TESTGEN+ : An Environment for Protocol Test Suite Generation, Selection and Validation

Sangho Lee and Son T. Vuong
Technical Report 93-25
July 1993

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1Z2

TESTGEN+: An Environment for Protocol Test Suite Generation, Selection and Validation

Sangho Lee*and Son T. Vuong

Dept. of Computer Science
University of British Columbia
Vancouver, BC Canada V6T 1Z2
Email : shlee@cs.ubc.ca, vuong@cs.ubc.ca

Abstract

Protocol testing is an important phase in the overall protocol development process. In this paper, we present and discuss TESTGEN+, an integrated environment for protocol test suite generation, selection, and validation, developed at the University of British Columbia. This environment is menu driven and unique in that it is constraint oriented, and is thus general and flexible. It is based on an intermediate extended transition system formalism and test coverage metrics and directly supports ASN.1 and Estelle. The test generation component TESTGEN deals with both the control flow and the data flow testing. Test selection, TESTSEL is based on coverage metric and is useful in reducing the size of a large test suite generated. Test generation and selection are integrated and guided by user-defined test suite generation constraints and parameter variation constraints as well as test selection parameter constraints. In addition, a test validation facility, TESTVAL, is integrated into TESTGEN+ to allow for the validation of test cases with respect to a given specification. The environment will serve as a useful test-bed for experimenting with protocol test generation, selection, and validation as well as being a productive system for developing useful test suites for real-life protocols.

1 Introduction

To ensure interworking among communication systems, protocol implementations should be submitted to a test center for conformance testing [ISO-a]. An implementation under test (IUT) is said to conform to the respective standard if no discordance

*Sangho Lee is a visiting research professor from Chungbuk National University of Korea.

between the external behavior of the IUT and the standard can be found. To detect potential errors in an IUT the test system exchanges service primitives (SP) with the IUT according to the test scenarios in a test suite. CCITT and ISO have developed a standard notation to describe test suites for OSI conformance testing, known as the tabular and tree combined notation (TTCN) [ISO-b].

The generation of TTCN test suites is a tedious and repetitive process. Test suites must often be updated or rewritten because both the protocol specifications and the TTCN standard are subject to periodic modifications. Also the test suite should be reliable enough in terms of coverage to provide a confidence on the conformance testing results [McA-92]. Thus an environment for test suite generation, selection and validation will be essential for protocol conformance testing.

Most well known protocol test generation methods are based on the finite state machine (FSM) model and are thus inherently limited to the testing of the control flow part of the protocol [Sid-89, Fuj-91]. The data part of OSI protocols is often more complex than the control part, specially in higher layer protocols. In order to ascertain the correctness of a protocol implementation and its conformance to a standard, it is essential to test on both the control part and the data part of the protocol. Some techniques have been proposed to handle both the control flow and the data flow of protocol, using basically static data flow analysis [Ura-88], and parameter variations. In general, these techniques generate very large test suites, especially those dealing with data flow testing. Test selection is thus essential to reduce large test suites to manageable subsets, which retain the "most representative" test cases. Objective coverage measures are needed to guide the test selection process [Vuo-91]. When test suites are developed heuristically, their validations are necessary to ensure the test suites developed are valid according to some formal specification. The problem of test validation is basically equivalent to the one of trace analysis since a test case can simply be viewed as a trace.

In this paper, an integrated environment for test suite generation, selection, and validation, called TESTGEN+ is discussed and the experiments performed using

TESTGEN+ are presented. TESTGEN+ consists of three major functions: test suite generation, selection, and validation, which can be used separately or in combination. The remainder of this paper is organized as follows. Section 2 provides some theoretical background material for TESTGEN+ including the definition of the extended transition system (ETS), testing distance, and coverage metrics. An overview of TESTGEN+ is presented in Section 3 with the three components, TESTGEN, TESTSEL and TESTVAL, which are explained in more details in the subsequent sections. Section 4 describes the test suite generation(TESTGEN) component, including a brief overview of the data structure for the internal protocol representation, the Estelle and ASN.1 parser, the constraint editor and the test suite generation (TSG) engine. In Section 5 we present the test case selection (TESTSEL) algorithm based on coverage metrics. Test validation (TESTVAL) for protocols specified in Estelle is discussed in Section 6. We explain briefly how to use TESTGEN+ in Section 7. In Section 8 we present and discuss the overall results obtained from some well known In-Res, TP0 and LAPB protocols using TESTGEN+. Finally, some concluding remarks and suggestions for further work are offered in Section 9.

2 Theoretical Background

In this section, we introduce some basic concepts and relevant definitions used in the theories behind TESTGEN+. TESTGEN+ accepts a single-module Estelle specification and ASN.1 specification which are assumed to be error-free and to conform to the standard. The accepted specifications will be transformed into an internal formalism called extended transition system(ETS). The following basic definitions are used.

Definition 1 *A finite state machine (FSM) F is defined as a 4-tuple* $F=< S, L, \Sigma, I_o >$.

- *S is the set of states of F.*
- *L is a set of labels on the transitions of F and each label is represented in the format "input symbol/output symbol".*

- $\Sigma \subseteq S \times L \times S$ *is a relation of the transitions.*
- I_o *is the initial state.* □

Definition 2 *A single-module Estelle specification E is defined as a 5-tuple* $E=<S,P,\Delta,I_o,D>$.

- S *is the set of states of E.*
- P *is a set of program segments, each program segment associated with a transition of E.*
- $\Delta \subseteq S \times P \times S$ *is a relation of the transitions.*
- I_o *is the initial state.*
- D *is declaration and/or initialization of variables and interaction service primitive.* □

The FSM F can be viewed as an abstraction of the corresponding Estelle specification E. When the domain sizes of the parameters of the service primitives are small and also the interaction service primitives and their parameters can be enumerated the mapping from an Estelle specification E to a finite state machine F is straightforward [Lu-91].

Definition 3 *An* extended transition system *(ETS) is a quadruple* $ETS = (Q, E, \rightarrow, q_{init})$

- *"*$Q = States \times Variables \times Constants \times Timers$*" is the set of states of the ETS.*
- *"*$E = ISP \cup OSP \cup PDU$*" is the set of events of the ETS.*
- *"*$\rightarrow \subseteq Q \times E \times Q$*" is the transition relation for the ETS.*
- *"*$q_{init} \in Q$*" is the initial state of the ETS.* □

The ETS model we use is based on Keller's labelled transition systems [Kel-76]. Since we are parsing Estelle and ASN.1 specifications to generate an internal protocol data structure (PDS) of the ETS, we borrow the Estelle terminology in naming some of the ETS elements. A transition can be executed iff an ISP and/or PDU associated with the transition(if any) is received and if the enabling predicate is true. As the transition fires, the associated action function is executed atomically. Variables and timers are set, OSP(s) and/or PDU(s) are assembled and sent. The semantics of enabling predicates and action functions are similar to the semantics of the Estelle *enabling clauses* and *statements* except for two important protocol aspects: the data part of ISPs, OSPs, PDUs and the time handling.

Definition 4 *Subtours(X,S_i,S_o,C) is the set of all executable paths(subtours) from an initial state S_i to a final state S_o which is the initial state of an Estelle (or finite state machine) specification X with Constraint C.* □

Typically $S_i = S_o$ since most communication protocols begin and end at the same initial state S_o. Test cases based on subtours have two advantages : First, the initial state (usually "*idle*") is the most stable or trustworthy protocol state and can always be reached (e.g. with "*reset*"). Second, a "user session" with the IUT will most naturally start and finish at the initial state so that a subtour can be mapped to a meaningful test case with a well defined test purpose. The constraint C bounds the set of feasible paths and includes restrictions on the iteration number of while loops and transition loops, and conditions limiting the domain space of variables and interaction service primitive parameters for *Estelle*. Since finite state machines have no program segments and do not deal with data and predicates, the constraints C_f of a finite state machine derived from an *Estelle* specification E using the mapping procedure [Lu-91] is a subset of the constraints C_e of E.

Definition 5 *The set of test cases TC generated from a specification X (E or F) with constraints C, initial state S_i, and final state S_o, is $TC = \{tc \mid tc = \mathcal{T}(p)$ such that $p \in Subtours(X, S_i, S_o, C)\}$ where $\mathcal{T}$ is a one-to-one function from a path to a test*

case tc. □

The set TC consists of a set of test cases *tc* exhibiting external behaviors of specification X. A test case *tc* (sometimes called test sequence or derivation interchangeably) can be represented by a sequence of input and output service primitives.

Definition 6 *A test case tc is valid with respect to specification X if it corresponds to an element in Subtours(X,S_i,S_o,C) derived from the specification X.* □

Definition 6 states that a test case *tc* of a specification X with constraints C is valid with respect to the specification X if the external behaviors of the test case is a subset of the allowable behaviors specified by X.

Definition 7 *Let $A = (a_1, \alpha_1), \ldots, (a_K, \alpha_K)$ and $B = (b_1, \beta_1), \ldots, (b_L, \beta_L)$ be an abstract representation of two test cases. Every a_i and b_j consist of input and output service primitives, I_i/O_i and I_j/O_j respectively. And each α_i and β_j is number of recursion of the ith and jth service primitive element a_i and b_j, respectively. The measure of the difference between the levels of recursion of the respective* kth *elements of two test cases A and B of length K and L, respectively, is defined as follows:*

$$\delta_k = \begin{cases} 0 & \text{if } a_k = b_k \text{ and } \alpha_k = \beta_k \\ \mid \alpha_k - \beta_k \mid & \text{if } a_k = b_k \text{ and } \alpha_k \neq \beta_k \\ \infty & \text{if } a_k \neq b_k \end{cases}$$

for $k = 1, 2, \ldots, min\{K, L\}$, and take $\delta_k(A, B) = \infty$, for $min\{K, L\} < k \leq max\{K, L\}$. □

Definition 8 *Let A and B be two derivations in a test suite TS. In abstract representation, $A = \{(a_k, \alpha_k)\}_{k=1}^{K}$, and $B = \{(b_k, \beta_k)\}_{k=1}^{L}$, K,L $\in N \cup \{\infty\}$. The testing distance* **dt** *between any two derivations A,B in TS is defined as follows:*

$$\mathbf{dt}(A, B) = \sum_{k=1}^{max\{K,L\}} p_k r(\delta_k(A, B))$$

*where p_k represents the weight of the k*th *service primitive element reflecting the weight of the computational pattern and $r(\delta_k(A,B))$ represents the weight of the difference in the level of recursion of that element within the derivations.* □

Definition 9 *The coverage of a (sub)set of test cases TC with respect to a test suite TS is defined as*

$$COV(TC,TS) = 1 - m(TC,TS)$$

where $$m(TC,TS) = \frac{sup\{d(x,TC) | x \in TS \setminus TC\}}{\sum_{k=1}^{\infty} p_k}$$

and $$d(x,TC) = inf\{\mathbf{dt}(x,y) \mid y \in TC\}$$

where $m(TC,TS)$ represents the normalized distance between TC and TS and $d(x,TC)$ represents the distance between element (test case) x and the set TC. □

More details on testing distance and coverage metrics can be found in [Vuo-91].

3 Overview of TESTGEN+

Well known hardware and software test methods have been applied to conformance testing with various degrees of success. [Hsi-71] describes checking experiments for fault detection in sequential machines in which a "machine under test" is checked against a state table definition. Refinements of this method have led to the transition tour method [Nai-81] and characterizing sequence based methods such as U [Sab-88], D [Gon-70] and W-methods [Cho-78]. The papers by Sidhu [Sid-89] and Fujiwara [Fuj-91] provide a nice survey and comparison of these methods. Even though these methods have been widely improved and optimized [Cha-89] [Vuo-89] they still have two major shortcomings. First, they are weak in discovering errors due to additional states or transitions in the implementation. Second, they only address the control part

of protocols. TESTGEN+ is a protocol TEST Generation, selection, and validation ENvironment for conformance testing developed at the University of British Columbia [Vuo-93]. The environment provides a menu-driven interface for generating, selecting and validating protocol test suites. The test suite generation component incorporates both the control flow testing and the data flow testing with parameter variation. Both types of testings are controlled by a set of user-defined constraints which allows a user to focus on the protocol as a whole or just on restricted areas of the protocol during test suite generation. This constraint based test suite generation method is general and flexible, and can produce test suites with a various levels of fault coverage and complexity by using appropriate constraints. TESTGEN+ has been used to generate, select, and validate test suites for practical protocols such as the InRes, OSI class 0 transport and LAPB protocols [Vuo-93]. The overall functional structure of TESTGEN+ is depicted in Figure 1. TESTGEN+ consists of three major functions:

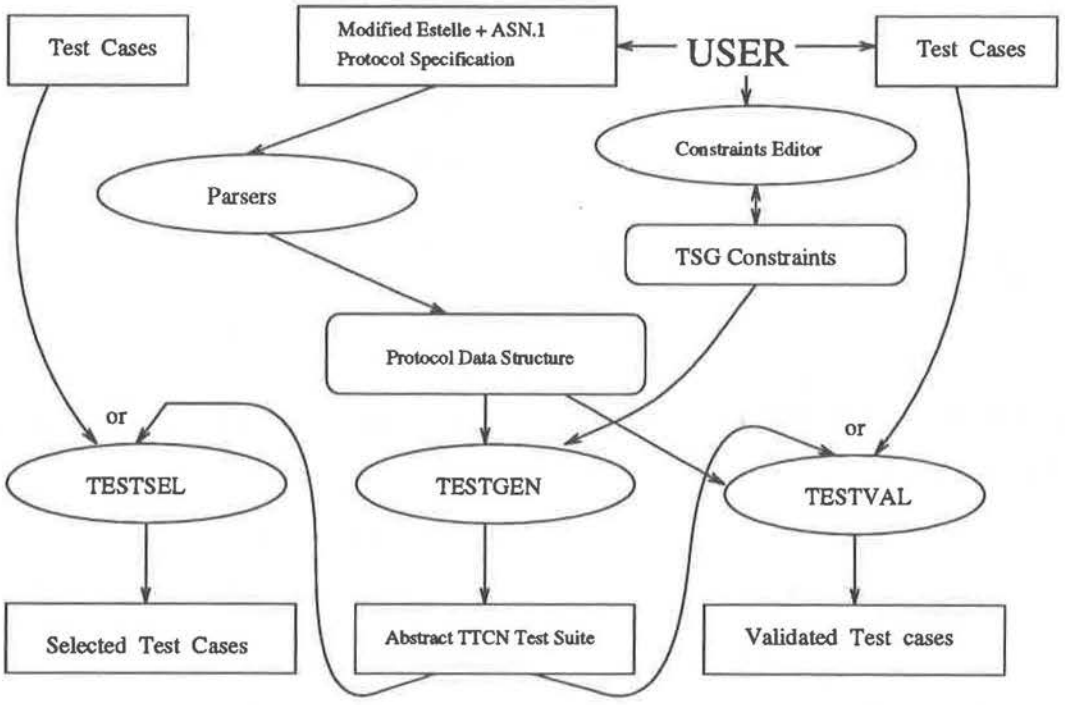

Figure 1: Overall Structure of TESTGEN+

TESTGEN, TESTSEL and TESTVAL for test generation, selection and validation,

respectively. The frontend modules of TESTGEN+ include the parsers which accept a protocol description in Estelle.Y (a slight variant of Estelle) and an ASN.1 description of the ISPs, OSPs, and PDUs. Via a constraint editor, the users can define the control flow constraints by specifying the minimum and maximum usages of protocol states, transitions, ISPs, OSPs, PDUs, internal variables or timers. Parameter variation constraints can also be defined for each field of the ISPs or PDUs where each parameter field is varied independently of the other fields to generate more suitable test suites. Once the PDS is produced and all the constraints are defined, the module TESTGEN will identify the subtours within the specification which satisfy the given usage constraints. Each subtour undergoes the parameter variation as defined by the types of service primitives in the subtour and by the parameter variation constraints. After all unique subtours have been detected, TESTGEN returns to its main menu interface and allows the user to alter the previous constraints to (re)generate a modified test suite, or to select the other functions such as TESTSEL or TESTVAL.

The second module TESTSEL is based on the concept of testing distance and coverage metric and performs test selection from the subtours generated by TESTGEN. The metric **dt** defined in Definition 8 captures the main sources of protocol specification complexity, namely recursion and parallelism, and forms the basis of the multipass selection algorithm [McA-92]. TESTSEL accepts as input the control sequence of an arbitrarily large test suite. The user can then define some constraints to be used in the test selection algorithm: the minimum target distance (ϵ_{min}), the scale value for each pass, and the maximum cost (maxCost). The user can also define the pattern and weight function, p_k and r_k respectively according to some external expert's knowledge. TESTSEL selects test cases which satisfy the given constraints from the subtours generated by TESTGEN. Thus, this module can run concurrently with TESTGEN or can be used independently of the other modules.

The last module TESTVAL checks whether a given test case is valid with respect to a given specification. TESTVAL makes use of the PDS produced by the frontend parsers to perform test case validation for a given stream of test cases. It also produces

some helpful information for diagnosing errors in invalid test cases.

4 TESTGEN

Depending on the constraints specified, TESTGEN can generate an arbitrary number of subtours which satisfy an arbitrary number of specified conditions on the external behavior of a protocol implementation. A separate test case is generated for each subtour. TESTGEN consists of two major functional modules: the test generation module and the output module. TESTGEN also makes use of the PDS generated by the frontend parsers and the constraints produced by the constraint editor. The PDS and these relevant modules are described in this section. The detail components of TESTGEN are depicted in Figure 2. A formal Estelle.Y and ASN.1 protocol specification are parsed to generate a PDS for the internal representation of the extended transition system. The TSG and parameter variation constraints are set to default values that can be interactively modified by the user with the constraints editor. Based on the PDS and the TSG constraints the test suite generation engine identifies all the subtours of the protocol graph that fulfill the TSG constraints and generates three kinds of outputs : first is an abstract TTCN test case for each identified subtour, second is a local formal form for test case validation and the last is a graphical form for the user.

4.1 Protocol Data Structure(PDS)

A protocol data structure representing a complete real-world communication protocol, e.g. X.25, OSI transport or session layer, is very complex. In order to keep the data structure manageable we adopt a structure similar to the object descriptor access concept [Lu-91]. Each PDS element (e.g. a state or a constant) is allocated its own data structure. The pointer to this specific data structure is stored in a two dimensional central PDS structure. This pointer can be retrieved via the PDS element's *type* (STATE, VAR, ISP,...) and *key* (type specific array index). Each PDS element is thus uniquely referenced by its *type* and its *key*.

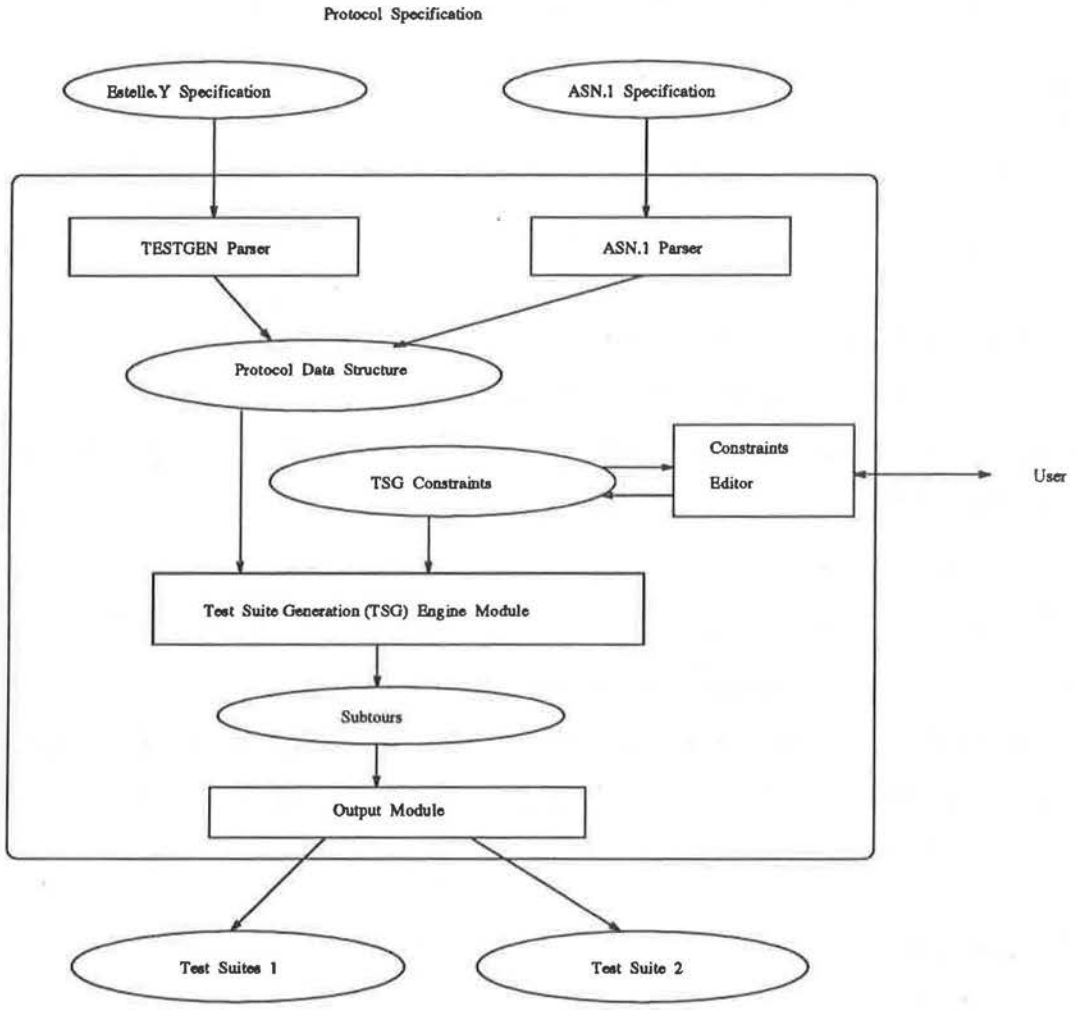

Figure 2: Structure of TESTGEN

The data types used to represent states $\in$ STATES, variables $\in$ VAR, timer $\in$ TIMER and constants $\in$ C are mapped from the PDS definition.

The ISP, OSP and PDU data types are stored as the parameter structures in ASN.1 enode trees [Sam-90].

The data type representing transitions $\in$ T contains the keys to the following elements:

- the state from which the transition can be fired,
- the state to which the transition is leading,

- the priority of the transition (set to default if not defined),
- the ISP, OSP(s) and input and output PDU(s) associated with the transition,
- the expression representing the enabling predicate of the transition,
- the action function representing the action function.

Expressions and action functions are represented with a recursive data structure by using the following additional data types: TEXPR for timer expressions, CSTMT for compound statements, IFSTMT for if statements, ASTMT for assignment statements and TSTMT for timer statements. The detailed definition of the PDS representation can be found in [Lu-91].

Since memory use is not an issue for this static data structure the PDS representation is designed to be slightly redundant. The advantages of this redundancy are twofold. First, this scheme guaranties easy access to most PDS elements (by reducing the number of pointer indirect references). Second, some static consistency conditions on PDS are defined and can be verified for each generated data structure.

4.2 Parser

The parser takes into account both the control and data flows of formal protocol specifications. This module parses the Estelle.Y/ASN.1 specifications and translate them into the PDS of the ETS. The ASN.1 parser [Sam-90] is used to parse the ASN.1 specification of PDUs and SPs into ASN.1 type trees. The generated ASN.1 type trees are linked to the PDS when the Estelle.Y specification is being parsed. As a result, the PDS includes both Estelle.Y and ASN.1 protocol information generated from an Estelle.Y/ ASN.1 specification. Estelle.Y uses an ETS to model the observable behaviors of a protocol, and ASN.1 is used to model the data representations(PDUs and SPs) of the protocol. The main differences between Estelle.Y and the normal form specification(NFS) which is used in many test generation methods based on Estelle , are that Estelle.Y uses ASN.1 and supports more Pascal statements such as

the conditional and loop statements as well as some timer primitives in the ASN.1 (TTCN) style. The structure of ISPs, OSPs and PDUs in Estelle.Y are specified in ASN.1. The ASN.1 parser was developed using the UNIX Lex /Yacc tools, and produces an ASN.1 type tree from an ASN.1 specification. The PDS is designed to be a machine accessible form of the ETS/ASN.1-based formal description. It holds information of both the control and the data flows of a protocol specification. The ISP, OSP and PDU data types are stored in the data structures of the ASN.1 type trees.

4.3 Constraints Editor

The constraints editor module provides the user with an interactive screen-oriented interface for defining of constraints. The user can modify the values of the "*min_*" and "*max_*" default constraints values as well as the parameter values of the SPs or PDUs to be sent to the IUT. From one to ten parameter values can be assigned to each parameter of a SP or PDU. It should be noted that the term "constraint" we use here has a different meaning from the one used in TTCN. In TESTGEN, a constraint is a boolean predicate that has to hold for each subtour. A subtour is generated for each branch of the behavior generated by the test generation (TSG) engine which will be described in Section 4.4, for which all the constraints are fulfilled. Constraints have two types : one is the max and min usage conditions for all ETS elements, states, transitions, ISP, OSP, PDU, constants, variables, and timers; the other is the parameter variations which define a set of values for each parameter of each ISP or PDU.

4.4 Test Suite Generation Engine

This is the main module of TESTGEN; it identifies and generates all the subtours based on the PDS obtained from a given specification and a set of constraints. The subtour identification algorithm performs an exhaustive depth first search over the behavior tree representation of the protocol. A test-branch of this tree is said to

be *valid* if and only if the subtour associated with this branch fulfills all the defined constraints. A global constant "*MAXUSE*" limits the value range of the *max_reached* and *max_used* constraints. These constraints limit the length of each valid test-branch. In each state only a finite number of transitions can be applied according to the protocol specification. The parameter variation constraints on the parameter of the service primitives exchanged in each transition limits the number of different instances of the service primitive in that transition. Thus the length and the number of different valid test-branches(subtours) are kept finite so that the backtracking algorithm is guaranteed to terminate.

The test suite generation algorithm is initially called as :

$$Generate\_subtours(ETS, C_{min}, C_{max}, q_{init}, u_0, q_{init}),$$

where C_{min} is a vector of all the minimal constraints, C_{max} is a vector of all the maximal constraints, q denotes the current state, u denotes the vector representing the current record of use of all elements so far for the current subtour, u_0 means the vector of zeros representing the initial use of all elements, and ST denotes subtour. CV is a set of all parameter value combinations allowed by the parameter variation constraints.

The test suite generation algorithm is succinctly given as follows:

Test Suite Generation Algorithm

$Generate\_subtours(ETS, C_{min}, C_{max}, q, u, ST)$:=

```
Find transitions in the ETS that can possibly be fired at state q ;
For each such transition t do
    if event of transition t is an ISP or IPDU
        For each ISP or IPDU parameter value combination v ∈ CV do
            if Enabling_predicate (t,v) is true
              {
              q' ← Apply action function to state q and event (IPDU or ISP);
              u' ← u + Use_of_elements(action function of transition t);
              ST' ← ST + {(t,v)};
              if u' ≤ C_max
```

```
                {
                if Major_state(q') = Major_state(q_init)
                  if u' ≥ C_min
                    Output_subtour(ST');
                Generate_subtours(ETS, C_min, C_max, q', u', ST');
                }
              }
        enddo
    else /* spontaneous transition with a PROVIDED clause and no WHEN clause */
        if Enabling_predicate (t) is true
            {
            q' ← Apply action function to state q;
            u' ← u + Use_of_elements(action function of transition);
            ST' ← ST + {(t, q')};
            if  u' ≤ C_max
              if Major_state(q') = Major_state(q_init)
                if u' ≥ C_min
                  Output_subtour(ST') ;
              Generate_subtours(ETS, C_min, C_max, q', u', ST');
            }
enddo
```

4.5 Output Module

The output module produces the result of the test generation. There are four subtour output formats. The first is a graphic form for visuality. The second contains only the control parts which can be fed into TESTSEL for test selection. The third form is a local formalism which can also be used in TESTVAL for test validation. The last form is TTCN.MP for convenient interface with a TTCN workbench or other TTCN.MP support tools.

5 TESTSEL

TESTSEL has the function to select test cases from a test suite such that the test cases form an approximation to the test suite within a specified degree of coverage tolerance. It can also accept a test suite generated from TESTGEN or some other test suite specified in the appropriate input format. TESTSEL consists of three major

functional modules. The detail structure of TESTSEL is shown in Figure 3. Test suite 1 and 2 in Figure 3 have same form and contents. A test suite to be selected is filtered to generate uniqued test suite which have only control parts and are unique each other. User can set and modify the parameter values for his selection purpose. Using the uniqued test suite and parameter value list, selection module selects test suite and calculates test coverage. If the test coverage is satisfied by user, he can run merge module to recover all original test cases which have both the control part and data part and are within the test coverage.

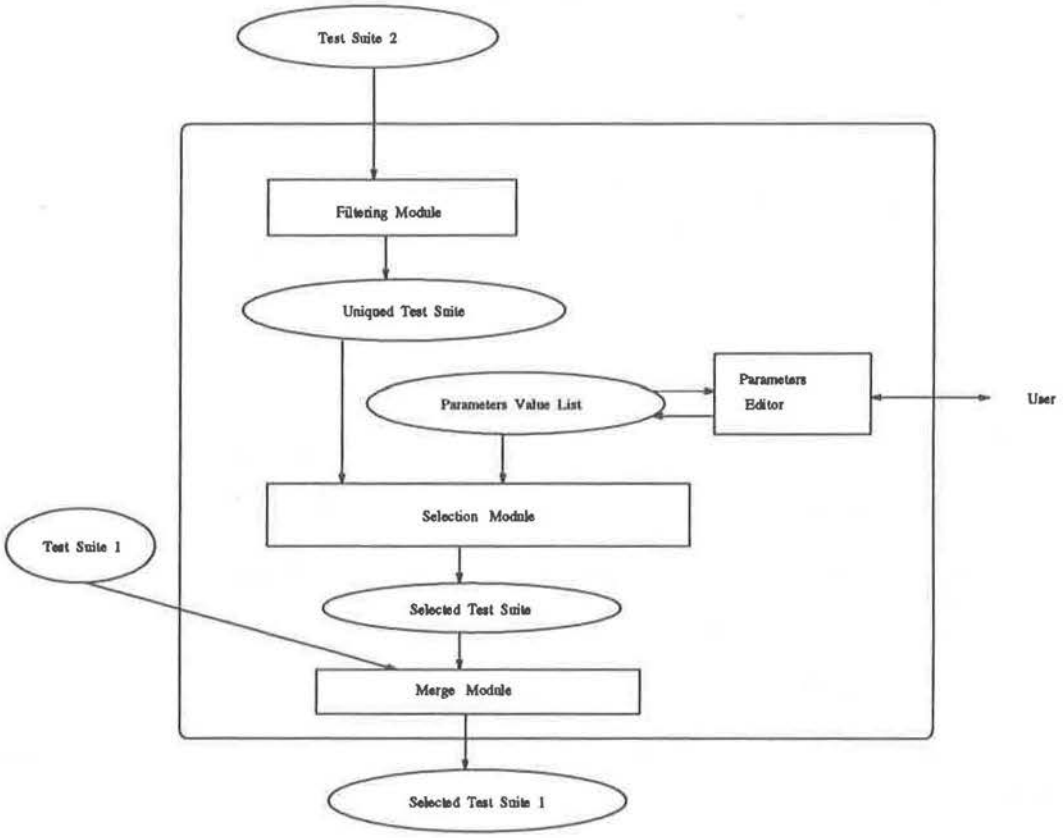

Figure 3: Structure of TESTSEL

5.1 Filtering Module

TESTSEL performs test selection based only on the control flow information of protocols. Thus, the test cases (which contain both the control and data flow information)

produced by TESTGEN must be fed to a filtering module which strips off the data part and produces only the control sequences. We use recursion depth α (or β) for the abstract representation of a control test case as shown in Definitions 7 and 8. This module maps many data variant instances of a (control) test case into a single control test case and produces an internal representation for each control test case to be processed by the selection submodule.

5.2 Selection Module

The selection module selects test cases from an internal form of a given test suite (produced by the filtering module) using some user defined constraints. These constraints include the testing distance definition parameters p_k and r_k, and the test selection constraints: the maximum cost and the initial and minimum coverage tolerances ϵ and ϵ_{min} (test selection radius), as well as the radius scale factor for each pass. The multipass algorithm selects test cases to maximize coverage subject to the cost constraint. Since it is proven that all Cauchy sequences converge in the metric space and since successive passes of the algorithm with successively narrower test selection radius will produce such sequences, the test selection algorithm is guaranteed to yield a set of test cases which converge to the initial test suite as more test cases are selected [McA-92]. This guarantees that no specific (subsets of) test cases are missed out due to mere overlooking, a problem commonly encountered in a manual test selection process.

The algorithm begins with an empty set of selected test cases TC. For each control sequence in the test suite, we compute its minimum distance to the sequences already present in the set of selected test cases. The control sequence is added to set TC if this minimum distance is greater than the test selection radius ϵ. Once all the sequence in the test suite have been examined we shrink the test selection radius ϵ by a user-defined scale factor and reiterate the algorithm without emptying the set TC. Thus, each pass of the algorithm guarantees that the nonselected test cases are at most a distance ϵ from the selected ones. The reiteration continues until we

reach some minimum test selection radius ϵ_{min} or the execution cost of TC exceeds a user-defined maximum cost value. Some α-pruning techniques provide an efficiency gain when the distance of a control sequence to a set TC is computed. In most cases we are mainly concerned with whether or not this distance is above or below some value ϵ. Once the distance is below ϵ, the search through the remaining test cases can be avoided since the distance can only decrease from this point onward. The test selection algorithm is succinctly given as follows:

Test selection algorithm

$Test\_select(TS, \epsilon, \epsilon_{min}, scale, maxCost, TC) :=$

```
TC ← { };
curCost ← 0;
overflow_flag ← False;
while (ε ≤ ε_min) and (overflow_flag = False) do
        reset TS;
        while (unprocessed test case remainded in TS) do
                tc ← consider a test case in TS randomly;
                dist ← Min_{tc∈TC} { dt (tc,TC) };
                if (dist > ε)
                    if (Cost(tc) + curCost < maxCost)
                            {
                            TC ← TC ∪ { tc };
                            curCost ← curCost + Cost(tc);
                            }
                    esle
                            overflow_flag ← True;
                Mark tc as processed;
        enddo
        ε ← ε × scale;
enddo
Calculate test coverage according to Definition 9;
```

5.3 Merge Module

The merge module reproduces the selected test cases in the original form which has both control flow and data flow information, i.e. all of their original instances as

generated by TESTGEN. This is possible since the (selected) control test cases have pointers to all of their instances. Using pointer information, this module merges the original test suite and the selected control test cases to produce all of the final selected test cases which contain both the control and data flow information.

6 TESTVAL

TESTVAL allows the checking whether a given test case is valid with respect to a given Estelle specification. TESTVAL makes use of the PDS generated by the frontend parsers from a protocol specification in Estelle.Y and ASN.1. TESTVAL consists of three major functional modules: the preprocessor, the validation module and the output module. The detail structure of TESTVAL is depicted in Figure 4. The TESTGEN parser which accepts Estelle.Y and ASN.1 specification is exactly same as parser of TESTGEN. The preprocessor translates some transitions into more suitable forms for the fast and efficient processing of test case validation. The main module starts in the initial state for each input test case and can handle two types of parameter values such as integer and boolean. There are data structure contains information such as a number of transitions and next state of each transition with respect to the same current state. Also the current pointer of test cases being processed is kept in order to the right position in the input test case.

6.1 Preprocessor

The preprocessor consists of two submodules and converts certain types of transitions in the specification into a more suitable form for the main processing. Its function is to improve the performance and efficiency of the main module.

It produces a list containing ISP (or OSP) parameters and their types using an ASN.1 type tree generated by the ASN.1 parser. The ISP (or OSP) parameters in the list are assigned values according to the Estelle specification. It also translates the EFSM form of an Estelle.Y specification into a simple FSM form [Kim-92]. Instead of enumerating all possible combinations of service primitives and their parameters, some

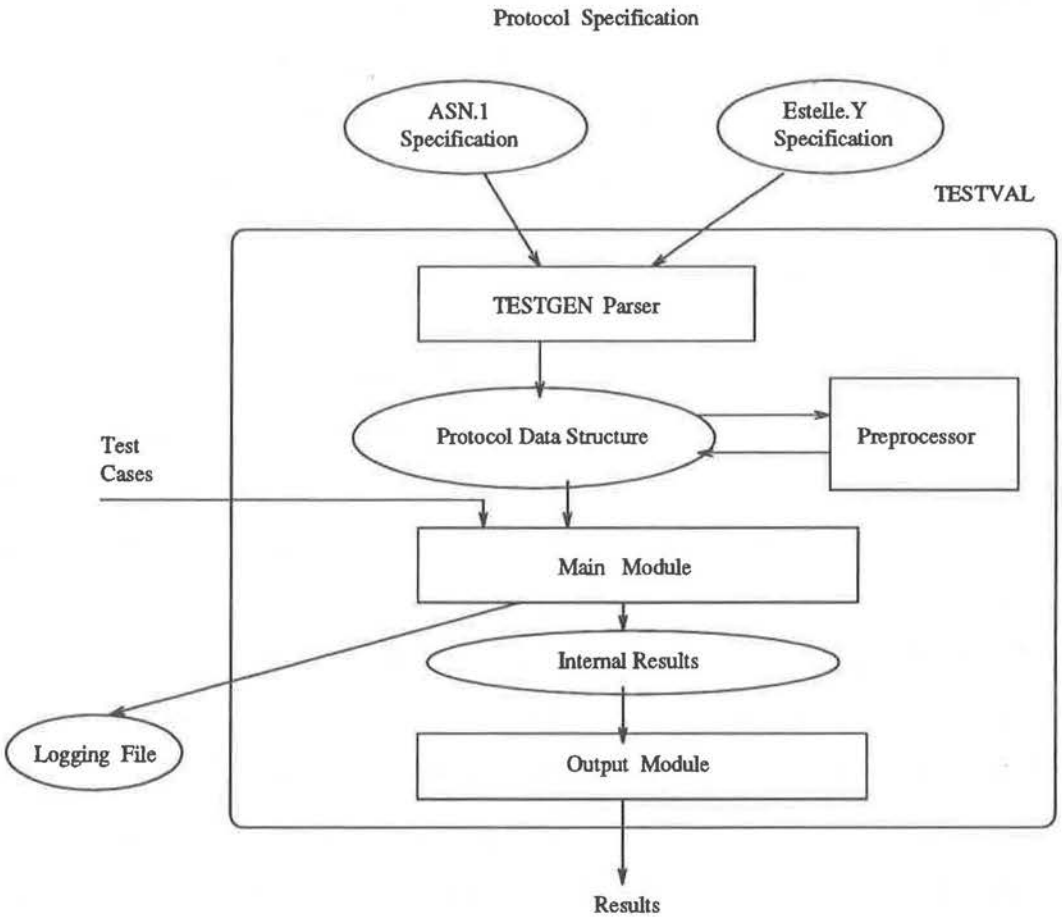

Figure 4: Structure of TESTVAL

parts of the enumeration are realized based on the cost-effectiveness consideration. The "PROVIDED" clause of a transition corresponds to the input condition of the transition. If the clause consists of ISP parameters with values and operators such as "EQUAL" and "AND", it is considered to be a list of input symbols of a transition of the FSM.

In order to make output symbols of a transition of the FSM correspond to the ones of the EFSM, the assignment statements are checked to select those consisting of OSP parameters with values.

6.2 Validation Module

The validation module validates given test cases, TS with respect to the PDS of the protocol specification. To achieve this goal, it makes use of some data structures. "Paths" contains information such as the number of transitions, "from_state" and "to_state" for each transition in the path. "From_states" and "to_states" are implemented by array and contain the state information of the system, via the values of related variables and the information on all candidate transitions at these states. The pointer "no_proc" keeps track of the processing position of the event in the test case. For each candidate transition, the values of variables and service primitives from the test case are substituted to the symbolic representation of the PDS. If the predicate and the output primitives of a transition are satisfied in the test case, then the test case is valid according to the formal specification. The algorithm in the validation module is given as follows:

Test validation Algorithm

$Test\_validate(ETS, TS, from\_state, Paths, no\_proc)$:=

```
Point to first test case;
while (not Eof(TS)) do
        Read one test case into buffer;
        Initialize Paths with from_state as initial state;
        Set variables using the PDS;
        Find a set of executable transitions {tr} at initial state;
        while ( ∃ events in test case ) do
                Set no_proc to 1;
                while (∃ unprocessed path) do
                        while ( ∃ unprocessed tr at from_state) do
                                Call FIRE(tr, from_state, to_state);
                                if (firable)
                                  Increment no_proc;
                        enddo
                enddo
        enddo
        new Paths ← current Paths;
        if (∃ no tr in the new Paths)
```

```
            if (no_proc < | test case |)
                    Print "test case is invalid";
            else
                    Print "test case is valid";
        Read next test case into buffer;
enddo
```

The initial value of the "from_state" is an initial state of given protocol specification. The function FIRE assigns values of parameters to the transition, sets "to_state" by target state, copies values of variables at the "from_state" to the "to_state", and searches all firable transitions and saves it in "Paths". The FIRE returns "True" if it finds firable transitions at the given "from_state".

6.3 Output Module

TESTVAL logs the traces of states and transitions which satisfy the given test cases in a file. If a test case is valid according to the given formal specification, TESTVAL generates a message "test case valid" displayed on the terminal and in the log file. Otherwise, if a test case is not valid, the position of the event in error is logged. Using the information logged in the file and messages on the terminal, possible errors in the test case with respect to the formal specification may be located. Whether the test case conforms to the formal specification or not, TESTVAL keeps on processing until there is no test case left. The logged information shows which transitions have been executed. In some cases, more than one path of the transitions may satisfy the test case as a result of nondeterminism. In that case we can follow those paths using "transition keys" given in the file. Every transition has a unique "transition key" identifying its location in the source of the Estelle specification. Therefore these logged information could be helpful in locating and diagnosing the errors occurred.

7 How to use TESTGEN+

In this section, we briefly explain how to use our TESTGEN+. Basically TESTGEN+ were developed based on the user friendly interface mechanisms. Therefore user only choose the number or character from menu with respect to his running purposeand input some proper values for certain constraints or parameters. There are three major functions in TESTGEN+ : test case generation, selection and validation. With the assumption of using same input formats there are many practical ways to use TESTGEN+. The followings are typical ways.

- Test case generation.
- Test case selection with already generated test cases.
- Test case validation.
- Test case generation/validation.
- Test case generation/selection.
- Test case selection/validation.
- Test case generation/selection/validation.

8 Experiments

In this section, we discuss the experiments performed on TESTGEN+ using the InRes protocol [Hog-91], the OSI class 0 transport protocol (TP0) [Vuo-93] and the X.25 LAPB protocol [Zho-92]. The overall major attributes of the above protocol specifications are summarized in Table 1 and the experiments were conducted on a SUN 4/690 workstation running under UNIX.

Kinds	InRes	TP0	LAPB
No. of States	3	4	6
No. of Transitions	21	21	132
No. of ISPs	6	6	5
No. of OSPs	5	5	6
No. of PDUs	5	5	1
No. of Timers	0	1	2
No. of Variables	2	8	20
No. of Constants	0	5	6

Table 1. Summary of Protocols Attributes

The InRes protocol consists of two parts: the initiator and the responder. For simplicity, we deal with only the responder in our experiments. There are ten frame types in LAPB protocol and these are represented by one parameter (frametype) in our specification.

8.1 InRes Protocol

Since the InRes protocol is simple, we use it as the first protocol example from which to generate, select, and validate test suite. The major min and max constraints values for the InRes protocol to generate test cases in our TESTGEN experimentation is as follows:

1. State
   Disconnected (0/1), Wait (0/6), Connected (0/8)

2. Transitions
   For all transitions (0/99)

3. ISP
   For all ISP (0/1)

4. OSP
   For all OSP (0/1)

5. PDU
   For all PDU (0/1)

The followings are the major parameter variation constraints for ISP or PDU:

1. DT PDU
   seqNo (0,1)
   info (any value)

2. AK PDU
   seqNo (0,1)

For test selection experiment, we choose the parameter values of the initial and minimum test selection radii ϵ, ϵ_{min}, scale factor and maxCost as 1.8, 0.1, 0.5 and 60, respectively. A linear cost function is assumed.

8.2 TP0 Protocol

We use the TP0 protocol as the second protocol example from which to generate, select, and validate test suite. The major min and max constraints values for the TP0 protocol to generate test cases in our experimentation is as follows:

1. State
   idle (0/2), wftr (0/2), wfcc (0/2), data (0/2)

2. Transitions
   For all transitions (0/99)

3. ISP
   For all ISP (0/2)

4. OSP
   For all OSP (0/2)

5. PDU
   For all PDU (0/2)

We also define the parameter variation constraints. The followings are the major parameter variation constraints for ISP or PDU:

1. TCRES ISP

qtsReq (11,27)

2. CR PDU
   option (-100)

For test selection experiment, we choose the parameter values of the initial and miminum test selection radii ϵ, ϵ_{min}, scale factor and maxCost as 1.2, 0.05, 0.5 and 270, respectively.

8.3 LAPB Protocol

We use the LAPB protocol as the last example. It is much more complex than the InRes and TP0 protocols. The Estelle.Y LAPB protocol specification has 6 states, the ABMONE state was added to handle the while loop of the original LAPB protocol specification. The major min and max constraints values for the LAPB protocol to generate test cases in our experimentation is as follows:

1. State
   SEND_DM (0/3), SEND_SABM (0/3), ABM (0/5),
   ABMONE (0/3), WAIT_SABM (0/4), SEND_DISC (0/2)

2. Transitions
   For all transitions (0/99)

3. ISP
   For all ISP (0/1) except DataIndicat (0/3)

4. OSP
   For all OSP (0/1) except DataRequest (0/3)

5. PDU
   For all PDU (0/1)

The followings are the major parameter variation constraints for ISP or PDU:

1. DataIndicat ISP
   frametype (0,1,2,3,4,5,6,7,8,9)
   address (0)

pf (0,1)
nr (0)
ns (0)

2. DataIndicat OSP
   frametype (0,1,2,3,4,5,6,7,8,9)
   address (0)
   pf (0,1)
   nr (0)
   ns (0)

The LAPB specification can deal with addresses (0,1), the nr and ns (modulo 8 system), however we simplify the parameter variation constraints to focus on the control flow. For test selection experiment, we choose the parameter values of the initial and minimum test selection radii ϵ, ϵ_{min}, scale factor and maxCost as 1.2, 0.1, 0.5 and 600, respectively.

8.4 Summary

The whole experiments performed with TESTGEN+ are summarized in Table 2. The number of generated test cases are dependent on the min and max constraints values specified by user. The coverage measures express how much the selected test cases cover the initial set of control test cases, and the measures vary with the parameters ϵ, ϵ_{min}, and maxCost. For the InRes protocol, the 27 selected test cases cover the initial 92 generated test cases with a 0.890625 coverage. For the LAPB protocol, 101 (control) test cases selected out of 566 (control) test cases would correspond to a coverage of 0.9375. After merging, this leads to 1001 final test cases chosen out of 19,546 original test cases. This clearly indicates that the metric guided test selection is particularly important when applying to very large test suites. Using TESTVAL, Table 2 shows that test cases for the three protocols generated by TESTGEN have 100 percent of validity with respect to their specification, and this means the validity of TESTGEN and TESTVAL checks out against each other, as expected since both TESTGEN and TESTVAL are based on the same specification (or PDS). In summary,

from our experiments we find TESTGEN+ is a useful and practical tool for protocol test generation, selection, and validation.

Kinds	InRes	TP0	LAPB
No. of Generated Test Cases	137	375	19546
No. of Control Test Cases	92	252	566
No. of Selected Test Cases	27	77	101
Value of Test Coverage	0.890625	0.968750	0.937500
No. of Merged Test Cases	39	113	1001
No. of Valid Test Cases	137	375	19546

Table 2. Summary of experimental results

9 Conclusions

This paper presents and discusses TESTGEN+, an integrated environment for test suite generation, selection, and validation. These functions are implemented in three modules: TESTGEN, TESTSEL and TESTVAL respectively. TESTGEN adopts the TSG-constraints based test suite generation method which integrates the generation and selection of abstract TTCN test suites for formally specified communication protocols. The TSG constraints approach offers a flexible mechanism for generating conformance as well as special purpose test suites for real life protocols. TESTGEN thus serves as a useful test-bed for experimenting with test generation. TESTSEL adopts a metric guided multipass test selection algorithm which guarantees the set of selected test cases approaches the original set as more test cases are selected with no particular cases being merely overlooked as typically happened in manual test selection process. The inclusion of TESTSEL allows the user to assess and select from the generated test cases, as well as providing the information necessary to tune the TSG constraints for better results. We already demonstrated the validity of TESTVAL by applying real life test cases having both the control flow and the data flow information, taken from the LAPB standard test suite [Vuo-93b]. TESTVAL provides a means to ensure the test suites developed are valid with respect to the specification.
The current prototype of TESTGEN+ has been applied to a number of protocols, including the InRes protocol, the OSI class 0 Transport protocol and the LAPB protocol. We present and discuss the results of the test generation, selection, and validation experiments on the above protocols. Automatic test selection as provided by TESTSEL is particularly important when we have to select a manageably small set of test cases with a sufficiently good coverage from a cost-prohibitively large test suite. Using TESTVAL and TESTGEN, we have shown the validity and consistency of both of these tools. In general, we find TESTGEN+ performs well as a useful, general, flexible, and semi-automated tool for test suite generation, selection, and validation in conformance testing. Some aspects of TESTGEN+ can be improved. A major

extension of TESTGEN+ will be the handling of multi-module Estelle specifications. Other possible extensions include enhancement on the TSG engine, TSG constraints, user interface, and interface with TTCN tools, as well as further experimentations with practical real-life protocols.

References

[Cha-89] W. Y. L. Chan, S. T. Vuong, and M. R. Ito, *An Improved Protocol Test Generation Procedure Based on UIOs,* Proceedings of the ACM SIGCOMM '89 Symposium on Communication Architectures and Protocols, September 1989.

[Cho-78] T. S. Chow, *Testing Software Design Modeled by Finite State Machines,* IEEE Transactions on Software Engineering, Vol. 4, No. 3, pp. 178-187, March 1978.

[Gon-70] G. Gonenc, *A Method for the Design of Fault Detection Experiments,* IEEE Transactions on Computers, Vol. 19, No. 6, pp. 551-558, June 1970.

[Fuj-91] S. Fujiwara, G. v. Bochmann, and et al, *Test Selection Based on Finite State Models,* IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 591-603, June 1991.

[Hog-91] D. Hogrefe, *OSI Formal Specification Case Study : The InRes Protocol and Service,* Technical Report IAM-91-012, Universitaet Bern, Institut fuer Informatik und Angewandte Mathematik, Bern, Switzerland, May 1991.

[Hsi-71] E. P. Hsieh, *Checking Experiment for Sequential Machine,* IEEE Transactions on Computers, Vol. C-20, No.10, October 1971.

[ISO-a] *Information Technology - OSI Conformance Testing Methodology and Framework,* Draft International Standard, ISO/IEC DIS 9646 (5 Parts).

[ISO-b] *Information Technology - OSI Conformance Testing Methodology and Framework,* Part 3: The Tree and Tabular Combined Notation, ISO/IEC DIS 9646-3, 1990.

[Kel-76] R. M. Keller, *Formal Verification of Parallel programs,* Communications of the ACM Vol. 19, No. 7, pp. 371-384, July 1976.

[Kim-92] Myung Chul Kim, *Trace Analysis of Protocols based on Formal Concurrent Specifications,* Ph.D Thesis, Dept. of Computer Science, Uinversity of British Columbia, 1992.

[Lu-91] Y. Lu, *On TESTGEN, An Environment for Protocol Test Sequence Generation, And Its Application to the FDDI MAC Protocol,* M.Sc. Thesis, Dept. of Computer Science, University of British Columbia, 1991.

[McA-92] M. McAllister, S. T. Vuong and J. Curgus, *Automated Test Case Selection Based on Test Coverage Metrics,* Proceeding of International Workshop on Protocol Test Systems, Montreal, Canada, September 1992.

[Nai-81] S. Naito and M. Tsunoyama, *Fault Detection for Sequential Machines by Transition Tours,* Proceedings of the 11th IEEE Fault-Tolerant Computing Symposium. pp. 138-243, June 1981.

[Sam-90] M. Sample and G. Neufeld, *Support for ASN.1 within a Protocol Testing Environment,* The Third International Conference on Formal Description Techniques (FORTE '90), Madrid, Spain, November 1990.

[Sab-88] K.K. Sabnani and A. T. Dahbura, *A Protocol Test Generation Procedure,* Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, September 1988.

[Sar-87] B. Sarikaya, G. v. Bochmann, and et al, *A Test Design Methodology for Protocol Testing,* IEEE Transactions on Software Engineering, Vol. 13, No. 5, pp. 518-531, May 1987.

[Sid-89] D. P. Sidhu and T. -K. Leung, *Formal Methods for Protocol Testing: A Detailed Study,* IEEE Transactions on Software Engineering, Vol. 15, No. 4, pp. 413-426, April 1989.

[Ura-88] H. Ural, B. Yang, and R. L. Probert, *A Test Sequence Selection Method for Protocols Specified in Estelle,* Technical Report TR-88-18, Department of Computer Science, University of Ottawa, June 1988.

[Vuo-89] S. T. Vuong, W. Y. L. Chan, and M. R. Ito, *The UIOv-Method for Protocol Test Sequence Generation,* Proceedings of the Second International Workshop on Protocol Test Systems, Berlin, Germany, October 1989.

[Vuo-91] S. T. Vuong and J. Curgus, *Test Coverage Metrics for Communication Protocols,* Invited paper, Proceedings of IWPTS IV - International Workshop on Protocol Test Systems, Leischendam, Netherlands, 1991.

[Vuo-93] S. T. Vuong, Y. Lu, C. Mathieson, and B. Do, *TESTGEN : An Environment for Test Suite Generation and Selection,* To appear in the Computer Communications Journal, 1993.

[Vuo-93b] S. T. Vuong, Sangho Lee, and Peter Zhou, *Protocol Test Validation : Principles, Tools, and Examples* , Invited paper, Proceeding of CFIP'93 - Colloque Francophone sur L'Ingénierie des Protocoles, Montreal, Canada, September 1993.

[Zho-92] P. Zhou, *On TESTGEN+, An Environment for protocol Test Generation and Validation,* Master Thesis, Dept. of Computer Science, University of British Columbia, 1992.