if it can execute only a finite number of events. Otherwise, we say that "t" is infinite.

[Definition 2.1]

A term which consists of integers, variables of integer type, and operators "+" and "-" is called a P-term ("P" stands for Presburger who first studied this limited arithmetic). A P-sentence is defined inductively as follows.

(A) If t1 and t2 are P-terms, then "t1=t2", "t1<t2", "t1≤t2", "t1≥t2" and "t1>t2" are P-sentences.

(B) If α and β are P-sentences, then "(α) and (β)", "(α) or (β)", "not (α)", "(α) ⊃ (β)" are P-sentences. []

For example, "x+y-3" and "(x≥y-z) or (z=w)" are a P-term and a P-sentence, respectively. However, "x^2+2x-3=0" is not a P-sentence because multiplication is used.

[Definition 2.2]

A LOTOS expression "t=<P_0,P_1(..),...,P_k(..)>" is called a "P-LOTOS expression" if it satisfies the following restrictions.

(1) All guards in the LOTOS expression are described as P-sentences.

(2) The data types of all formal process parameters of the sub-processes P_1,...,P_{k-1} and P_k are integers, and all actual process parameters in the LOTOS expression are described as P-terms.

(3) All input/output parameters of the events are described as P-terms. []

The LOTOS expression t_1 = <R,D(w)> in Example 2.1 is a P-LOTOS expression. Although only integer and boolean types are treated in our P-LOTOS expressions, we can also treat enumeration types, such as for instance "PDUtype = (CR, CC, DR, DC, DT)". Such enumeration types are very common in most specifications. The values of an enumeration type may be represented as integer values, and the operators included in P-LOTOS are sufficient to handle such values. Therefore, P-LOTOS expressions have considerable power for the purpose of system description.

2.2 Extended labeled transition system

In this section, for a given LOTOS expression "t=<P_0,P_1(..),...,P_k(..)>", we will define a tree called an "extended labeled transition system" which represents the possible sequences of events that may be executed for the LOTOS expression "t". First, we will explain the outline of our extended labeled transition system ELTS(t) for a LOTOS expression "t", and then, we will give its formal definition.

Let LTS(t) denote the general labeled transition system [ISO 89a] for the LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>". If "t" is a Basic LOTOS expression (i.e. without input/output parameters), then the ELTS(t) is the same as the tree representation of the LTS(t). That is, each node in the ELTS(t) has a label which is a behaviour expression. The label of the root node in the ELTS(t) is the main process "P_0". Let "*n*" and "*label*(*n*)" denote a node in the ELTS(t) and its label, respectively. Suppose that *label*(*n*) is "B". If an event "a" is executable for the behaviour expression "B" and "B_a" denotes the behaviour expression after "a" is executed for "B", then there is a node "n_a" whose label is "B_a" which is a child node of the node "*n*" in ELTS(t).

If "t" is not in Basic LOTOS, that is, if "t" contains data values, then there exists a little difference between the ELTS(t) and LTS(t). Suppose that the label of a node "*n*" in LTS(t) is B and that B is "f!x[0≤x]; B'(x)" where "x" is a variable of the type integer. Since "f!0", "f!1" , "f!2",... are executable at the node *n*, the node *n* has infinitely many children whose labels are B'(0), B'(1), B'(2),..., respectively. However, in the corresponding ELTS(t), the number of children for the node *n* is only one. Let *n*' denote this child node. We define that *label*(*n*') is B'(x), and that the relation B-<f!x, [0≤x]>->B'(x) holds between the two behaviour expressions B and B'(x). That is, the relation B-<f!x, [0≤x]>->B'(x) represents that "f!k" is executable for B if the integer "k" is greater than or equal to 0, and that B'(k) is the behaviour expression after "f!k" is executed. With the edge $n \rightarrow n'$, we associate two labels *Event*($n \rightarrow n'$) and *Cond*($n \rightarrow n'$). The labels *Event*($n \rightarrow n'$) and *Cond*($n \rightarrow n'$) represent the event "f!x" and the condition "[0≤x]", respectively. For instance, Fig. 1 is the ELTS(t_1) for the LOTOS expression t_1 = <R,D(w)> in Example 2.1.

Next, we will give the formal definition of our extended labeled transition systems. First, for two behaviour expressions B and B', we define the relation B-<a,c>->B' by using the axioms and

inference rules of Table 1. The extended labeled transition system is then defined by using this relation as follows.

[Definition 2.3]

The extended labeled transition system ELTS(t) for a given LOTOS expression "t=<P_0,P_1(..),.., P_k(..)>" is a tree (in general, infinite tree) satisfying the following conditions.

(1) Each node *n* in the ELTS(t) has a label "*label*(*n*)" which is a behaviour expression. The label of the root node is the main process "P_0".

(2) Let *n* denote a node in the ELTS(t) and suppose that *label*(*n*) is B. The node *n* has a child node *n'* whose label is B' if and only if there exists a behaviour expression B' satisfying the relation B-<a,c>->B'.

(3) Let *n* and *n'* denote a node and its child node in the ELTS(t), and suppose that *label*(*n*)=B, *label*(*n'*)=B' and the relation B-<a,c>->B' hold. The edge $n \rightarrow n'$ has two labels *Event*($n \rightarrow n'$) and *Cond*($n \rightarrow n'$) where *Event*($n \rightarrow n'$) is "a" and *Cond*($n \rightarrow n'$) is "c". []

For example, in Fig. 1, *label*(m_2) is "g!x ; h?y:int[-8≤y≤8] ; ...)". At the nodes m_6 and m_9, the same process "D" is invoked. The variables "z" of two events "a?z" in the processes D(x-1) and D(x-2) must be treated as different variables. In order to distinguish the two variables "z", the variable "z" in the second process D(x-2) is replaced by a new variable "z1" at the node m_9 using an inference rule in Table 1. Then, *Event*($m_9 \rightarrow m_{10}$) is "a?z1".

3. Problems related to specification analysis and test selection

Since a LOTOS expression describes non-determinism and parallelism, several event sequences are executable in general. The ELTS(t) described in Section 2 represents the set of event sequences which cover all branches of a given specification as a tree structure. In order to show an implementation conforms a given specification, we need to check whether the implementation executes correctly the event sequence corresponding to each path in a given ELTS(t). Such event sequence is called a test case. In this section, first we explain test cases and then we explain the four questions described in Section 1.

Let us consider the LOTOS expression t_1 = <R,D(w)> in Example 2.1, and let T_R denote the sequence of the output events "f!8 ; g!8 ; h!8 ; k!8". If R and "T_R ; **stop**" are executed in parallel, that is, if R || (T_R ; **stop**) is executed, then the events on the path from the root node m_1 to the node m_5 in the ELTS(t_1) in Fig. 1 are executed sequentially as follows.

R || (T_R ; **stop**) -f!8-> (g!8 ; h?y:int[-8≤y≤8] ; ...) || (g!8 ; h!8 ; k!8 ; **stop**)

-g!8-> (h?y:int[-8≤y≤8] ; ...) || (h!8 ; k!8 ; **stop**)

-h!8-> (([8≥0 and 8=8]->k!8 ; **stop**) [] ([8≤-1]->k!8 ; D(8-1))) || (k!8 ; **stop**)

-k!8-> **stop** || **stop**

This means that T_R makes the process R trace the path from the root node m_1 to the node m_5 in Fig. 1. In this paper, a sequence of output events, such as T_R, is called a test case. Next, we define the test cases treated in this paper more precisely.

Let s_1 be the root node of the ELTS(t) for a given LOTOS expression "t=<P_0,P_1(..),..., P_k(..)>", and let s_1,s_2,s_3..s_{n-1},s_n be a path from the root node s_1 to a node s_n in the ELTS(t). For this path s_1,s_2,s_3..s_{n-1},s_n, we assume that the following relations hold.

$label(s_1)$ -<a_1,c_1>->$label(s_2)$ -<a_2,c_2>->$label(s_3)$....$label(s_{n-1})$ -<a_{n-1},c_{n-1}>->$label(s_n)$

Let $\alpha_{[s1,sn]}(x_1,...,x_k)$ denote the sequence of the output events which is obtained by replacing all input symbols "?" in the sequence "a_1 ; ... ; a_{n-1}" by the output symbols "!" and deleting all internal events from the sequence. Here, $x_1,...,x_k$ are variables appearing in the sequence "a_1 ; ... ; a_{n-1}", and let $\alpha_{[s1,sn]}(x_1/n_1,...,x_k/n_k)$ denote the sequence of the output events which is obtained by substituting the integer values $n_1,...,n_k$ for the variables $x_1,...,x_k$, respectively. Let $\Psi s_n(x_1,...,x_k)$ denote the conjunction of the conditions $c_1,...,c_{n-1}$. In order to execute the sequence $\alpha_{[s1,sn]}(x_1/n_1,...,x_k/n_k)$ for the behaviour expression "P_0 || ($\alpha_{[s1,sn]}(x_1/n_1,...,x_k/n_k)$;**stop**)" and trace the path from the root node s_1 to the node s_n in the ELTS(t), the value of $\Psi s_n(n_1,...,n_k)$ must be true. In this paper, the predicate $\Psi s_n(x_1,...,x_k)$ is called a "reachability condition from the root node s_1 to the node s_n in the ELTS(t)".

[Definition 3.1 (Test case)]

If the value of the reachability condition $\Psi s_n(n_1,...,n_k)$ from the root node s_1 to a node s_n in a given ELTS(t) is true, then the sequence $\alpha_{[s1,sn]}(x_1/n_1,...,x_k/n_k)$ is called a "test case to trace the path from the root node s_1 to the node s_n in the ELTS(t)". []

[Problem 1 (The test case derivation problem)]

The test case derivation problem is the problem for deriving a test case to trace the path from the root node to a given node in an ELTS(t). []

In general, for a given node s_n in an ELTS(t), there may not exist a test case to trace the root node to the node s_n. For example, consider the following LOTOS expression t_2 = <S>.

[Example 3.1]

t_2 = <S>

S := f ? x : int [-2≤x≤2] ; (([x≥5] -> (p ! x ; **stop**)

[] ([x≥0] -> (**i** ; g ! x ; **stop**)

[] ([x≤0] -> g ! x ; q ! z [z=-x] ; **stop**)) []

The ELTS(t_2) is described in Fig. 2. Since an input for "f?x" is less than or equal to 2, the condition [x≥5] is always false. That is, there is no test case to trace the path from the root node to the node s_3 in the ELTS(t_2) and we say that the branch $s_2 \rightarrow s_3$ is non-executable.

[Problem 2 (The non-executable branch detection problem)]

The non-executable branch detection problem is the problem for deciding whether a given LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>" contains non-executable branches in the ELTS(t) and detecting all non-executable branches if such branches exist. []

For a given ELTS(t), let RG(t) denote a tree obtained from the ELTS(t) by deleting all non-executable branches and their descendants. Here, we call the RG(t) the "reachability graph for t". Let us consider the following LOTOS expression t_3 = <S>.

t_3 = <S'>

S' := f ? x : int [-2≤x≤2] ; (([x≥0] -> (**i** ; g ! x ; **stop**)

[] ([x≤0] -> g ! x ; q ! z [z=-x] ; **stop**))

The reachability graph RG(t_2) for the LOTOS expression t_2 in Example 3.1 is the same as the ELTS(t_3).

[Definition 3.2 (Test suite)]

For any leaf node s_n in the reachability graph RG(t) for a given LOTOS expression "t", if a set of test cases TS(t) contains a test case to trace the path from the root node to the node s_n, then the set TS(t) is called a "test suite for t". []

[Problem 3 (The test suite derivation problem)]

The test suite derivation problem is the problem for deriving a test suite for a given LOTOS expression "t". []

In general, a LOTOS expression may contain deadlocks. For example, for the LOTOS expression t_1 = <R,D(w)> in Example 2.1, suppose that an event sequence "f!0 ; g!0 ; h!1" is executed for the behaviour expression R. This means that a test case to trace the path from the root node m_1 to the node m_4 in Fig. 1 is executed. Since the integers "0" and "1" are assigned to the variables "x" and "y" in the process R, respectively, the values of the two predicates [y≥0 and x=y] ($Cond(m_4 \rightarrow m_5)$) and [y≤-1] ($Cond(m_4 \rightarrow m_6)$) are both false. Therefore, both "k!x" and "k!y" cannot be executed, which means that the system enters a deadlock state. For such a case, we say that the LOTOS expression t_1 = <R,D(w)> contains a deadlock. In this paper, an event sequence such as "f!0 ; g!0 ; h!1" is called a test case for leading to a deadlock state.

[Problem 4 (The deadlock detection problem)]

The deadlock detection problem is the problem for deciding whether a given LOTOS expression "t" contains deadlocks and generating test cases for leading to the deadlock states if they exist. []

LOTOS expressions may describe non-deterministic behaviors. For the above t_2 = <S>, if an event sequence "f!0 ; g!0" is executed for the behaviour expression S, then there are two possibilities. If "f!0 ; g!0" represents a test case to trace the path from the node s_1 to the node s_5 in Fig. 2, then there is no executable event after the interaction at gate g in the node s_5. If it represents a test case to trace the path from the node s_1 to the node s_6, then "q!0" is executable after the

interaction at gate g in the node s_6. In this case, we say that the LOTOS expression describes non-determinism, and "f!0 ; g!0" is called a test case representing non-deterministic behaviors.

[Problem 5 (Non-determinism detection problem)]

The non-determinism detection problem is the problem for deciding whether a given LOTOS expression "t" describes non-determinism and generating test cases representing non-deterministic behaviors if they exist. []

4. Basic idea for automatic analysis and test case derivation

It would be desirable that the test cases described in the above Problems 1 to 5 could be derived algorithmically for any LOTOS expression, however, this is impossible in general. For example, consider the following LOTOS expression t_4 = <Q> which is the specification of Fermat's last conjecture [Tret 89].

[Example 4.1]

t_4 = <Q>

Q := f ? x ? y ? z ? n [x≥1 and y≥1 and z≥1 and n>2 and $x^n+y^n=z^n$] ; **stop** []

No algorithm is known to determine whether this condition is satisfiable. This shows that we cannot derive algorithmically test cases for full LOTOS expressions in general. Hereafter, we will give a restriction on LOTOS specifications which will ensure that test cases can be derived algorithmically. In this paper, we will consider P-LOTOS expressions. In this section, we will give the basic idea for automatic analysis and test case derivation.

4.1 Automatic test case derivation

In this subsection, we will describe an algorithm to derive a test case which traces a specific path in the corresponding ELTS(t) of a LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>" written as a P-LOTOS expression. We must give concrete values for the input/output parameters in order to derive such test cases. At first, we will explain how to calculate the condition which is necessary for taking the path from the root node to a given node. Next, we will show that it is decidable whether the

condition is satisfiable or not. If it is satisfiable, we will give the concrete values for the variables through integer linear programming. If not, we conclude that the path is non-executable.

In Section 3, we defined the reachability condition $\Psi s_n(x_1,...,x_k)$ from the root node to a given node s_n. For the ELTS(t_1) in Fig. 1, for instance, we have

Ψm_1 = true

$\Psi m_2(x)$ = $(-8 \le x \le 8)$

$\Psi m_4(x,y)$ = $(-8 \le x \le 8)$ and $(-8 \le y \le 8)$

$\Psi m_5(x,y)$ = $(-8 \le x \le 8)$ and $(-8 \le y \le 8)$ and $(y \ge 0)$ and $(x=y)$

Note that the path from the root node s_1 in an ELTS(t) to a node s_n is non-executable if and only if $\Psi s_n(x_1,..,x_k)$ is unsatisfiable, i.e., "not($\exists\ x_1,...,x_k\ [\Psi s_n(x_1,...,x_k)]$)".

[Lemma 4.1]

For any node s in a given ELTS(t), the reachability condition $\Psi s(x_1,...,x_k)$ is a P-sentence if "t" is a P-LOTOS expression. []

[Lemma 4.2]

For any P-LOTOS expression "t" and a node s in the corresponding ELTS(t), it is decidable whether the reachability condition $\Psi s(x_1,...,x_k)$ is satisfiable, i.e., whether the following predicate holds.

$\exists\ x_1,...,x_k\ [\Psi s(x_1,...,x_k)]$

(proof) From Lemma 4.1, $\Psi s(x_1,...,x_k)$ is a P-sentence. Therefore, $\exists\ x_1,..,x_k\ [\Psi s(x_1,..,x_k)]$ is a Presburger sentence [HoUl 79]. It is decidable whether a given Presburger sentence is true or not [HoUl 79]. []

One way to decide this problem is through integer linear programming. By replacing "not($E_1 \ge E_2$)" by "($E_1 < E_2$)", we can transform a given reachability condition $\Psi s(x_1,...,x_k)$ into an equivalent P-sentence $\Psi' s(x_1,...,x_k)$ which does not contain "not". Let

$\Psi^1 s(x_1,...,x_k)$ or $\Psi^2 s(x_1,...,x_k)$ or ... or $\Psi^m s(x_1,...,x_k)$

be a disjunctive normal form of $\Psi' s(x_1,...,x_k)$ where each $\Psi^q s(x_1,...,x_k)$ $(1 \le q \le m)$ is a conjunction of some linear inequalities. So, "$\exists\ x_1,...,x_k\ [\Psi s(x_1,...,x_k)]$" is true if and only if, for some q $(1 \le q \le m)$, $\Psi^q s(x_1,...,x_k)$ is satisfiable. By regarding each linear inequality in $\Psi^q s(x_1,...,x_k)$ as a constraint on an integer linear programming problem, we can decide whether the integer linear programming problem has integer solutions.

[Theorem 4.1]

For any P-LOTOS expression "t", the test case derivation problem (Problem 1) described in Section 3 can be solved algorithmically.

(Proof) For each node *s* in the corresponding ELTS(t), we can decide whether $\Psi s(x_1,...,x_k)$ is satisfiable. If $\Psi s(x_1,...,x_k)$ is satisfiable, then we can also give a solution $<X_1,...,X_k>$ such that $\Psi s(X_1,...,X_k)$ is true. Therefore, we can derive algorithmically a test case which traces the path from the root node in the ELTS(t) to the node *s* if such a test case exists. []

For example, the reachability condition $\Psi m_5(x,y)$ for the node m_5 is

$$\Psi m_5(x,y) = (-8 \le x \le 8) \text{ and } (-8 \le y \le 8) \text{ and } (y \ge 0) \text{ and } (x = y)$$

which is a P-sentence. We find that "$\exists x,y[\Psi m_5(x,y)]$" holds. One such solution is <8,8> since $\Psi m_5(8,8)$ is true. Therefore, "T_P := f!8 ; g!8 ; h!8 ; k!8" is a test case which traces the path from the root node m_1 to the node m_5 of the ELTS(t_1) in Fig. 1.

4.2 Detecting non-executable branches

We can find all non-executable branches in the corresponding ELTS(t) of a given P-LOTOS expression "t" as follows. A branch "$s \rightarrow u$" in the corresponding ELTS(t) is non-executable if the following two predicates hold.

(1) $\exists\ x_1,...,x_k\ [\Psi s(x_1,...,x_k)]$

(2) $\text{not}(\ \exists\ x_1,...,x_k\ [\Psi u(x_1,...,x_k)]\ )$

[Lemma 4.3]

It is decidable whether a given branch in the ELTS(t) is non-executable.

(proof) Obvious. []

For example, for the node m_{10} and m_{12} in Fig. 1, we can show that

(1) $\exists$ x,y,z,z1 [Ψm_{10}(x,y,z,z1)]

= $\exists$ x,y [(-8≤x≤8) and (-8≤y≤8) and (y≤-1) and (x-1≥7)]

= true

(2) $\exists$ x,y,z,z1 [Ψm_{12}(x,y,z,z1)]

= $\exists$ x,y [(-8≤x≤8) and (-8≤y≤8) and (y≤-1) and (x-1≥7) and (x-2≥7)]

= false.

Therefore, we conclude that m_{10} is reachable, but the branch "$m_{10}\rightarrow m_{12}$" is non-executable. It means that t_1 is finite and that it cannot invoke sub-process D more than two times.

4.3 Test cases for checking the presence of specified deadlocks

For a LOTOS expression "t", let *s* be a node of the corresponding ELTS(t) and let u_1, u_2, ... and u_p be all children of the node *s*. We assume that the relations *label*(*s*)-<a_1,c_1>->*label*(u_1),.., *label*(*s*)-<a_p,c_p>->*label*(u_p) hold. Then, we can conclude that a P-LOTOS expression "t" contains a (possible) deadlock if and only if there exists a node "*s*" in the ELTS(t) such that the following predicate is true.

$\exists$ x_1,...,x_k [Ψs(x_1,...,x_k) and not(c_1) and ... and not(c_p)]

Because, if the above predicate is true, then there is no executable event at the node *s*, which means that node *s* is a deadlock state. The above predicate is called the predicate for checking the presence of a deadlock at the node "*s*".

For the node m_4 of the ELTS(t_1) in Fig. 1, for example, we can show that the predicate for checking the presence of a deadlock is

$\exists$ x,y [Ψm_4(x,y) and not(*Cond*($m_4\rightarrow m_5$)) and not(*Cond*($m_4\rightarrow m_6$))]

= $\exists$ x,y [(-8≤x≤8)) and (-8≤y≤8) and not((y≥0) and (x=y)) and not(y≤-1)]

This can be satisfied, for instance by <x,y>=<0,1>. Therefore, the node m_4 is a deadlock state and "f!0 ; g!0 ; h!1" is a test case leading to the deadlock state.

[Lemma 4.4]

For a given node *s* in the ELTS(t) of a P-LOTOS expression "t", it is decidable whether the node *s* is a deadlock state. If *s* is a deadlock state, then a test case leading to the deadlock state can be derived algorithmically. []

4.4 Detecting non-determinism

Next, we will give a procedure for deciding whether a given P-LOTOS expression contains non-determinism. Let "t" be a P-LOTOS expression and let *s* denote a node in the corresponding ELTS(t). Suppose that there are two different paths "$s \xrightarrow{*} u$" and "$s \xrightarrow{*} v$" satisfying the following conditions (1) and (2), respectively.

(1) *label*(*s*)-<**i**,c_1>->*label*(u_1)-<**i**,c_2>->*label*(u_2)...-<**i**,c_n>->*label*(u_n)-<a_1,c_{n+1}>->*label*(*u*)
holds and the event "a_1" is "g\$$e_1$". Here, "g" is a gate name and "$" denotes either the input symbol "?" or the output symbol "!". If n=0, then the above sequence represents "*label*(*s*)-<a_1,c_1>->*label*(*u*)".

(2) *label*(*s*)-<**i**,d_1>->*label*(v_1)-<**i**,d_2>->*label*(v_2)...-<**i**,d_m>->*label*(v_m)-<a_2,d_{m+1}>->*label*(*v*)
holds and the event "a_2" is "g\$$e_2$". Here, the gate name "g" is the same as that of the event "a_1" in the condition (1). If m=0, then the above sequence represents "*label*(*s*)-<a_2,d_1>->*label*(*v*)".

We also assume that the following predicate

$$\exists\, x_1,...,x_k\ [\Psi s(x_1,...,x_k) \text{ and } \{c_1 \text{ and } ... \text{ and } c_{n+1}\} \text{ and } \{d_1 \text{ and } ... \text{ and } d_{m+1}\} \text{ and } (e_1=e_2)\ ]$$

is true and integers $n_1,...,n_k$ are their solutions. Since $\Psi s(n_1,...,n_k)$ is true, the event sequence $\alpha_{[s1,s]}(x_1/n_1,...,x_k/n_k)$ is a test case to trace the path from the root node s_1 of the ELTS(t) to the node *s*.

If the above predicate holds, then "$\alpha_{[s1,s]}(x_1/n_1,...,x_k/n_k)$; g!$e_1(x_1/n_1,...,x_k/n_k)$" is a test case to trace the path from the root node s_1 to the node "*u*". Here, g!$e_1(x_1/n_1,...,x_k/n_k)$ represents the

output event obtained by substituting the values $n_1,...,n_k$ into the variables $x_1,...,x_k$ of the expression e_1, respectively. Since the values $n_1,...,n_k$ also satisfy the conditions $d_1,...,d_{m+1}$, "$\alpha_{[s1,s]}(x_1/n_1,...,x_k/n_k)$; $g!e_2(x_1/n_1,...,x_k/n_k)$" is a test case to trace the path from the root node s_1 to the node "v". The value of the expression "$e_1(x_1/n_1,...,x_k/n_k)$" is the same as that of "$e_2(x_1/n_1,...,x_k/n_k)$". It means that "t" contains non-determinism and that the sequence "$\alpha_{[s1,s]}(x_1/n_1,...,x_k/n_k)$; $g!e_1(x_1/n_1,...,x_k/n_k)$" is a test case representing non-deterministic behaviours. Hereafter, the node "s" is called the node starting non-deterministic behaviours. Since it is decidable whether the above predicate "$\exists\, x_1,...,x_k$ [$\Psi s(x_1,...,x_k)$ and {...} and {...} and (e_1=e_2)]" is true, the following lemma holds.

[Lemma 4.5]

For a given node "s" in the ELTS(t) of a P-LOTOS expression "t", it is decidable whether "s" is the node starting non-deterministic behaviours. If "s" is such a node, then a test case representing non-deterministic behaviours is also derived algorithmically. []

For the ELTS(t_2) of P-LOTOS expression t_2 = <S> in Fig. 2, for instance, there are two paths $s_2 \rightarrow s_4 \rightarrow s_5$ and $s_2 \rightarrow s_6$. We can show that $Event(s_2 \rightarrow s_4)$="**i**", $Event(s_4 \rightarrow s_5)$="g!x" and $Event(s_2 \rightarrow s_6)$="g!x", and that

$\exists$ x [Ψs_2(x) and $Cond(s_2 \rightarrow s_4)$ and $Cond(s_4 \rightarrow s_5)$ and $Cond(s_2 \rightarrow s_6)$ and (x=x)]

= $\exists$ x [($-2 \le x \le 2$) and ($x \ge 0$) and true and ($x \le 0$) and (x=x)]

which can be satisfied by "x=0". Therefore, we conclude that the P-LOTOS expression "t_2" is non-deterministic.

5. Automatic analysis and test selection for finite P-LOTOS expressions

5.1 Algorithms for finite P-LOTOS expressions

[Theorem 5.1]

If a given P-LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>" is finite, then the following analysis problems, which are described in Section 3 as Problems 2, 4 and 5, can be solved algorithmically.

(1) The non-executable branch detection problem (Problem 2)

(2) The deadlock detection problem (Problem 4)

(3) Non-determinism detection problem (Problem 5)

(proof) Let RG(t) be the reachability graph for "t". That is, RG(t) is a tree obtained from the ELTS(t) by deleting all non-executable branches and their descendants. In general, the ELTS(t) may be infinite even if "t" is finite. However, by definition, RG(t) is finite if and only if "t" is finite. For instance, although the P-LOTOS expression "t_1" in Fig. 1 is finite (see Section 4.2), the ELTS(t_1) in Fig. 1 is an infinite tree. Since the branch "$m_{10} \rightarrow m_{12}$" in Fig.1 is non-executable, the corresponding RG(t_1) is a finite tree, which is obtained by deleting the branch "$m_{10} \rightarrow m_{12}$" and its descendants from the ELTS(t_1). By Lemma 4.3, it is decidable whether a given branch in the ELTS(t) is non-executable. By checking whether each branch in the ELTS(t) is non-executable inductively from the root node, we can construct the corresponding finite RG(t) if "t" is finite. Then, Problem 2 can be solved algorithmically. By Lemmas 4.4 and 4.5, Problems 4 and 5 can also be solved algorithmically. []

Since Theorem 5.1 holds, we can decide whether a given finite P-LOTOS expression "t" does not contain non-executable branches, deadlocks nor non-determinism.

[Theorem 5.2]

For any finite P-LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>", we can derive a set of test cases (a test suite) which covers all paths in the RG(t) algorithmically. That is, Problem 3 in Section 3 can be solved algorithmically.

(proof) We can assume that the RG(t) is finite if "t" is finite. By Theorem 4.1, we can derive the set of all test cases which trace the paths from the root node of the RG(t) to all leaf nodes. Therefore, Problem 3 can be solved algorithmically. []

Since we show that the P-LOTOS expression "t_1" in Fig. 1 is finite and the branch "$m_{10} \rightarrow m_{12}$" is non-executable, the following set $TS_1(t_1)$ of test cases is a test suite for "t_1".

$TS_1(t_1)$ = { f!0 ; g!0 ; h!0 ; k!0, f!0 ; g!0 ; h!-1 ; k!-1 ; a!0 ; b!0 ,
f!8 ; g!8 ; h!-1 ; k!-1 ; a!0 ; c!0 ; a!1 ; b!1 }

5.2 Discussion

For the purpose of test suite development, we assume that the objective is the coverage of all the branches in the specification. Therefore solutions to all the three problems in Theorem 5.1 are

required for the test suite development. It is clear that non-executable branches cannot be tested. Given the detection of non-executable branches, as described in Section 4.2, we can assume that all branches are executable.

The detection of deadlocks is also important; in fact, a testing framework has been proposed [Lang 89] in which the presence of deadlocks in the tested implementation can be detected, possibly through waiting until a timeout occurs. In the case that the specification contains a deadlock for certain input parameter values, it could be suggested to generate tests with input values that lead to the deadlock (to test that this case is correctly implemented), and other tests with input values that avoid the deadlock (to test that the remaining part of the specified branch is correctly implemented).

The detection of non-determinism is important since the testing process becomes much more difficult (see for instance [Brin 89b, Fuji 91b]). The implementation may either implement one possibility only, or include both possibilities. In the latter case, special precautions must be taken to assure that all branches of the implementation are sufficiently tested. The specification of a test case must also foresee all possible responses from the implementation, even if the purpose of the test case is the testing of one of the branches only [Weze 89].

We conclude that the problem of test case selection for a specification written in the form of a P-LOTOS expression is solved by the algorithm described above. Each element (event sequence) of the resulting test suite represents a test case. The execution of all these test cases leads to the coverage of all branches of the given specification. In the case that the specification contains the possibility of deadlock for certain input parameter values, additional tests for testing these deadlock cases should be foreseen. In the case of non-deterministic choices in the specification, the specification of the test cases must be enhanced for taking into account the different branches that the implementation may choose.

6. Treatment of infinite P-LOTOS expressions

6.1 Termination of processes

It has been shown that a Basic LOTOS expression can simulate a Turing machine [FaGn 90]. Since P-LOTOS is an extension of Basic LOTOS, it is undecidable whether a given P-LOTOS

expression "t=<P_0,P_1(..),..,P_k(..)>" will terminate eventually, i.e., whether "t" is finite. In general, LOTOS expressions can be treated as term rewriting systems (TRS's). Several sufficient conditions to guarantee the termination of TRS's have been proposed [Klop 87]. These sufficient conditions can be used to guarantee the termination of P-LOTOS expressions. If "t" is infinite, we cannot derive a test suite, in general, because there may exist infinite leaves. In the next sub-section, we will give a solution for this problem. We can also show that it is undecidable whether the main process P_0 will eventually invoke a process P_j. Therefore, even if the process P_j contains a deadlock, it is undecidable whether the main process P_0 can reach the deadlock state. That is, in general, it is undecidable whether a P-LOTOS expression "t" contains deadlocks if "t" is infinite. In Section 6.3 and 6.4, we will explain how to treat this problem.

6.2 Test suite derivation for infinite systems

In general, if a given P-LOTOS expression "t" is an infinite system, then the number of test cases may be infinite. A solution for the purpose of practical conformance testing is to reduce the maximum number of executed events or process invocations. In this paper, we will limit the number of executed events to a maximum "M". Let $ELTS_M(t)$ denote the sub-graph of the ELTS(t) which is obtained from the ELTS(t) by deleting all branches whose distances from the root node are greater than "M". And let $RG_M(t)$ denote the sub-graph of the $ELTS_M(t)$ which is obtained from the $ELTS_M(t)$ by deleting all non-executable branches and their descendants. If a set $TS_M(t)$ of test cases whose lengths are less than or equal to "M" satisfies the following condition (1), then the set $TS_M(t)$ is called an M-test suite for "t".

(1) For any leaf node s_n in the $RG_M(t)$, $TS_M(t)$ contains a test case to trace the path from the root node in the $RG_M(t)$ to the node s_n.

[Theorem 6.1]

For a given P-LOTOS expression "t" and a given positive integer M, an M-test suite for "t" can be derived algorithmically.

(proof) Obvious. []

[Example 6.1]

t_5 = <R,D>

R:= f?x:int[-8≤x≤8] ; g!x ; h?y:int[-8≤y≤8] ;

(([y≥0] -> k!x ; **stop**) [] ([y≤-1] -> k!y ; D))

D:= a?z:int[z≥0] ; (b!z[z=0] ; **stop** [] (c!z ; D)) []

The P-LOTOS expression t_5 = <R,D> in Example 6.1 is an infinite system. By using the above technique, we can derive an M-test suite of "t_5" for any positive integer M. For instance, the following $TS_4(t_5)$ and $TS_6(t_5)$ are a 4-test suite and a 6-test suite of "t_5", respectively.

$TS_4(t_5)$ = { f!0 ; g!0 ; h!0 ; k!0, f!0 ; g!0 ; h!-1 ; k!-1 }

$TS_6(t_5)$ = { f!0 ; g!0 ; h!0 ; k!0, f!0 ; g!0 ; h!-1 ; k!-1 ; a!0 ; b!0,

f!8 ; g!8 ; h!-1 ; k!-1 ; a!1 ; c!1 }

6.3 Regular P-LOTOS expressions

In this sub-section, we will introduce a sub-class of P-LOTOS, called "regular P-LOTOS" such that we can detect all non-executable branches, deadlocks and non-determinism algorithmically. If a given P-LOTOS expression "t" contains only the action prefix operators ";" and choice operators "[]" as the operators, then "t" is called a "regular P-LOTOS expression". The control flow of regular P-LOTOS expressions is restricted to the equivalent of FSM's. That is, a regular P-LOTOS expression corresponds to a specification model of extended finite state machines (EFSM's) where variables are of type boolean or integer, and where the integer operations are restricted to addition, subtraction and comparison. More particularly, if a regular P-LOTOS expression "t" does not contain processes with process parameters, then "t" is called a "regular P-LOTOS expression without process parameters". The P-LOTOS expression t_5 = <R,D> in Example 6.1 is a regular P-LOTOS expression without process parameters.

For each process P_h in a regular P-LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>", let $[t]_{Ph}$ denote the regular P-LOTOS expression $[t]_{Ph}$=<P_h(..)>. Since "t" is a regular P-LOTOS expression, the corresponding ELTS($[t]_{Ph}$) (1≤h≤k) are all finite trees and the labels of all leaf nodes in each ELTS($[t]_{Ph}$) are either "**stop**" or process names. For example, Fig. 3 (a) and Fig. 3 (b) are the corresponding ELTS($[t_5]_R$) and ELTS($[t_5]_D$) for the regular P-LOTOS expression t_5 = <R,D> in

Example 6.1. For the ELTS($[t_5]_R$) and ELTS($[t_5]_D$), we can also define the test cases and the predicates for checking the presence of a deadlock, which are defined in Section 4. For example, "f!0 ; g!0 ; h!-1 ; k!-1" is a test case to trace the path from the root node "m_1" of the ELTS($[t_5]_R$) to the node "m_6". It corresponds to the case where the process R invokes the process D.

In general, the ELTS(t) for a regular P-LOTOS expression "t=<P_0,P_1(..),..,P_k(..)>" can be constructed by concatenating the corresponding ELTS($[t]_{Ph}$) ($1 \le h \le k$). For example, for the regular P-LOTOS expression t_5 = <R,D> in Example 6.1, the ELTS(t_5) can be constructed as follows. First, suppose that "ω" represents the ELTS($[t_5]_R$). The label of the leaf node "m_6" of the tree "ω" is "D". Then, we append the ELTS($[t_5]_D$) to the node "m_6" of the tree "ω". Since the label of the leaf node "p_4" of the new tree "ω" is "D", we append the ELTS($[t_5]_D$) to the node "p_4" of the tree "ω". The derived tree "ω" by repeating this process corresponds to the ELTS(t_5). If "t" contains processes with process parameters, then the formal process parameters in the appended ELTS($[t]_{Ph}$) must be replaced by the actual process parameters. Note that the ELTS(t) can be constructed only by appending the corresponding ELTS($[t]_{Ph}$) ($1 \le h \le k$) without replacing process parameters if "t" does not contain processes with process parameters.

[Theorem 6.2]

For a regular P-LOTOS expression "t=<P_0,P_1,..,P_k>" without process parameters, the three analysis problems (Problems 2, 4 and 5) described in Theorem 5.1 can be solved algorithmically even if "t" is infinite.

(proof) As we described above, the ELTS(t) can be constructed simply concatenating the corresponding ELTS($[t]_{Ph}$) ($1 \le h \le k$) and the label of each node of the ELTS($[t]_{Ph}$) is either "**stop**" or a process name. Therefore, we can determine what processes can be invoked directly from a given process "P_h". Then, we can also determine whether the main process P_0 of "t=<P_0,P_1,..,P_k>" can invoke each process "P_h" and derive a test case to invoke the process P_h from the main process P_0 algorithmically. If the main process P_0 can invoke a process P_h and there exists a deadlock state in the corresponding ELTS($[t]_{Ph}$), then "t" contains a deadlock. Otherwise, "t" does not contain a deadlock. Then, the deadlock detection problem (Problem 4) can be solved algorithmically for any regular P-LOTOS expression without process parameters. By using similar techniques, Problems 2 and 5 can also be solved algorithmically. []

6.4 Sufficient conditions for proving that a specification has no deadlocks

Next, we will consider regular P-LOTOS expressions with process parameters. In general, it is undecidable whether a regular P-LOTOS expression with process parameters contains deadlocks because we can derive a regular P-LOTOS expression which simulates a given Turing machine. Although the deadlock detection problem is undecidable in general, we can give a sufficient condition to guarantee that a given regular P-LOTOS expression with process parameters contains no deadlocks.

[Example 6.2]

t_6 = <P,R(w)>

P := f?x:int [-8≤x≤8] ; (([x<0] -> g!x ; **stop**) [] ([x≥0] -> h!x ; R(x-1)))

R(w:int) := (([w≥-1] -> p!w ; **stop**) [] ([w≥0] -> q!w ; R(w-1))) []

Consider Example 6.2. We can derive the ELTS($[t_6]_P$) and ELTS($[t_6]_R$) in Fig. 4 from t_6 = <P,R(w)>. Let us apply the technique described in Section 6.3 for proving that t_6 has no deadlocks. Then, we find a deadlock occurs if R(-2) is invoked by checking the ELTS($[t_6]_R$). However, we can easily find that "t_6" contains no deadlocks because R(w) is invoked only for an integer "w" such that "w≥-1" holds. Therefore, we will use the following technique.

At first, we specify the range conditions for the processes P and R(w) as follows :

$Range_P$:= true

$Range_{R(w)}$:= (w≥-1)

These range conditions must be described as P-sentences. "$Range_P$:= true" means that there is no assumption as the range condition. "$Range_{R(w)}$:= (w≥-1)" means that the range of "w" must be greater than or equal to "-1". We prove that these range conditions hold as for each process invocation.

For the ELTS($[t_6]_P$) in Fig. 4 (a), we have "Ψp_4(x) = (-8≤x≤8) and (x≥0)". Then, the following predicate holds.

$$\forall\ x\ [\ \text{Range}_P \text{ and } \Psi p_4(x) \supset \text{Range}_{R(x-1)}\ ]$$

This means that the range condition for R(x-1) is satisfied when P is invoked and the events "f?x:int" and "h!x" are executed. Similarly, for the nodes r_1 and r_3 in the ELTS($[t_6]_R$), the following predicate holds.

$$\forall\ w\ [\ \text{Range}_{R(w)} \text{ and } \Psi r_3(w) \supset \text{Range}_{R(w-1)}\ ]$$

Since the above two predicates hold, the two range conditions Range_P and $\text{Range}_{R(w)}$ hold as the assertions when the processes P and R are invoked recursively.

For any node "s" in the ELTS($[t_6]_P$) in Fig. 4 (a), the following predicate holds (here, u_1, u_2, .. and u_p denote the children nodes of the node s).

$$\text{not}(\ \exists\ x\ [\ \text{Range}_P \text{ and } \Psi s(x) \text{ and not}(\mathit{Cond}(s \rightarrow u_1)) \text{ and } \ldots \text{ and not}(\mathit{Cond}(s \rightarrow u_p))\ ]\ )$$

For any node "t" in the ELTS($[t_6]_R$) in Fig. 4 (b), the following predicate also holds (here, v_1, v_2, .. and v_q denote the children nodes of the node t).

$$\text{not}(\ \exists\ w\ [\ \text{Range}_{R(w)} \text{ and } \Psi t(w) \text{ and not}(\mathit{Cond}(t \rightarrow v_1)) \text{ and } \ldots \text{ and not}(\mathit{Cond}(t \rightarrow v_q))\ ]\ )$$

Therefore, we can conclude that "t_6" contains no deadlocks.

This technique is related to the program proof technique using invariant assertions. The range conditions play the role of invariants. If suitable range conditions have been selected for each process, their satisfaction can be shown algorithmically, as well as the implied deadlock. Similar techniques can also be applied to the non-executable branch detection problem and non-determinism detection problem for regular P-LOTOS expressions with process parameters.

7. Example

In this section, we will give an example of automatic analysis and test selection. A specification of a simplified Session protocol is described in Table 2 (a). The specification treats only the data transfer phase, not the connection establishment and release phases. It describes the four functional units, kernel, half-duplex and minor and major synchronization [ISO 87]. The specification is

described as an extended finite state machine (EFSM) model. Some enumeration types are treated as integer type. Three processes P713, P04A and P10A correspond to the states of this model. P713 corresponds to the data transfer state. P04A and P10A correspond to the "state waiting for Major-Sync-Ack SPDU" and the "state waiting for S-Sync-Major response", respectively [ISO 87]. There are four integer variables "Va", "Vm", "Vr" and "Vsc" which correspond to the state variables of this EFSM model. The variable "Va" holds the lowest serial number to which a synchronization point confirmation is expected. No confirmation is expected when "Va=Vm" holds. The variable "Vm" holds the next serial number to be used. The variable "Vr" holds the lowest serial number to which resynchronization restart is permitted. The value of "Vsc" is either 1 or 0. If the value of "Vsc" is 1 and the value of "Va" is less than that of "Vm", then the SS-user has the right to issue minor synchronization point responses. If the value of "Vsc" is 0, then the SS-user does not have the right to issue minor synchronization point responses. The events such as the transmission/reception of MIA, MIP, MAA and MAP messages correspond to the state transitions. We have proved that this specification does not have any deadlocks, non-executable branches, nor non-determinism, by using the technique described in Section 6.4. A generated 3-test suite is shown in Table 2 (b).

8. Test system for P-LOTOS expressions

We have implemented a test system for P-LOTOS expressions. The system has the following facilities : (1) to draw the $ELTS_M(t)$ graphically on a display as a tree for a given P-LOTOS expression "t" and a positive integer M, (2) to delete all non-executable branches and their descendant nodes in the $ELTS_M(t)$ and draw the M-reachability graph $RG_M(t)$, (3) to generate a set of test cases for the $RG_M(t)$ automatically based on the techniques described in Sections 4, 5 and 6, and (4) to check whether the implementation under test (IUT) satisfies a given specification by using the generated test cases. The test system has been developed on SUN SPARC workstations. Hereafter, we will explain the outline of these facilities.

In general, the size of the ELTS's of P-LOTOS expressions may become large. For displaying large trees, we have developed a graph editor VTM [MaNa 92]. VTM makes it easy to observe a whole tree and some specified sub-parts simultaneously. The users can give the input commands for

the tree by direct manipulation on the display. VTM is a library program using the X Window systems which can be used from any application program. Our test system uses VTM for displaying the extended labeled transition systems and reachability graphs. The users can enlarge and reduce the size of the graph arbitrarily on the display. It takes less than 0.2 seconds for VTM to display a tree which has 1000 nodes (SUN SPARCstation ELC).

For a given P-LOTOS expression "t" and depth "M", our test system constructs the $ELTS_M(t)$ which is a sub-graph of the ELTS(t) where all nodes in the ELTS(t) whose distances from the root node are greater than M are deleted. Then, our system draws the graph. For example, for the P-LOTOS expression "t_1" in Example 2.1, the $ELTS_{10}(t_1)$ and $RG_{10}(t_1)$ are drawn in Fig. 5 (a) and Fig. 5 (b), respectively. The corresponding $ELTS(t_1)$ is given in Fig. 1. In Fig. 5, all executable events for a node "N" are described as the labels of its descendant nodes and the conditions for executing their events are described as the labels of the branches from the node N. First, our test system reads the P-LOTOS specification "t_1" and depth "10" and draws $ELTS_{10}(t_1)$ in Fig. 5 (a). By clicking the button "*cut node*" on the display, all non-executable branches and their descendant nodes in the $ELTS_{10}(t_1)$ are deleted automatically. Then, $RG_{10}(t_1)$ in Fig. 5 (b) is obtained. This facility helps the designer to understand what kinds of event sequences are executable for a given P-LOTOS expression.

If the user clicks a node on the $ELTS_M(t)$ (or $RG_M(t)$), the test system generates automatically a test case to execute the event sequence on the path from the root node to the designated node. For example, if the node "k!y" in Fig. 5 (b) is clicked, then a test case "f!0; g!0; h!-1; k!-1" is generated and the values of the variables are shown on a small window. If the user clicks the button "*test_suite*", then an M-test suite for a given M-reachability graph is derived automatically. The M-test suite can also be derived without displaying the M-reachability graph when the depth M is large.

The test system also decides whether a given node is a deadlock state, and if the node is a deadlock state, then it derives a test case leading to the deadlock state. For example, Fig. 5 (c) represents that "f!0 ; g!0 ; h!1" is a test case leading to a deadlock state for the P-LOTOS expression "t_1" in Example 2.1.

We have also implemented a tester to test a given IUT using the derived test suite. The tester communicates with the IUT through the communication ports on UNIX. Suppose that a test case "T" is given. When the IUT executes an input event, say "a?x", the tester finds the value of the variable "x" in the test case "T" and sends it to the IUT through the communication port corresponding to the gate "a". When the IUT executes an output event, say "b!y", the tester receives the value of the expression "y" from the IUT through the communication port corresponding to the gate "b", and checks whether it is the same as the value of the expression "y" in the test case "T". If they are not the same, then the test fails. Therefore, the tester shows a warning message on the display. The tester also checks whether the value of the expression "y" is sent from the gate "b". If it is sent from another port, then the tester also generates a warning message. If all tests for "T" succeed, then we consider that the test for "T" is success.

In order to generate a test case, some integer linear programming problems must be solved. It takes about 10 and 20 seconds for our tester to solve the integer linear programming problems whose constraints' numbers are 10 and 30, respectively. For example, for the P-LOTOS expression "$t_{Session}$" of the OSI Session protocol in Section 7, it takes about 12 minutes to draw the $ELTS_3(t_{Session})$ which has 106 leaf nodes, delete all non-executable branches and their descendant nodes, draw the $RG_3(t_{Session})$ and generate the 3-test suite (28 test cases) in Table 2 using a SUN SPARCstation ELC (12MB Memory). It takes about 75 minutes to generate a 4-test suite.

9. Conclusion

In this paper, we consider specifications written in a restricted form of LOTOS, called P-LOTOS where variables are of type boolean or integer, and where the integer operations are restricted to addition, subtraction and comparison. We show that in this context, the problems of detection of deadlock, non-executable branches and non-deterministic choice, as well as the selection of a test suite, can be solved by using a decision procedure for integer linear programming. However, these problems become in general undecidable if specifications with infinite loops are considered. However, with invariant assertions provided by the user, some of these problems can be solved algorithmically in the case of regular P-LOTOS expressions. By using a similar technique, the

question of equivalence between two specifications (with process parameters) is solved algorithmically [HiNi 89].

For a given branch in a specification written in P-LOTOS, the algorithm described in this paper determines a sequence of input values which lead to the execution of the branch, if the branch is executable. Similarly, for a given alternative choice in the specification, it finds a sequence of input values which allow for both alternatives to be chosen by the specified system (non-determinism), if such a sequence exists. And for each transition with a guard (or alternatives with guards), it determines whether a sequence of input values exists which leads the execution of the system to this transition and the guard is false (or to the alternative and the guards of all alternatives are false), which is a deadlock. In the context of programming languages written in high-level programming languages, such questions can be resolved through symbolic execution [ChLe 73] or through the calculus of weakest preconditions [Dijk 75]. The algorithm proposed in this paper, which is applicable for P-LOTOS is more efficient than those methods. A tool for the analysis and test selection based on our techniques has been implemented.

In this paper, we have assumed a restricted specification language, called P-LOTOS. Although it seems that in many areas, most aspects to be specified can be described in this restricted framework, including for instance, sequence numbering in communications protocols, it would be desirable to extend the power of the specification language for which the here described methods for analysis and test suite development could be applied.

Acknowledgments

This work was partly supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at Université de Montréal.

References

[Boch 90] G. v. Bochmann : "Protocol specification for OSI", Computer Networks and ISDN Systems 18, pp.167-184, April 1990.

[Brin 88] E. Brinksma : "A Theory for the Derivation of Tests", Proc. 8th Int. Conf. Protocol Specification, Testing and Verification, pp.63-74, North-Holland, June 1988.

[Brin 89b] E. Brinksma, R. Alderden, R. Langerak, J. v. d. Lagemaat and J. Tretmans : "Formal approach to conformance testing", Proc. Int. Workshop on Protocol Test Systems, pp.311-325, North-Holland, Oct. 1989.

[CCITT 88] CCITT : "SDL : Specification and Description Language", Recommendation Z.100, Nov. 1988.

[ChLe 73] C. L. Chang and R. C. Lee : "Symbolic Logic and Mechanical Theorem Proving", Academic Press, 1973.

[Dijk 75] E. W. Dijkstra : "Guarded Commands, Nondeterminacy and Formal Derivation of Programs", Comm. ACM 18, 8, pp. 453-457, Aug. 1975.

[FaGn 90] A. Fantechi, S. Gnesi and G. Mazzarini : "How much Expressive are LOTOS Behaviour Expressions ?", Proc. 3rd Int. FORTE Conf., pp.17-32, North-Holland, Nov. 1990.

[Fuji 91a] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi : "Test Selection Based on Finite State Models", IEEE Trans. Soft. Eng., Vol. 17, No. 6, pp.591-603, June 1991.

[Fuji 91b] S. Fujiwara and G. v. Bochmann : "Testing Non-deterministic State Machines with Fault Coverage", Proc. 4th Int. Workshop on Protocol Test Systems, pp.267-280, North-Holland, Oct. 1991.

[HiBo 92] T. Higashino, G. v. Bochmann, X. Li, K. Yasumoto and K. Taniguchi : "Test System for a Restricted Class of LOTOS Expressions with Data Parameters", Proc. 5th Int. Workshop on Protocol Test Systems, North-Holland, Sept. 1992 (to appear).

[HiNi 89] T. Higashino, K. Ninomiya, T. Kimoto, K. Taniguchi and M. Mori : "Automated Verification of Equivalence of Protocol Machines", Proc. 9th Int. Conf. Protocol Specification, Testing and Verification, pp.235-246, North-Holland, June 1989.

[HoUl 79] J. E. Hopcroft and J. D. Ullman : "Introduction to Automata Theory, Languages, and Computation", Addison-Weslay, 1979.

[ISO 87] ISO : "Information Processing System - Open Systems Interconnection - Basic Connection Oriented Session Protocol Specification", IS 8327, Aug. 1987.

[ISO 89a] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807, Jan. 1989.

[ISO 89b] ISO : "Estelle : A Formal Description Technique Based on an Extended State Transition Model", ISO 9074, July 1989.

[Klop 87] J. W. Klop : "Term Rewriting Systems : A Tutorial", Bull EATCS, 32, pp.143-183, 1987

[Lang 89] R. Langerak : "A Testing Theory for LOTOS using Deadlock Detection", Proc. 9th Int. Conf. Protocol Specification, Testing and Verification, pp.87-98, North-Holland, June 1989.

[MaNa 92] T. Matsuura, T. Nakamura, T. Higashino, K. Taniguchi and S. Masuda : "VTM: A Graph Editor for Large Trees", Proc. 12th IFIP World Computer Congress'92, Vol. I, pp.210-216, Sept. 1992.

[Oppe 78] D. C. Oppen : "A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic", J. Comput. Syst. Sci., No. 16, pp.322-332, 1978.

[Pres 29] M. Presburger : "Uber die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt", in Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs, 1929.

[Sari 87] B. Sarikaya, G. v. Bochmann and E. Cerny : "A Test Design Methodology for Protocol Testing", IEEE Trans. on Soft. Eng., pp. 518-531, May 1987.

[TrSa 89] P. Tripathy and B. Sarikaya : "Test Generation from Protocol Specification", Proc. 2nd Int. FORTE Conf., pp.329-343, North-Holland, Nov. 1989.

[Tret 89] J. Tretmans : "Test Case Derivation from LOTOS Specifications", Proc. 2nd Int. FORTE Conf., pp.345-359, North-Holland, Nov. 1989.

[Weze 89] C. D. Wezeman : "The CO-OP Method for Compositional Derivation of Conformance Testers", Proc. 9th Int. Conf. Protocol Specification, Testing and Verification, pp.145-158, North-Holland, June 1989.

Figures & Tables

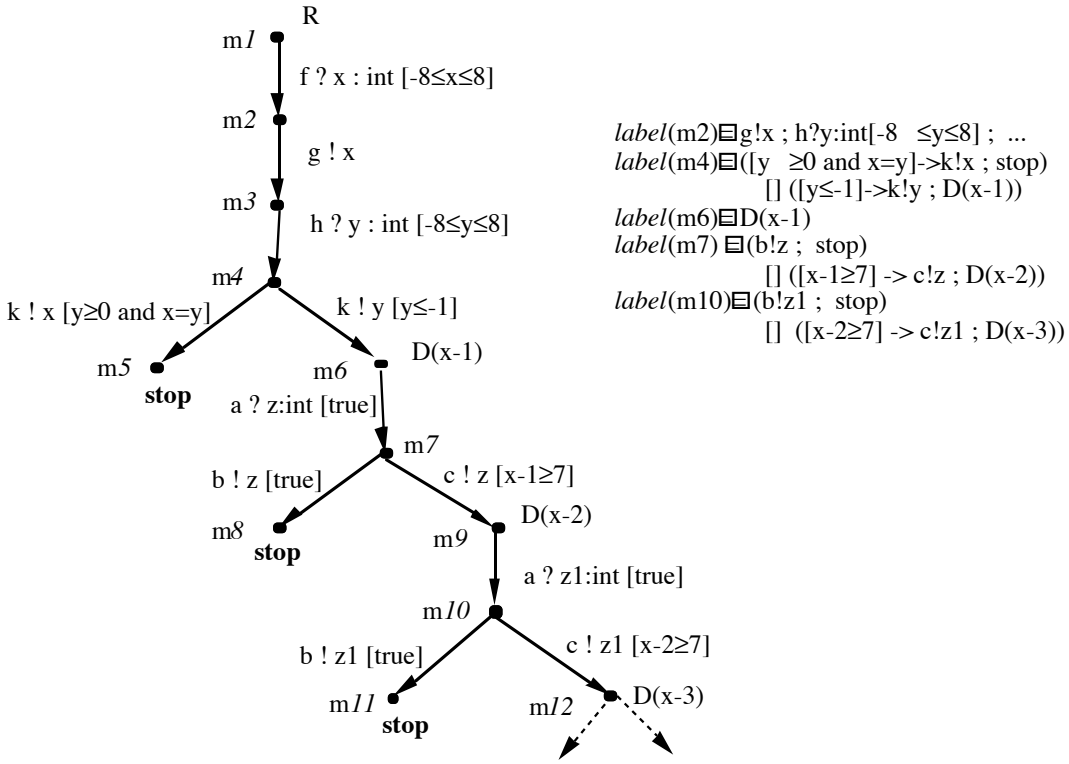


Fig. 1 An Extended Labeled Transition System ELTS(t_1) for LOTOS Expression t_1

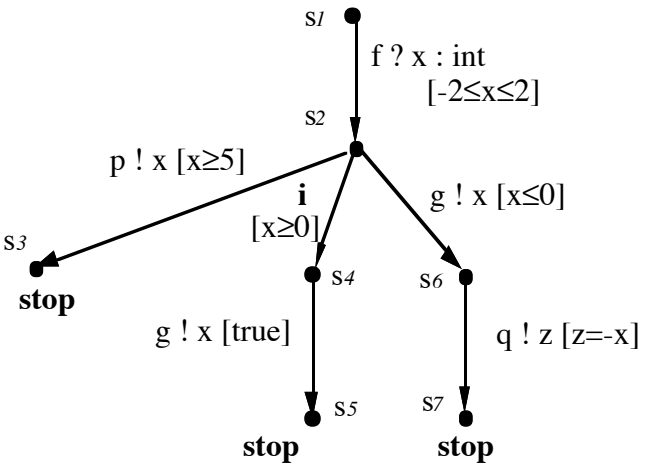
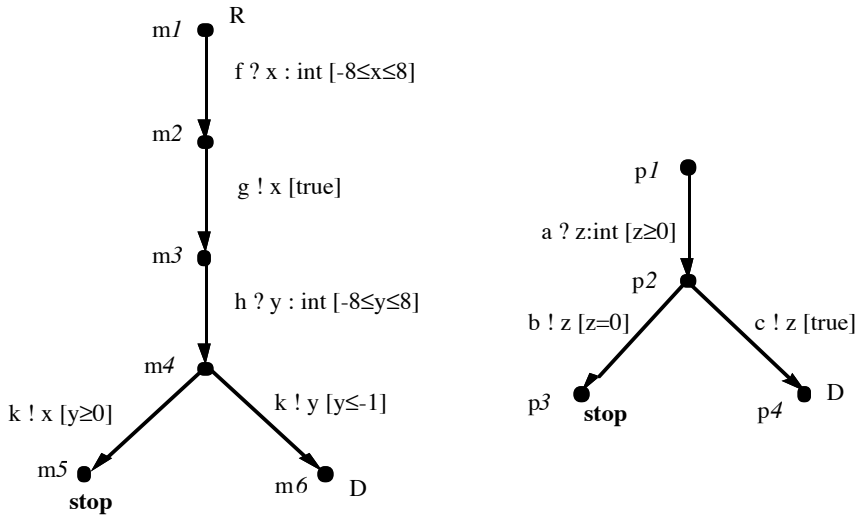
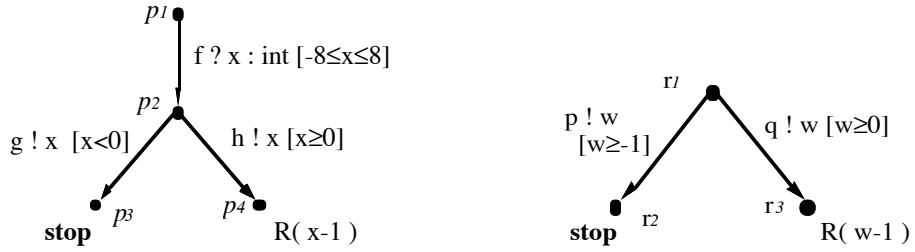


Fig. 2 An Extended Labeled Transition System ELTS(t_2) for LOTOS Expression t_2



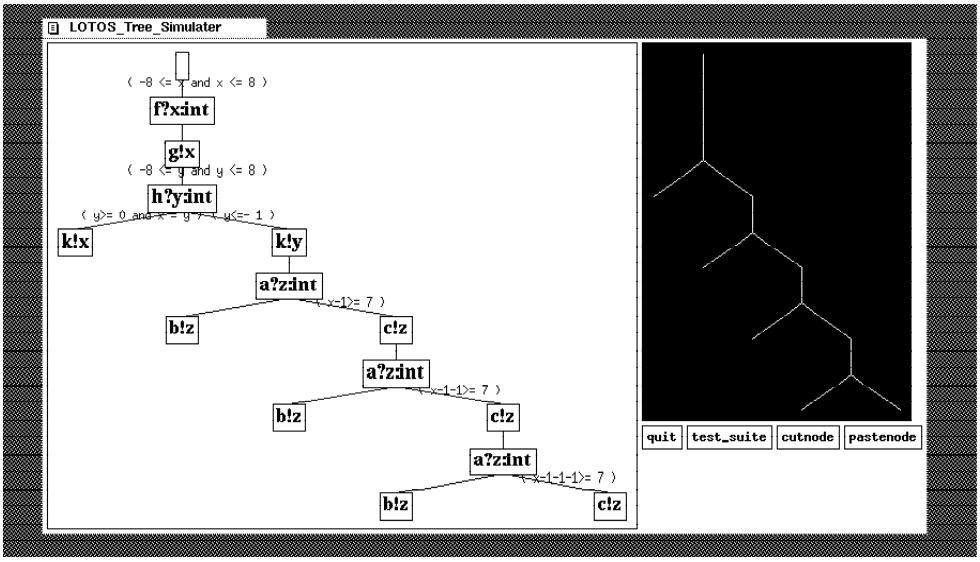
(a) ELTS($[t_5]_R$) (b) ELTS($[t_5]_D$)

Fig. 3 ELTS($[t_5]_R$) and ELTS($[t_5]_D$) for the regular P-LOTOS expression t_5 = <R,D>

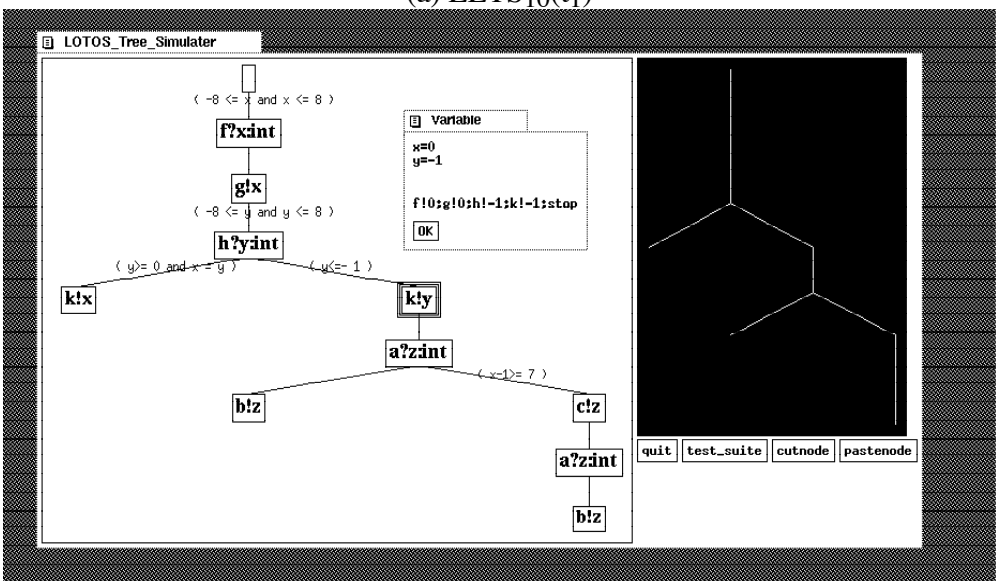


(a) ELTS($[t_6]_P$) (b) ELTS($[t_6]_R$)

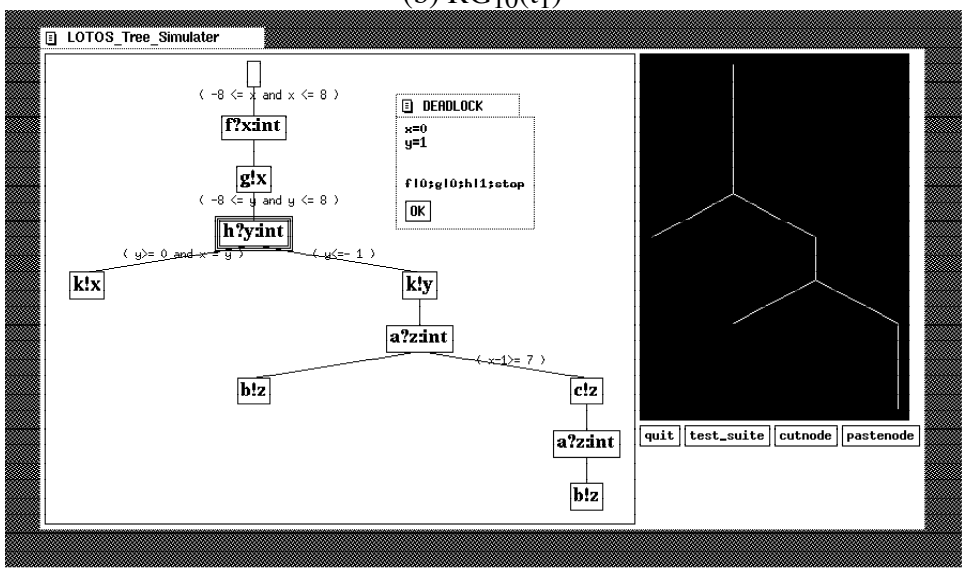
Fig. 4 ELTS($[t_6]_P$) and ELTS($[t_6]_R$) for the regular P-LOTOS expression t_6 = <P,R>



(a) $ELTS_{10}(t_1)$



(b) $RG_{10}(t_1)$



(c) Derivation of a test case leading to a deadlock state

Fig. 5 Test System for P-LOTOS Expressions

Table 1 Axioms and Inference Rules to Define the Relation B-<a,c>->B'

Axioms

• h ? x1:int ...! E1 ... ; B	-<h ? x1:int ...! E1..,true>->	B
• h ? x1:int ...! E1 ... [Q] ; B	-<h ? x1:int ...! E1..,Q>->	B
• **i** ; B	-<**i**,true>->	B
• **i** [Q] ; B	-<**i**,Q>->	B
• exit	-<exit,true>->	stop

Inference Rules

- $\dfrac{B \text{ -<a,c>-> } B'}{([Q] \text{ -> } B) \text{ -<a,c and Q>-> } B'}$

- $\dfrac{B \text{ -<a,c>-> } B'}{(B \text{ [] } B'') \text{ -<a,c>-> } B' \qquad (B'' \text{ [] } B) \text{ -<a,c>-> } B'}$

- $\dfrac{B \text{ -<a,c>-> } B' \quad \& \quad a \neq exit}{(B \text{ ||| } B'') \text{ -<a,c>-> } (B' \text{ ||| } B'') \qquad (B'' \text{ ||| } B) \text{ -<a,c>-> } (B'' \text{ ||| } B')}$

- $\dfrac{B1 \text{ -<exit,c1>-> } B1' \quad \& \quad B2 \text{ -<exit,c2>-> } B2'}{(B1 \text{ ||| } B2) \text{ -<exit,c1 and c2>-> } (B1' \text{ ||| } B2')}$

- $\dfrac{B \text{ -<a,c>-> } B' \quad \& \quad a \notin G \cup \{exit\}}{(B \text{ |[G]| } B'') \text{ -<a,c>-> } (B' \text{ |[G]| } B'') \qquad (B'' \text{ |[G]| } B) \text{ -<a,c>-> } (B'' \text{ |[G]| } B')}$

- $\dfrac{\begin{array}{l}B1 \text{ -<a,c1>-> } B1' \ \& \ a = g\$e_1 ... \$e_k \ \& \ B2 \text{ -<a',c2>-> } B2' \ \& \ a' = g\$e_1' ... \$e_k' \\ \& \ g \in G \cup \{exit\} \quad \text{(Here,"\$" denotes either the input symbol "?" or the output symbol "!".)}\end{array}}{(B1 \text{ |[G]| } B2) \text{ -<a,c1 and c2 and } (e_1 = e_1' \text{ and ... and } e_k = e_k') \text{>-> } (B1' \text{ |[G]| } B2')}$

- $\dfrac{B \text{ -<a,c>-> } B' \quad \& \quad a \neq exit}{(B \text{ >> } B'') \text{ -<a,c>-> } B' \text{>> } B''}$

- $\dfrac{B \text{ -<exit,c>-> } B'}{(B \text{ >> } B'') \text{ -<}\mathbf{i}\text{,c>-> } B''}$

- $\dfrac{B \text{ -<a,c>-> } B' \quad \& \quad a \neq exit}{(B \text{ [> } B'') \text{ -<a,c>-> } (B' \text{ [> } B'')}$

- $\dfrac{B \text{ -<exit,c>-> } B'}{(B \text{ [> } B'') \text{ -<exit,c>-> } B'}$

- $\dfrac{B'' \text{ -<a,c>-> } B'}{(B \text{ [> } B'') \text{ -<a,c>-> } B'}$

- $\dfrac{\begin{array}{l}P(x) \text{ is a process } \& \ P(x) := B(x) \ \& \ B(x) \text{-<a,c>->} B'(x) \ \& \ "\alpha" \text{ is an expression} \\ \& \ B''(x) \text{ is a behavior expression which is obtained by replacing all variables except "x" in } \\ \quad B'(x) \text{ by new variables}\end{array}}{P(\alpha) \text{ -<a,c>-> } B''(\alpha)}$

* For simplicity, the inference rules for "let", "hiding", "par", "generalized choice" and "accept" operators are omitted.

Table 2 A Regular P-LOTOS expression "$t_{Session}$" for Simplified OSI Session Protocol and a 3-Test Suite for "$t_{Session}$"

(a) Regular P-LOTOS expression

$t_{Session}$ =<Pinit,P713(Va,Vm,Vr,Vsc),P04A(Va,Vm,Vr,Vsc),P10A(Va,Vm,Vr,Vsc)>

```
Pinit := Init ? Va:int ? Vm:int [Va=Vm] ; P713(Va,Vm,0,0)

P713(Va,Vm,Vr,Vsc):=
        (rcvDT ; P713(Va,Vm,Vr,Vsc))
     [] (sndDT ; P713(Va,Vm,Vr,Vsc))
     [] ([Vsc=1] -> rcvMAP ? Sn:int  [Sn=Vm] ; P10A(Va,Vm+1,Vr,Vsc))
     [] ([Vsc=0] -> rcvMAP ? Sn:int  [Sn=Vm] ; P10A(Vm,Vm+1,Vr,Vsc))
     [] ([Vsc=1] -> sndMAP ; P04A(Vm,Vm+1,Vr,0))
     [] ([Vsc=0] -> sndMAP ; P04A(Va,Vm+1,Vr,0))
     [] ([Vsc=0] -> rcvMIA ? Sn:int  [(Vm>Sn)and(Sn>Va)] ; P713(Sn+1,Vm,Vr,Vsc))
     [] ([Vsc=1] -> rcvMIP ? Sn:int  [Sn=Vm] ; P713(Va,Vm+1,Vr,1))
     [] ([Vsc=0] -> rcvMIP ? Sn:int  [Sn=Vm] ; P713(Vm,Vm+1,Vr,1))
     [] ([Vsc=1] -> sndMIA ? Sn:int  [(Vm>Sn) and (Sn>Va)] ; P713(Sn+1,Vm,Vr,Vsc))
     [] ([Vsc=1] -> sndMIP ; P713(Vm,Vm+1,Vr,0))
     [] ([Vsc=0] -> sndMIP ; P713(Va,Vm+1,Vr,0))

P04A(Va,Vm,Vr,Vsc) :=
        (rcvDT ; P04A(Va,Vm,Vr,Vsc))
     [] (rcvMAA ? Sn:int  [Sn=Vm-1] ; P713(Vm,Vm,Vm,Vsc))
     [] ([Vsc=0] -> rcvMIA ? Sn:int [not(Sn=Vm-1)and(Vm>Sn) and (Sn>Va)] ;
                                                        P04A(Sn+1,Vm,Vr,Vsc))

P10A(Va,Vm,Vr,Vsc) :=
        (sndDT ; P10A(Va,Vm,Vr,Vsc))
     [] (sndMAA ? Sn:int  ; P713(Vm,Vm,Vm,Vsc))
```

(b) 3-test suite $TS_3(t_{Session})$ for the above regular P-LOTOS expression "$t_{Session}$"

$TS_3(t_{Session})$

```
= {  Init!0!0 ; rcvDT ; rcvDT,        Init!0!0 ; rcvDT ; sndDT,        Init!0!0 ; rcvDT ; rcvMAP!0,
     Init!0!0 ; rcvDT ; sndMAP,       Init!0!0 ; rcvDT ; rcvMIP!0,     Init!0!0 ; rcvDT ; sndMIP,
     Init!0!0 ; sndDT ; rcvDT,        Init!0!0 ; sndDT ; sndDT,        Init!0!0 ; sndDT ; rcvMAP!0,
     Init!0!0 ; sndDT ; sndMAP,       Init!0!0 ; sndDT ; rcvMIP!0,     Init!0!0 ; sndDT ; sndMIP,
     Init!0!0 ; rcvMAP!0 ; sndDT,     Init!0!0 ; rcvMAP!0 ; sndMAA!0,
     Init!0!0 ; sndMAP ; rcvDT,       Init!0!0 ; sndMAP ; rcvMAA!0,
     Init!0!0 ; rcvMIP!0 ; rcvDT,     Init!0!0 ; rcvMIP!0 ; sndDT,     Init!0!0 ; rcvMIP!0 ; rcvMAP!1,
     Init!0!0 ; rcvMIP!0 ; sndMAP,    Init!0!0 ; rcvMIP!0 ; rcvMIP!1,  Init!0!0 ; rcvMIP!0 ; sndMIP,
     Init!0!0 ; sndMIP ; rcvDT,       Init!0!0 ; sndMIP ; sndDT,       Init!0!0 ; sndMIP ; rcvMAP!1,
     Init!0!0 ; sndMIP ; sndMAP,      Init!0!0 ; sndMIP ; rcvMIP!1,    Init!0!0 ; sndMIP ; sndMIP       }
```