TUGEN: A Tool for Automatic Test Suite Generation

Hao Ruibin and Wu Jianping

Department of Computer Science, Tsinghua University, Beijing, 100084

Samuel. T. Chanson

Department of Computer Science, Hong Kong University of Sci & Tech

Abstract

This paper presents a tool called TUGEN which is used for automatic test suite derivation from formal protocol specification. TUGEN is based on a formal model called EBE (External Behaviour Expression) which can be obtained from formal protocol specification in either Estelle or LOTOS. This model specifies only the external behaviour of a protocol in terms of the input/output sequences and their logical(function and predicate) relations. Based on the EBE specification of a protocol, a test sequence derivation method is used to identify associations between inputs and outputs through the interaction paths and their I/O subpaths, then generic test cases specified in TTCN (Tree and Tabular Combined Notation) can be generated from these I/O subpaths. Comparison of test cases generated from this tool and those in ISO/IEC DIS 8882 part 2 for X.25 LAPB protocol shows that the resulting set of test cases of TUGEN is concise and effective. It is our belief that TUGEN can be a powerful utility for protocol test suite generation.


1. Introduction

Conformance testing is an important stage in Protocol Engineering and its purpose is to test implementations of protocols for conformance to their specifications. For this reason, ISO has developed a standard named OSI Conformance Testing Methodology and Framework[1][2]. Based on various test architectures for OSI conformance testing which have been proposed in this standard, an IUT (Implementation Under Test) is tested
locally or remotely as a block box by observing its external behaviour to the stimuli provided by the test sequences. A test sequence is a sequence of inputs and expected outputs for the IUT. Generating test sequences from formal protocol specification is the most difficult and time-consuming work in the procedure of conformance testing. Consequently, a great deal of attention has been given to developing automatic test sequences generation tools.

At present, there exist a number of methods to automatically derive test sequences from some formal models such as the FSM (Finite State Machine). Some of them assume that the formal protocol specification is given in a particular FDL (Formal Description Language) such as Estelle or LOTOS. Generally, these methods do not take into account the data flow portion of an IUT such as PDU (Protocol Data Unit) and service primitive parameters, and only derive test sequences to test the control flow portion of an IUT. Although some test sequences derivation methods coping with values and variations of the PDU and service primitive parameters have been proposed, all of them make use of some information relating to the protocol internal structures, variables and actions that are contained in the formal protocol specifications. In our opinion, this is unnecessary and will complicate the procedure of test sequences generating. As well, the test sequences obtained by using these methods are generally less effective in terms of coverage and some protocol behaviour may be covered more than once.

This paper presents a tool called TUGEN (Tsinghua/UBC test suite Generator). This tool is used for automatic test sequence derivation from formal protocol specification. TUGEN is based on a formal model called EBE (External Behaviour Expression). This model specifies only the external behaviour of a protocol in terms of the input/output sequences and their logical (function and predicate) relations. Our test derivation strategy used in TUGEN is defined as three steps: first identifying all possible IP’s (Interaction Path) from the EBE specification of a protocol; then generating SIP’s (I/O Subpath) from the above IP’s by checking the dependences between transitions in each IP; finally, covert all SIP’s to test cases specified in TTCN. We use X.25 LAPB protocol as an example to evaluate TUGEN, and compare the result set of test sequences generated by TUGEN with those in ISO/IEC DIS 8882 part 2. Comparison shows that our test derivation method is simple and concise, as well, the result set of test sequences of TUGEN is
effective and have a good coverage. So, it is our belief that TUGEN tool can be a powerful utility for protocol test sequence generation.

This paper proceeds as follows. In Section 2, a brief introduction to the EBE (External Behaviour Expression) model is given. Section 3 describes the structure of TUGEN and discusses some technical issues encountered in the design and implementation of TUGEN. An example of deriving test suite from an EBE specification of X.25 LAPB protocol and some comparison with ISO/IEC DIS 8882 part 2 are shown in Section 4. Finally, Section 5 concludes this paper.

2. Theoretical foundation of TUGEN

Our TUGEN tool is based on a formal model called EBE (External Behaviour Expression)[3]. The reason we propose this new formal specification model is that applications of those existing FDT (Formal Description Techniques) such as Estelle and LOTOS to ISO protocols have always included internal structures, variables and actions of the protocols[5][6][7]. Making use of information relating to the protocol's internal structures and variables will complicates the procedure of test sequences generating. In the same time, the effect of the internal variables on the states transition can be expressed by the external observable behaviour. So, we choose EBE which describes only the external behaviours of a protocol as our tool's theoretical foundation. The details of EBE model are described as follows.

2.1 Basic definition of EBE

The EBE (External Behaviour Expression) models the externally observable behaviour of a system in terms of possible sequences of interactions exchanged between the system and its external environment, and possible logical (function and predicate) relations among elements (input and/or output primitives and their parameters) of these sequences.
Definition 2.1

An External Behaviour Expression (EBE) is a quadruple \( EBE = <S, s_0, T, R> \)

where:
- \( S \) is a set of external finite states of a system modeled by EBE;
- \( s_0 \in S \) is the initial external state of the system;
- \( T \) is a set of transitions between external states;
- \( R \) is a set of logic relations of transitions.

External states of a system are pause states of interactions exchanged between the system and its external environment.

Definition 2.2

A transition between external states is the interactions exchanged between the system and its external environment in terms of input and/or output primitives and their parameters. The general form of a transition is given by \( T_{ij}=<l_p, o_q> \), where \( T_{ij} \in T, l_p \in l, o_q \in O \) and:

1) \( l \) is a set of input primitives from the external environment, and each input primitive is denoted by: \( l_p(X_{p1}, ..., X_{pn}) \), where \( l_p \) is the input primitive identifier, and \( X_{p1}, ..., X_{pn} (n>=0) \) are parameters of the input primitive \( l_p \).

2) \( O \) is a set of output primitives to the external environment of the system, and each output primitive is denoted by: \( o_q(Y_{q1}, ..., Y_{qm}) \) where \( o_q \) is the output primitive identifier, and \( Y_{q1}, ..., Y_{qm} (m>=0) \) are parameters of the output primitive \( o_q \).

3) The absence of an input primitive or an output primitive is denoted by "--".

Definition 2.3

The set of logical relations of a transition \( R_{ij}=(F,P) \) holds if and only if there exists a transition \( T_{ij} \) from state \( s_i \) to state \( s_j \).

1) \( Z \) is the group of elements which influence the output of transition \( T_{ij} \).

The elements are typically input primitives and their parameters and
may also include protocol parameters mentioned in system implementation statements.

2) \( F \) is a set of function relations of transition \( T_{ij} \). The output parameters \( Y_{q1}, ..., Y_{qm}(m>0) \) of transition \( T_{ij} \) will be produced if and only if there exists \( Z \) which satisfies a set of function \( F(Z) \), i.e., \( \{Y_{q1}, ..., Y_{qm}\} = F(Z) \).

3) \( P \) is a set of predicate relations of transition \( T_{ij} \). The transition \( T_{ij} \) will happen if and only if there exists \( Z \) which satisfies the property \( P(Z) \).

Definition 2.2 and 2.3 are used to model the external behaviour of a system, and deal with values and variations of the parameters of the input and output primitives.

Except the logical relations \( F \) and \( P \), some operations may occur during a transition. These operations are mainly used to deal with system global variables and some additional actions that need to be done during a transition.

### 2.2 Description of EBE

There are two ways to describe the formal EBE model. One way to describe this model is by a directed graph with tree structure which we call Behaviour Tree oriented EBE (EBE-BT). The other method is called Normal Form oriented EBE (EBE-NF).

In the EBE-BT, tree nodes represent externally observable states of a system (i.e., the set \( S \)). In particular, the root of the tree is the initial state \( s_0 \). Tree branches linking tree nodes represent transitions among the external states of the EBE (i.e., the set \( T \)). Logical relations associated with a transition of the EBE may be described in terms of additional specifications (i.e., the set \( R \)). Thus it can be seen that there are clear mapping between the basic definition of EBE and the EBE-BT. In our TUGEN tool, we use EBE-BT as the internal data structure to store information included in protocol's EBE specification.

The Normal Form oriented EBE (EBE-NF) is another form of describing a system using EBE. The typical framework of a system's specification using EBE-NF is showed below:
SPECIFICATION protocol-name
[list of input/output primitives and their parameters]
CONST
  system constant list
VAR
  system variable list
FUNC
  system global function definition
BEHAVIOUR
  system external behaviour expression
ENDSPEC

In EBE-NF, an essential component of a system specification is its external behaviour expression. An external behaviour expression is built by applying the syntax rules of the EBE-NF that are described using BNF (Backus-Naur Form) which is simpler than the FDT's such as Estelle or LOTOS because it describes only the externally observable behaviour.

The EBE of a protocol may be produced directly from the protocol document in English. However, we are more interested in derivation from FDT's such as Estelle or LOTOS. A framework of obtaining an EBE from formal protocol specification in Estelle and LOTOS is given in [4]. The formal structures of Estelle are very similar to those of EBE. The tree structure of EBE associated with the states and transitions can be formally obtained from the finite state machines in Estelle. Logical relations of EBE can be formally produced by searching the operation part associated with each transition of Estelle. Thus, formal protocol specification in Estelle can be directly transformed into EBE by using some transformation rules. Now we are developing a tool to transform formal protocol specification in Estelle and LOTOS into EBE. The operational semantics of LOTOS provides a means to derive a transition tree from its behaviour expression. This transition tree has the same structure as EBE-BT. Logical relations of the EBE can be obtained by examining the description of data structures and value expressions in LOTOS. Therefore, formal protocol specification in LOTOS can also be transformed into EBE.

2.3 Test sequence derivation strategy used in TUGEN

In [1,2], test sequences for protocol conformance testing are known as test suites. Test suites have a hierarchical structure in which the basic unit
is the test case. Each test case has a narrowly defined purpose. ISO has defined a notation called TTCN (Tree and Tabular Combined Notation) for abstractly specifying test suites. Test sequences generated by our TUGEN tool is specified in TTCN. In TUGEN, we adopt the test sequence derivation strategy proposed in [3] and do some extension and revision to it. This test sequence derivation strategy is defined in three steps.

2.3.1 Identifying all interaction paths (Step 1)

Definition 2.4 (interaction path, interaction path with loop)

An interaction path IP is the externally observable track on which a sequence of interactions between the protocol and its external environment occurs, starting from the initial external state $s_0$ of the protocol and ending in the same state. Any loop (or cycle) interaction in an IP is travelled only once. Interaction Path with loop is an interaction path in which there exists one or more loop interactions.

In this step, we use the algorithm A proposed in [3] to identify all possible interaction paths from the EBE specification of a protocol. Due to space limitation, details of algorithm A are not presented here.

2.3.2 Identifying all I/O subpaths (Step 2)

Definition 2.5 (I/O subpath)

An I/O subpath SIP is the externally observable track $e_1, ..., e_k$ in an IP, where:

1. $e_1$ is an input primitive $I_1$ with its parameters $X_{11}, ..., X_{1n}$ ($n >= 0$) and $e_k$ is an output primitive $O_k$ with its parameters $Y_{k1}, ..., Y_{km}$ ($m >= 0$);
2. $e_k$ is obtained when logical relations satisfy:
   a) property $P_k(Z)$; and/or
   b) a set of functions $\{Y_{k1}, ..., Y_{km}\} = F_k(Z)$, where $Y_{ki} = F_{ki}(Z)$
      ($m >= i >= 0$);
3. $e_1$ is the input of the earliest preceding transition which contains the elements $Z$ satisfying $P_k(Z)$ and/or $F_{ki}(Z)$ ($m >= i >= 0$).
In this step, we use the algorithm B proposed in [3] to identify all I/O subpaths by checking the dependences between elements of each IP generated in Step 1. Also due to space limitation, details of algorithm B are not presented here.

2.3.3 Deriving test cases in TTCN from SIP's (Step 3)

A test case is a complete and independent specification of the actions required to achieve a specific test purpose, defined by a test body starting in a stable testing state and ending in a stable testing state. Furthermore, a test suite is the complete set of test cases for a specific protocol. Each SIP generated by Step 2 is only a correctness sequence for a specific test purpose in the procedure of test executing. Based on the SIP, a test tree with the correctness, nondeterministic and defense branches can be derived, and three kinds of verdicts are assigned to the each leaf of the test tree. The test tree can be mapped to a test case in TTCN. Finally, these test cases are grouped according to some rules, and are mapped to a complete test suite in TTCN.

1. Deriving test cases in TTCN from SIP's

Definition 2.6

A test tree is a rooted and finitely I/O branching tree for a specific test purpose, where each branch is labelled either an input primitive I with its parameters or an output primitive O with its parameters.

Definition 2.7

A test tree is said to be nondeterministic if there exist more than one node which has a same input branch labelled by a same input primitive and more than one difference output branches labelled by difference output primitives, otherwise it is said to be deterministic.

Definition 2.8

A preamble of a state is the shortest executable path from the initial state $s_0$ to this state, a postamble of a state is the shortest executable path from this state to the initial state $s_0$.

In this step, we first derive all test trees from the SIP's of a protocol. These test trees are consisted of correctness, nondeterministic, or defensive
branches. Certainly, a test tree generated by this step is either deterministic, or nondeterministic. Some test trees have the same prefix, which means there is a transition sub-sequence beginning from the initial state exiting in each of these test trees. We should combine these test trees into one test tree that has several branches.

One of three kinds of verdicts can be assigned to each leaf of the test tree. For each leaf of correctness branches, a PASS test verdict is assigned. For each leaf of nonterministic branches, a INCONCLUSIVE test verdict is assigned. Furthermore, for each leaf of defensive branches, a FAIL test verdict is assigned.

For each state in EBE-BT, we find out a preamble and a postamble from the IP generated by Step 1. A identifier is assigned to each preamble and postamble for further reference.

After finishing above working, We use the Algorithm B proposed in [9] to map a test tree to a test case in TTCN.MP.

2. Deriving a test suite in TTCN

A test group is a named set of related test cases [1]. Our grouping strategy of test cases is that those test cases with the same starting test belong to one test group.

The final step of our test suite generation is to derive a test suite in TTCN. The Suite Overview part, the Declarations part, and the constraints part of a test suite can be directly extracted from the information contained in the EBE specification of a protocol. These grouped test cases are included in the Dynamic Part of a test suite.
3. The architecture of TUGEN

3.1 Overview

The architecture of TUGEN is shown in Figure 1. TUGEN is implemented using C in a workstation as a system integrating five modules. These modules are EBE parser, IP generation, SIP generation, Test suite generation and support routines. All these modules are controlled by a main procedure called TUGEN.

![Diagram of TUGEN architecture](image)

Figure 1. Architecture of TUGEN

The data flow relationship between these modules are described as follows:

1) Given a EBE-NF specification of a protocol, EBE parser translate it into an internal data structure in the form of corresponding EBE-BT.

2) IP generation module identifies all possible IP’s (Interaction Paths) from EBE-BT and outputs them into an IP table.

3) SIP generation module identifies all SIP’s from these IP’s generated in the previous step.

4) Based on the SIP’s generated in the previous step, Test Suite Generation module derives a test suite specified in TTCN as a final output.

3.2 Preliminary data structure of TUGEN

In TUGEN, we use two tables to store information extracted from the EBE specification of a protocol. All modules in TUGEN use these two
tables to finish their corresponding manipulation. These two tables are transition table and state table.

### 3.2.1 Transition table

The **EBE** parser stores the information extracted from an **EBE** specification into the transition table. Each entry in this table represents one transition described in the **EBE** specification, including its initial state, destination state, input/output primitives and their parameters, relationship (function and predicate) and operation bound to this transition. The table contains exactly the same information as the **EBE** specification of a protocol. The difference between the them is that **EBE** specification is designed for human-readability while the transition table is designed for computer-readability.

All entries in the transition table have the same size, this allows fast random access via array indexing. Variable-sized data such as the function body are stored in the table by using pointers to a dynamically allocated memory block.

### 3.2.2 State table

Pointer fields in each entry of the transition table allow the transitions to be linked into a transition tree identical to the Behaviour Tree form of specification ( **EBE-BT** ). The transition tree make use of a separate state table (which can be directly generated from transition table). Figure 2 shows how two tables are linked together as a sample transition tree.

The state table contains one fix-sized entry for each state. Each entry contains a count of number of outgoing transitions and a pointer to a list of transitions leaving the state. Each list of transition is a variable-sized, dynamically allocated array of transition table's indices.
3.3 Main function of each module

In this part, we give a brief illustration of the main function of each module, and discuss some technical issues and their solutions encountered in the implementation of TUGEN.

3.3.1 EBE parser

This module takes an EBE-NF specification as input, and produces the EBE-BT as output. It includes a lexical scanner and a syntax parser. The lexical scanner breaks up the input into tokens. For simplicity, the scanner returns tokens as character strings rather than enumerated values which are adopted in usual parsers. The syntax parser produces EBE-BT by using these tokens and the production grammar of EBE-NF.

3.3.2 IP generation

This module is used to generate all possible IP’s (Interaction Paths) from the EBE-BT. It is based on the algorithm A proposed in [3]. It basically generates all interaction paths through the tree from the root to each leaf. In
our implementation, the algorithm used is slightly different from the one specified in [3]. This implementation is simple and easier to understand because of the use of recursion.

The implementation of algorithm A is as follows:

1) Call step 2 with parameters (IP = NULL, NODE = root).
2) Given a IP and a tree NODE:
   If the NODE is a leaf
       The IP is complete, store it in the IP table.
   Else
       For each outgoing transition of this NODE:
           If it is not a loop transition:
               Fork off a copy of the current IP with the outgoing
               transition appended.
               Recursively call step 2 with parameters (IP = the new IP,
               NODE = destination node of the outgoing transition).
           Else /* loop transitions */
               Append all loop transitions in current state to current IP,
               then search a path to the initial state s0 and append it to
               current IP.

The simple intent of this implementation makes it easy to verify that it produces the same results as the original algorithm in [3].

3.3.3 SIP generation

Based on the algorithm B proposed in [3], this module refers to the IP generated by the above module and identify all SIP's from these IP's by checking the dependent relationship between transitions in each IP.

How to automatically identify the dependent relationship between transitions in each IP is an obstacle we encountered. Two transitions that having dependent relationship are defined as follows: the second transition's output primitive and its parameters are influenced by the first transition's input primitive, its parameters, and some parameters defined in the protocol implementation statement but been referred in this transition. In order to solve this problem, we construct two tables for each transition, RHSref_table and inparm_table. The RHSref_table stores all elements of Z which is defined in Definition 2.3. The inparm_table stores the input primitive's parameters of this transition and some system parameters defined in the
protocol implementation statement but refered in this transition. To identify dependent relationship between transitions, we search the second transition's \textit{RHSref_table} and the first transition's \textit{inparm_table} to see whether or not there exist some elements' mapping relations. If exist, we say the second transition depends on the first transition, so they should belong to the same SIP.

3.3.4 Test Suite generation

Based on the \textit{EBE} specification of a protocol and SIP's generated by above module, this module produces a test suite specified in TTCN as a final output. It uses the strategy described in 2.3.3.

In the practical executing, we found that there may be several pathes simultaneously satisfying the condition to be a preamble or postamble. we only choose the first path which satisfies the condition to be the preamble or postamble.

Sometimes we find that several SIP's in one group may have the same prefix, i.e., the input and output primitives and their parameters in the starting part of these SIP's are identical. So, we should do some combination to these SIP's and produce one test case as final result. In \textit{TUGEN}, procedure to deal with this situation is under developing.

3.3.5 Support routines

This module provides the common subroutines which are needed by the above four modules. They are mainly the operations which are used to access the internal data structures such as various tables. The reason we seperated these subroutines from each module is that we want to provide a good data encapsulation, which is a basic technique used in OOP (Object-Oriented Programming).

3.3.6 Main procedure \textit{TUGEN}

Main procedure \textit{TUGEN} coordinates each module to make them work together. Some helping information is provided by this Main procedure.
4. Evaluation of TUGEN

In this Section, we use the EBE specification of X.25 LAPB protocol as a example to evaluate the effectiveness of our TUGEN tool.

4.1 EBE specification of X.25 LAPB protocol

To ensure data equipment compatibility, CCITT has proposed Recommendation X.25, which is a standard interface protocol between Data Circuit Terminating Equipment and packet mode Data Terminating Equipment in packet networks. LAPB (Link Access Procedure Balanced) is the protocol of data link layer in X.25. Because X.25 LAPB does not define states explicitly, we define the following seven states to get a EBE specification[9].

S1 - Disconnect Phase
S2 - Link Disconnect and T1 started
S3 - Link Setup and T1 started
S4 - Normal Information Transfer
S5 - Frame Reject and T1 started
S6 - DTE Busy
S7 - Reject Sent

Based on above seven states and the text of X.25 LAPB protocol, we make a EBE specification of X.25 LAPB. Following is a part of the EBE specification of X.25 LAPB protocol which describes the behaviour of S0.

```
S0 [ DISC (0), DM (0) ]
FUNCTION: DM.f = 0;
PREDICATE: true;  J* S0 { T0-0(1) | R0-0(1) }

+S[ DISC (1), DM (0) ]
FUNCTION: DM.f = 1;
PREDICATE: true;  J* S0 { T0-0(2) | R0-0(2) }

+S[ SABM (0), DM (0) ]
FUNCTION: DM.f = 0;
PREDICATE: true;  J* S0 { T0-0(3) | R0-0(3) }

+S[ SABM (1), DM (0) ]
FUNCTION: DM.f = 1;
PREDICATE: true;  J* S0 { T0-0(4) | R0-0(4) }

+S[ DM (0), - ]
FUNCTION: nil;
PREDICATE: true;  J* S0 { T0-0(5) | R0-0(5) }
```

15
+[ DM (0), DISC (p) ]
FUNCTION: DISC,p = 1;
PREDICATE: true;
J" S1  \{ T0-1 | R0-1 \}

+[ DM (0), SABM (p) ]
FUNCTION: SABM,p = 1;
PREDICATE: true;
J" S2  \{ T0-2 | R0-2 \}

+[ SABM (0), UA (f) ]
FUNCTION: UA,f = 0;
PREDICATE: true;
J" S3  \{ T0-3(1) | R0-3(1) \}

+[ SABM (1), UA (f) ]
FUNCTION: UA,f = 1;
PREDICATE: true;
J" S3  \{ T0-3(2) | R0-3(2) \}

4.2 Results of application of TUGEN

We applied TUGEN to the EBE specification of X.25 LAPB protocol and get 1229 IP, following are several IP's:

IP[1]: S0, T0-0(1), S0
IP[2]: S0, T0-0(2), S0
IP[5]: S0, T0-1(1), S1, T1-0(1), S0
IP[6]: S0, T0-1(1), S1, T1-0(2), S0
IP[10]: S0, T0-1(1), S1, T1-1(1), S1, T1-1(2), S1, T1-0(1), S0

From these IP's, We get 85 SIP, following are several SIP's(test purpose of each SIP is omitted here):

SIP[1]: T0-0(1)
SIP[2]: T0-0(2)
SIP[3]: T0-0(3)
SIP[4]: T0-0(4)
SIP[5]: T0-0(5)
SIP[6]: T0-1
SIP[7]: T0-2
SIP[8]: T0-3(1)
SIP[9]: T0-3(2)

Then, these SIP are coverted into test trees and those test trees which have the same prefix are combined into one test tree containing several branches. Test trees are grouped into seven groups according to their initial state. Each of these seven groups has a preamble and a postamble. We choose one test case generated from these SIP's as an example to illustrate the result form of test case( only the main part of this test case is listed here):
To evaluate the effectiveness of our TUGEN tool, we made a comparison between the test suite generated by TUGEN and that in ISO/IEC DIS 8882's part 2. For convenience sake, in the following discussion, we use test suite LAPB to represent the test suite generated by TUGEN, use standard test suite to represent the test suite in DIS 8882's part 2.

We adopted the same grouping method of test cases in TUGEN as the one used in the standard test suite, that is, test cases having the same starting state belong to one group. We compare the test suite LAPB with the standard test suite, get the following conclusions:

1) The Suite Overview part, Declaration part of test suite LAPB are very similar to those in the standard test suite.

2) The Dynamical Behaviour part of test suite LAPB covers the basic part of those in the standard test suite. These can be seen from TABLE 1.

<table>
<thead>
<tr>
<th>Group #</th>
<th>test suite LAPB</th>
<th>standard test suite (proper part)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>

TABLE 1. Comparison Result

Our TUGEN generated some test cases which are not included in the standard test suite. This is because our test derivation strategy emphasize the dependent relationship between transitions, so some combination of test cases in standard test suite are treated as test cases in test suite LAPB. In our opinion, these part of test cases are correct and should be executed in testing.
3) Some test cases in standard test suite do not appear in test suite LAPB. The reason for this is that those test cases are used to test the defensive behaviour of the IUT. Defensive behaviour of an IUT is the behaviour which can be observed externally when the IUT receives a improper or unfortunate interaction. These test cases cannot be directly generated from the EBE specification of a protocol. How to generate this kind of test cases is still a theoretical problem which need further research.

In summary, our TUGEN tool basically achieves the goal of automatic test sequence generation from the formal specification of a protocol. The test cases generated by it are concise and effective.

5. Conclusions

As already mentioned, this paper presents a tool named TUGEN for automatic test sequences generation from the formal specification. This tool is based on the EBE specification of a protocol which can be obtained from formal specification in Estelle or LOTOS. In TUGEN, we use the test sequence derivation strategy proposed in [3] and do some extension and revision to them. By applying this tool to the EBE specification of X.25 LAPB protocol, we confirm the correctness and effectiveness of TUGEN.

At last, we believe our TUGEN tool will become a powerful utility for protocol conformance testing.

Acknowledgement

Thanks to Dennis and Stuart in University of British Columbia for their intelligent working in the early designing of our TUGEN tool.

References


