Test System for a Restricted Class of LOTOS Expressions with Data Parameters

Teruo Higashino†, Gregor v. Bochmann††, Xiangdong Li†,
Keiichi Yasumoto† and Kenichi Taniguchi†

†: Department of Information and Computer Sciences, Osaka University
Toyonaka, Osaka, 560, Japan
††: Département d'IRO, Université de Montréal,
C.P. 6128, Succursale A, Montréal, Québec, H3C 3J7, Canada

Abstract

In this paper, 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 have implemented a tool for the test selection based on our technique. The derivation of a test suite for a simplified Session protocol is described as an example.

Keyword Codes: C.2.1; C.2.2; C.2.4
Keywords: Network Architecture and Design; Network Protocols; Distributed Systems

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 concerned with the test case selection problem for LOTOS expressions with interaction parameters. 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 91]). 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 relations 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 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". Especially, for the class where all variables are bounded by existential quantifiers, the problem can be solved through integer linear programming [HoUl 79]. This is the reason that many questions related to specifications written in P-LOTOS are also decidable, as further discussed in this paper.

The paper is structured as follows. The definition of P-LOTOS expressions and the corresponding extended labeled transition systems (ELTS's) is given in Section 2. In Section 3, we introduce the test case selection problem in more details by discussing some simple examples in the context of LOTOS. In Sections 4 and 5, we provide an algorithm to solve this problem. As an example, the test suite derivation for a simplified Session protocol is described in Section 6. In Section 7, a tool for the test selection based on our techniques is explained.

2. P-LOTOS expressions and corresponding extended LTSs

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 $P_1(..),..,P_k(..)$. 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. Some LOTOS operators are used for specifying the temporal ordering of the execution of events. Let B, B_1 and B_2 denote behavior 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 gates in {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. 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 behavior 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.

[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, "t1=t2", "t1<t2", "t1≤t2", "t1≥t2" and "t1>t2" are P-sentences.

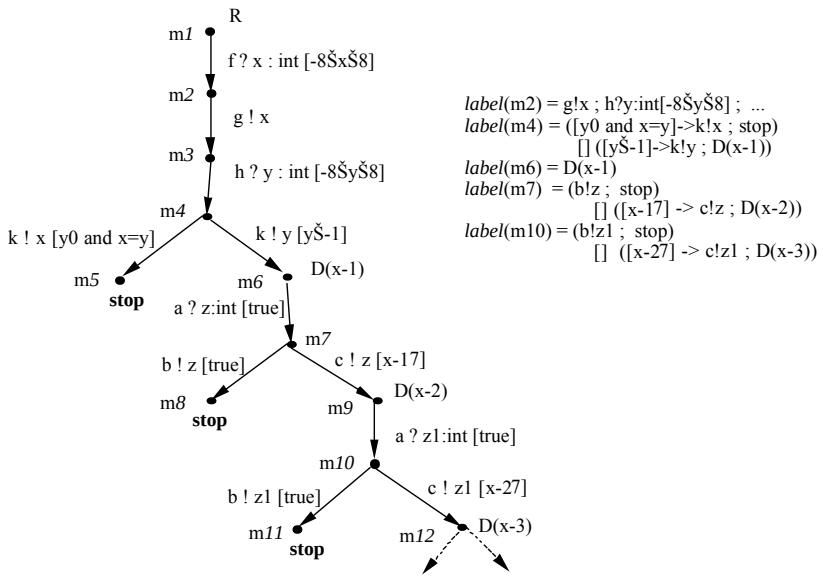

Fig.1 An Extended Labeled Transition System ELTS(t_1) for LOTOS Expression t_1

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

For example, "x+y-3" and "(x≥y-z) or (z=w)" are a P-term and a P-sentence, respectively. But, "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, we will define a tree called an "extended labeled transition system" which represents the possible sequences of events that may be executed for a given LOTOS expression "t". First, we will explain the outline of our extended labeled transition system ELTS(t) for a LOTOS expression "t". 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 behavior 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 behavior expression "B" and "B_a" denotes the behavior 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. But, 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 behavior 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 behavior expression after "f!k" is executed. With the edge *n*∅*n'*, we associate two labels *Event*(*n*∅*n'*) and *Cond*(*n*∅*n'*). The labels *Event*(*n*∅*n'*) and *Cond*(*n*∅*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.

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

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

$$\bullet\ \frac{B\ \text{-<a,c>->}\ B'}{}$$

(B [] B") -<a,c>-> B' (B" [] B) -<a,c>-> B'
B -<a,c>-> B' & a≠exit

(B ||| B") -<a,c>-> (B' ||| B") (B" ||| B) -<a,c>-> (B" ||| B')
B1 -<exit,c1>-> B1' & B2 -<exit,c2>-> B2'

(B1 ||| B2) -<exit,c1 and c2>-> (B1' ||| B2')
B -<a,c>-> B' & a≠G≈{exit}

(B |[G]| B") -<a,c>-> (B' |[G]| B") (B" |[G]| B) -<a,c>-> (B" |[G]| B')
B1 -<a,c1>-> B1' & a=ge_1$...$e$_k$ & B2 -<a',c2>-> B2' & a'=ge_1$'...$e$_k$'
& g≠G≈{exit} (Here,"$" denotes either the input symbol "?" or the output symbol "!".)

(B1 |[G]| B2) -<a,c1 and c2 and (e_1=e_1' and ... and e_k=e_k')>-> (B1' |[G]| B2')

B -<a,c>-> B' & a≠exit

(B >> B") -<a,c>-> B'>> B"

B -<exit,c>-> B'

(B >> B") -<**i**,c>-> B"

B -<a,c>-> B' & a≠exit

(B [> B") -<a,c>-> (B' [> B")

B -<exit,c>-> B'

(B [> B") -<exit,c>-> B'

B" -<a,c>-> B'

(B [> B") -<a,c>-> B'

P(x) is a process & P(x):=B(x) & B(x)-<a,c>->B'(x) & "α" is an expression & B"(x) is a behavior expression which is obtained by replacing all variables except "x" in B'(x) by new variables

P(α) -<a,c>-> B"(α)

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

Next, we will give the formal definition of our extended labeled transition systems. First, for two behavior 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 behavior 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 behavior 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*∅*n'* has two labels *Event*(*n*∅*n'*) and *Cond*(*n*∅*n'*) where *Event*(*n*∅*n'*) is "a" and *Cond*(*n*∅*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. Then, *Event*(m_9∅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) represents the set of possible event

sequences 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 define 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 || (T_R ; **stop**) is executed, then the events on the path from the root node m_1 to the node m_6 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 behavior 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, 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 Fig. 2 and we say that the branch $s_2 \varnothing s_3$ 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".

[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". []

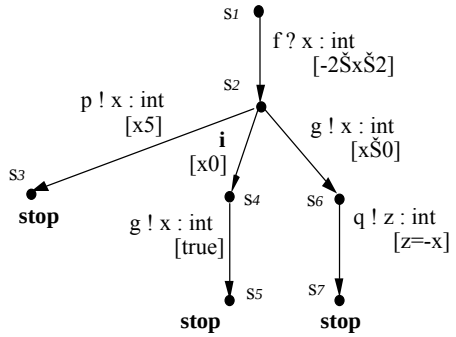

Fig.2 An Extended Labeled Transition System ELTS(t_2) for LOTOS Expression t_2

4. Basic idea for automatic analysis and test case derivation

It would be desirable that the test cases described in the above Problems could be derived algorithmically for any LOTOS expression, however, this is impossible in general [Tret 89]. Therefore, 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 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≤x≤8)
$\Psi m_4(x,y)$ = (-8≤x≤8) and (-8≤y≤8)
$\Psi m_5(x,y)$ = (-8≤x≤8) and (-8≤y≤8) and (y≥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(∃ $x_1,...,x_k$ [$\Psi s_n(x_1,...,x_k)$])". The reachability condition $\Psi s_n(x_1,...,x_k)$ is a P-sentence if "t" is a P-LOTOS expression. Then, it is decidable whether the reachability condition $\Psi s_n(x_1,...,x_k)$ is satisfiable, i.e., "∃ $x_1,...,x_k$ [$\Psi s_n(x_1,...,x_k)$]" [HoUl 79].

[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_n in the corresponding ELTS(t), we can decide whether $\Psi s_n(x_1,...,x_k)$ is satisfiable. If $\Psi s_n(x_1,...,x_k)$ 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_n if such a test case exists. []

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

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

be a disjunctive normal form of $\Psi' s_n(x_1,...,x_k)$ where each $\Psi^q s_n(x_1,...,x_k)$ (1≤q≤m) is a conjunction of some linear inequalities. So, "∃ $x_1,...,x_k$ [$\Psi s_n(x_1,...,x_k)$]" is true if and only if, for some q (1≤q≤m), $\Psi^q s_n(x_1,...,x_k)$ is satisfiable. By regarding each linear inequality in $\Psi^q s_n(x_1,...,x_k)$ as a constraint on an integer linear programming problem, we can decide whether the integer linear programming problem has integer solutions. For example, the reachability condition $\Psi m_5(x,y)$ for the node m_5 is

$\Psi m_5(x,y)$ = (-8≤x≤8) and (-8≤y≤8) and (y≥0) and (x=y)

which is a P-sentence. We find that "∃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

For any P-LOTOS expression "t", we can check whether each branch in the corresponding ELTS(t) is non-executable as follows. A branch "$s \varnothing u$" in the ELTS(t) is non-executable if the following two predicates hold.

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

(2) not(∃ $x_1,...,x_k$ [$\Psi u(x_1,...,x_k)$])

Since the predicates (1) and (2) are P-sentence, it is decidable whether the above two predicates hold. Then, it is decidable whether a given branch in the ELTS(t) is non-executable. For example, for the node m_{10} and m_{12} in Fig. 1, we can show that

(1) ∃ x,y,z,z1 [$\Psi m_{10}(x,y,z,z1)$]

= ∃ x,y,z,z1 [(-8≤x≤8) and (-8≤y≤8) and (y≤-1) and (x-1≥7)]
= true

(2) ∃ x,y,z,z1 [Ψm_{12}(x,y,z,z1)]
= ∃ x,y,z,z1 [(-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}∅m_{12}" is non-executable. It means that it cannot invoke sub-process D more than two times.

[Definition 4.1]

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.

[]

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. But, by definition, RG(t) is finite if and only if "t" is finite. For instance, although the ELTS(t_1) in Fig. 1 is infinite, the corresponding reachability graph RG(t_1) is finite since the branch "m_{10}∅m_{12}" is non-executable.

[Theorem 4.2]

If a given P-LOTOS expression "t" is finite, then the non-executable branch detection problem (Problem 2) can be solved algorithmically.

(proof) Since 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.

[]

4.3 Test suite derivation

[Theorem 4.3]

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

(proof) By Theorem 4.2, if "t" is finite, then we can construct the corresponding RG(t) algorithmically. 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 the P-LOTOS expression "t_1" in Fig. 1 is finite and the branch "m_{10}∅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 }

4.4 Discussion

For the purpose of test suite development, we assume that the objective is the coverage of all the branches in the specification. 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. 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 if the specification is finite. 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.

5. Test selection for infinite P-LOTOS expressions

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. 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.

5.1 M-test suite 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 a 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 leaf node s_n.

[Theorem 5.1]

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

(proof) Obvious. []

[Example 5.1]

t_3 = <R,D>

R:= f?x:int[-8≤x≤8] ; g!x ; h!y[-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_3 = <R,D> in Example 5.1 is an infinite system. By using the above technique, we can derive a M-test suite of "t_3" for any positive integer M. For instance, the following $TS_4(t_3)$ and $TS_6(t_3)$ are a 4-test suite and a 6-test suite of "t_3", respectively.

$TS_4(t_3)$ = { f!0 ; g!0 ; h!0 ; k!0, f!0 ; g!0 ; h!-1 ; k!-1 }

$TS_6(t_3)$ = { 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 }

Table 2 A P-LOTOS expression for Simplified OSI Session Protocol

---

$t_{Session}$ =<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))
```

---

6. Example

In this section, we will give an example of P-LOTOS expressions. A specification of a simplified Session protocol is described in Table 2. 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.

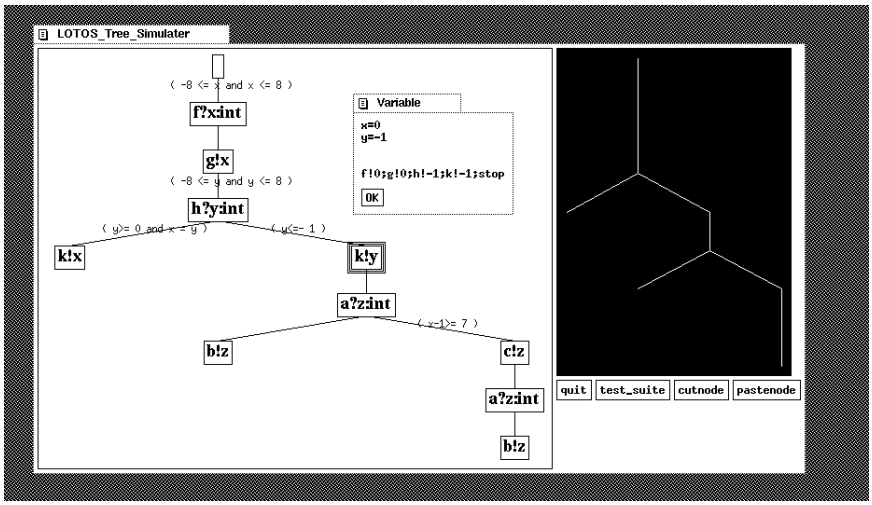

Fig. 3 Test System for P-LOTOS Expressions

7. Test system for P-LOTOS expressions

We have implemented a test system for P-LOTOS expressions [LiYa 92]. The system has the following facilities : (1) to draw a reachability graph of a given P-LOTOS expression graphically on a display as a tree, (2) to delete all non-executable branches and their descendant nodes in a given reachability graph and modify the graph, and (3) to generate a set of test cases for a given reachability graph automatically based on the techniques described in Section 4 and 5. 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 ELTSs 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 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 M-reachability graph $ELTS_M(t)$ which is a sub-graph of the reachability graph 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 P-LOTOS expression "t_1" in Example 2.1, $RG_{10}(t_1)$ is drawn in Fig. 3. The corresponding $ELTS(t_1)$ is given in Fig.1. In Fig. 3, 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 $ELTS_{10}(t_1)$. By clicking the button "*cut node*" on the display, all non-executable branches and their descendant nodes in $ELTS_{10}(t_1)$ are deleted automatically. Then, $RG_{10}(t_1)$ 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 $ELTS_M(t)$ on display, 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. 3 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 a 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. 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 of the OSI Session protocol in Section 6, it takes about 12 minutes to draw the 3-reachability graph which has 106 leaf nodes, delete all non-executable branches and their descendant nodes and generate the 3-test suite (28 test cases) using a SUN SPARCstation ELC (12MB Memory). It takes about 75 minutes to generate the 4-test suite.

8. 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 selection of a test suite can be solved by using a decision procedure for integer linear programming. 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. A tool for the test selection based on our techniques has been implemented. By using a similar technique, the question of equivalence between two specifications is solved algorithmically [HiNi 89]. 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 method for test suite development could be applied.

Acknowledgements

This work was partly supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at Université de Montréal and The Telecommunication Advancement Foundation, Japan.

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 89] 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, Oct. 1989.

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

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

[Fuji 91] 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.

[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.

[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.

[LiYa 92] X. Li, K. Yasumoto, T. Higashino and K. Taniguchi : "A Test System for a Restricted Class of LOTOS Specifications", Technical Report #91-DSP-54-4, Information Processing Society of Japan, pp.25-32, March 1992 (in Japanese).

[MaNa 92] T. Matsuura, T. Nakamura, T. Higashino, K. Taniguchi and S. Masuda : "VTM: A Graph Editor for Large Trees", Proc. of the 12th IFIP World Computer Congress'92, Madrid, Sept. 1992 (to appear).

[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.