Test Case Generation of a Protocol by a Fault Coverage Analysis*

Tae-hyong Kim[1], Ik-soon Hwang[2], *Min-seok Jang, **Sung-won Kang, Jai-yong Lee[3], Sang-bae Lee
NAS Lab., Dept. of Electronics, Yonsei Univ.
*Dept. of Computer Science, Kunsan National Univ.
** Testing Technology Research Section, Korea Telecom R&D Group
thkim@nasla.yonsei.ac.kr[1] his@nasla.yonsei.ac.kr[2] jyl@nasla.yonsei.ac.kr[3]

Abstract

In this paper, we generate conformance test cases for a communication protocol modeled in an EFSM(Extended Finite State Machine) by a fault coverage analysis. For the analysis model, we choose the expanded EFSM to resolve the inter-dependency problem between control and data flows within an EFSM. An expanded EFSM has several useful properties and makes it easy to generate test cases. For test case generation, at first we define data elements in the expanded EFSM. With the definition, we define some probable fault models in edges of the expanded EFSM and discuss what test cases to be needed for satisfying each fault model. The analysis shows that control flow test cases with full fault coverage and data flow test cases satisfying 'all-du-paths' criterion are needed to guarantee high fault coverage in the expanded EFSM. A mass of generated test cases by high fault coverage is optimized through some steps. The result of a simple protocol shows the efficacy of this method.

1. Introduction

Lately protocol models used in generating conformance test cases are being changed from the FSM(Finite State Machine) model to the EFSM(Extended Finite State Machine) model, because the FSM model cannot fully specify communication protocols. Not considering data flows, control flow test using FSM model has also limited applicability, cannot detect any errors of data flows, and moreover may be impractical for not considering executabilities of test sequences. The EFSM testing has both the control flow test part and the data flow test part. Its control flow test usually adopts the test methods used in FSM testing. In the data flow test part, the test sequences have been mostly used that satisfy one of Weyuker's data selection criteria[12] based on data flow analysis of software testing[11]. But control flows and data flows are dependent each other and especially a transition in control flows gets decided executable by their data set. Lately some methods considering them have been proposed: EUIO(Executable-UIO)-method which considers data flows for generation of executable test sequences for control flow test[9] and a unified test sequence generation method whose sequences can test both control and data flows of IUT(Implementation Under Test)[3]. But the relation between control flows and data flows of an EFSM model of a protocol interferes with generation of each test cases having the desired test coverage. Optimization of data flow test cases has not been tried for difficulties in the formalization of its fault coverage. Some researches trying to solve this problem have been studied, one of which expands the EFSM by freeing predicate statements[14]. To resolve the state explosion, this method assumes finite and countable domains of the variables that occur in predicate statements. From the expanded EFSM, test sequences can be generated using classic methods. But the expanded EFSM is not a pure FSM and is still an EFSM, so new test methods are needed to test fully it with considering fault coverage. New test case generation methods according to the fault coverage are also needed for the expanded EFSM.

In section 2, we define an expanded EFSM and explain its properties. Test case generation methods by fault coverage in the expanded EFSM are proposed in section 3. In section 4, we discuss an optimization scheme for both control and data flow test cases. Section 5 contains the empirical result of a real protocol and conclusions.

2. An Expanded EFSM and Its Properties

The test of an EFSM has been composed of the control flow test and the data flow test. Our concept of an EFSM is shown in Fig.1. M_1 shows an outline of conventional EFSM structure. Control flows and data flows are related each other in conditional transitions, or transitions with

* This research was supported by the Korea Telecom R&D Group in Korea

predicates. This property makes the test coverage of the tests somewhat vague. So we will transform it to the expanded EFSM to separate control flow and data flow by eliminating predicates like M_2. The property permeates both control flows and data flows, and the separated are functionally equivalent to the original. In addition, their independence makes it easy to generate their test cases. The generated expanded EFSM has large but finite states if all variables have a finite domain.

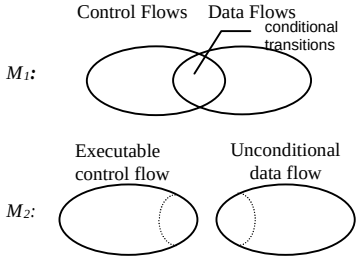

Fig.1 Transformation of EFSM structure

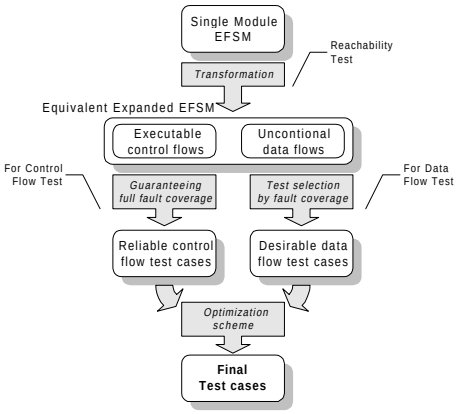

Fig.2 The test case generation procedure

Fig.2 shows our test case generation procedure for protocol testing. It starts from a single module EFSM. A communicating multi-module EFSM itself is nearly impossible to analyze and to test, so there has to be the simplification into the equivalent single EFSM, which has been a major issue in protocol testing. We already proposed a method to develop the lossless transformation by the reachability analysis in another paper[15]. Our target for the control flow test part is to generate test cases with full fault coverage, which guarantee the reliability of data flow test cases. For data flow test part, we also try to select test cases for the required fault coverage. The generated reliable control flow cases and desirable data flow test cases are optimized to the unified final test cases.

We, at first, define an expanded EFSM. They were defined in [14], but we modify them as follows.

Definition 1: A single module *expanded EFSM* is a 8-tuple $<\mathcal{S},v,p,V,S,I,O,T>$, where $\mathcal{S}$ is a general state set, v is a global and local variable set, p is a parameter set, V is a domain set of v, S is an extended state set which is $\mathcal{S}\times V$, I is a set of input declarations including the reset input *ri* and the delay input *D*, O is a set of output declarations including the null output, and T is a set of transitions which is $S\times I$ and is expressed as $<i/i(p:\mathcal{R}),M,o(e)>$, where $i\in I$, $o\in O$, $\mathcal{R}$ is an input range of p, e is an output parameter expression that is $f_o(v,p)$, M is an action block and is expressed as $v\leftarrow(v,p)$.

An expanded EFSM can be considered an reachability-tested EFSM and help the generation of test case of control flow test and data flow test for the following properties.

Property 1. All edges of an expanded EFSM are executable at any state.

Property 2. The values of variables at each edge in an expanded EFSM are identical regardless of incoming paths.

Property 3. The values of variables at each state in an expanded EFSM are identical regardless of incoming paths.

Property 4. An expanded EFSM has no self-loop which has actions of variable.

The added states and edges are related with the values of variables. The edges labeled a same second edge number is for faults of data flows and not for ones of control flows. In the aspect of data flows, we consider 'def', 'p-use', and 'c-use' of variables and parameters in the classic definitions. There are a def of the parameter in a parameter in an input declaration. The edges having a same input declaration but different values of the parameter in the input declarations are considered the different edges having different input declarations. Same as that is an use of the variables or the parameters in a parameter in an output declaration, 'o-use' to be defined in this paper. The edges having a same output declaration but different values of the parameter in the output declarations are considered the different edges having different output declarations. By a def of a variable in an assignment statement, 'd-def' to be defined in this paper, the value of the variable is changed. The def, 'd-def', is related with the state in an expanded EFSM. As discussed in the previous section, p-use of a variable or a parameter affect the transition in an expanded EFSM.

As shown above, data flows from defs to uses of variables or parameters are not clearly revealed in the expanded EFSM. But data elements in the original EFSM are permeated into the transformed expanded EFSM. The classic control flow test only is, therefore, insufficient for the test of an expanded EFSM.

3. Test Case Generation by Fault Coverage Analysis

In protocol testing and software testing, fault coverage of test cases has been a major issue. As there is a trade-off relation between fault coverage and the length of test cases, how to select test cases by considering the relation is the policy of a tester like the selection of test purposes. Analyzing fault coverage of test cases, we think, can eliminate the redundant part of the test cases and make clear their detectability and undetectability of the probable faults. Fault coverage of control flow test cases of a FSM-modeled protocol has been mainly studied, but rarely has that of data flow test cases of an EFSM-modeled protocol. The expanded EFSM will help the analysis of fault coverage.

We assume that the original EFSM representation of the protocol is deterministic and completely specified. Determinism and complete specification of an EFSM also mean that if there are predicates in an input declaration with parameters at a state, the regions satisfying the predicates are disjoint together and the union of the regions constructs the whole region.

3.1. Data elements in the expanded EFSM

In the expanded EFSM, to generate data flow test cases, we will use the Weyuker's criteria. As there is no p-use of a predicate statement in the expanded EFSM, we modify data flow elements of Weyuker's as follows[12].

The fault coverage of data flow test has been decided normally by using Weyuker's data selection criteria which use def-clear paths as test elements and define some test criteria by the test coverage of them[12]. We classify their bases, 'def' and 'use' more clearly and redefine the meaning of a def-clear path to fit for the protocol testing. The assumption of linearity would enable us to use matrix expressions.

Definition 2. An assignment, "$v \leftarrow (\psi, \underline{p})$" in an action block in an expanded EFSM contains an *i-def* (*independent-def*) of the variable v where $v \in \nu$ and $\psi \neq \underline{p} \subset p$. An assignment, "$v \leftarrow (\underline{v}, \underline{p})$" in an action block contains a *d-def* (*dependent-def*) of the variable v and a *u-uses* (*unobservable-uses*) of the variable $\underline{v}$ where $v \in \nu$, $\underline{v} \in \underline{\nu}$, $\psi \neq \underline{v} \subset \nu$, and $\psi \neq \underline{p} \subset p$. An output parameter expression $(\underline{v}, \underline{p})$ in an output declaration in the expanded EFSM contains an *o-use* (*observable-uses*) and an *i-def* (*independent-def*) of the variable $\underline{v}$, where $\underline{v} \in \underline{\nu}$, $\psi \neq \underline{v} \subset \nu$, and $\psi \neq \underline{p} \subset p$.

Definition 3. We redefine a *def-clear path* of a variable as a path which starts an edge having an i-def of the variable and ends other edge having an o-use of the variable without passing any edge having another i-def of the variable in the expanded EFSMs.

As Weyuker's test data selection criteria are based on software testing, its inclusion relation between criteria may not be valid for a protocol modeled in an EFSM. As an EFSM has many edges that do not have data flow element, the test cases satisfying 'all-du-path' criterion may not have all edges in the EFSM. Normally the probability of faults is thought to be almost equal, so untested edges reveal ineffectiveness of the test method. We, here, take the minimal and necessary assumption that all the faults can lie only in the edges constructing def-clear paths, which is acceptable to almost EFSM. By the above discussion, all the edges constructing any of def-clear paths must be tested. Therefore we have to select the all-du-path criterion for data flow test.

For control flow test, we will use the UIO(Unique Input Output) method that has been widely used for detecting faults in control flows. We will generate the test cases for the state identification part and the transition identification part for the full fault coverage[2].

3.2. Fault models in data elements of the expanded EFSM

In the general EFSM, faults of variables or parameters in data flows may exist in the following parts: input declarations with parameters, predicates, action blocks, and output declarations with parameters. Two kinds of faults exist. One is the modification fault that the existing part is wrongly modified. The other is the addition fault that the unspecified part is wrongly specified. Each kind of faults may occur alone or both of two may occur together. We assume that there is no fault in edges added only for completely specified condition for simplification. We classify the probable faults as follows.

Type I: The faults of the assignments, $v^* \leftarrow (\underline{v}, \psi)$ in action blocks alone. In the expanded EFSM, these faults replace existing states and edges with new ones.

Type II: The faults of the assignments, $v^* \leftarrow (\underline{v}, \underline{p})$ in action blocks alone. These are the same as type II faults except that these are related with input parameters.

Type III: The faults of the parameters in input declarations alone. The parameters in the input declaration of an edge, to be meaningful, should be used in the edge. These faults would mostly cause the faults of the action block or the output declaration with parameters in the edge. For the detection of this type of faults, therefore, we will check the faults of the action block with parameters, or type II faults, and the output declaration with parameters, or the next type IV faults.

Type IV: The faults of the parameters in output declarations alone. We assume that an expression with variables and parameters is allowed as a output parameter in an output declaration. We can detect of all the faults only by output declarations and values of their parameters. Type I or II faults are also detected by them. If we, therefore,

consider type IV faults, the probable expression in an output parameter can be regarded as an extension of the action block of the edge. So we can regard this type of faults as type I or II faults.

Type V: The faults of the predicates having no relation with parameters. In case of modification faults, the executable region of the predicate is wrongly changed, which may cause the nondeterminism or incompleteness of the specification, but that is not our concern. If the regions are changed with holding our assumption that any faults do not cause the nondeterminism or incompleteness, only the target states of the related edges are changed in the expanded EFSM. In case of addition faults, the finite executable region of the predicate is wrongly created, which may cause the incompleteness of the specification. If the wrongly created predicates hold our same assumption as above, the edges from the state are divided into more than one edge from the state in the original EFSM. In the expanded EFSM, however, the edges are not divided but only their target states are changed in the expanded EFSM, because there is no parameter in the predicate expression.

Type VI: The faults of the predicates having some relation with parameters. If faults exist in equalities, these can be treated as type V faults, because the possible values of input parameters which make the edge executable are unique each. But if they exist in inequalities, their possible values firing the edge can be in closed or open region. In case of modification faults, their properties are same as those of modification faults of type V, except the above property. In case of addition faults, their properties are same as those of addition faults of type V except the property that the edges from the state are also divided into more than one edge from the state in the expanded EFSM.

Type VII: The mixed faults of type I, II, and V of some edges. These faults may add many new states and edges in the expanded EFSM.

3.3. Test cases for the faults in data elements

Now, for each type of faults in data elements, we find out test cases fully covering the faults. Type I faults are the simplest. These faults do not increase the number of states and edges. A single fault of this type can be fully covered by the transition identification part with one UIO sequences at a state and the state identification part with the sequences. Multiple faults of this type can be also fully covered by the transition identification part with a UIO sequence at each state, the state identification part with the sequences, and the data flow test sequences satisfying all-du-paths criterion. Even if no test case can detect any fault, it does not matter because the faulty EFSM always works in the same manner to the original one.

Type II faults are a little more complicated. The defs of parameters of input declarations is the same as the defs of variables except the fact that the values of input parameters which make true the predicate expression may not be unique. We, the tester, do not know which values to select to surely detect this type of faults. To get feasible test cases covering these faults, we should assume the fault pattern of the implementation. If we assume that the action block, $v \leftarrow(v,\vec{p})$ is expressed as the linear functions, as $\mathbf{V} = \mathbf{AV} + \mathbf{BP} + \mathbf{C}$, where $\mathbf{V}$ is the variable matrix, $\mathbf{P}$ is the parameter matrix and $\mathbf{A}$,$\mathbf{B}$, and $\mathbf{C}$ are coefficient matrices, we can get the following theorem.

Theorem 1. In the expanded EFSM, all assignments of action blocks are expressed linear functions and type I and II faults change the assignments to another linear expressions, two identical test cases having different values of the input parameter can cover the faults.

Proof. By property 2 of the expanded EFSM, each variable in an edge has same value regardless of the previous paths. Therefore we can regard the action block of the edge, $\mathbf{V}=\mathbf{AV}+\mathbf{BP}+\mathbf{C}$ as $\mathbf{V}=\mathbf{BP}+\mathbf{C'}$ where $\mathbf{C'}=\mathbf{AV}+\mathbf{C}$. As this expression is a linear function of parameters, two different parameter values will determine the function. Therefore, two identical test cases having different values of input parameters can cover the faults. □

Type V faults may block right transitions or cause wrong transitions in the expanded EFSM. If these faults add extra edges in the general EFSM, they will cause wrong transitions without an addition of any extra edge in the expanded EFSM. This property clearly shows that these errors can be fully covered by classic control flow test cases only.

Test data selection of input parameters has been a major issue in software testing. The values of input parameters that make all the test cases executable may not be unique but may lie in closed or open region. The required test cases fully satisfying these faults, therefore, may not finite, so test data selection is necessary and it is the policy of the tester. Type VI faults face this problem. Therefore we assume that there is no type VI fault in the protocol implementation.

By the above analysis, we have only one combination type of faults, type VII faults. These faults are the most complex. As type II faults, to surely get finite test cases, we should have the same assumption as that of type II faults. By the assumption, this type of faults can be also fully covered by the control flow test cases with the transition identification part and the state identification part and the data flow test cases satisfying all-du-paths criterion. Even if no test case can detect any fault, it does not matter for the same reason as that of the type I fault.

The above analysis shows that control flow test cases with full fault coverage and data flow test cases satisfying ‘all-du-paths’ criterion are needed to guarantee high fault coverage in the expanded EFSM.

4. A unified optimization scheme

We are trying to generate the test cases having high fault coverage. Because the number of needed test cases, however, gets tremendous, some optimizations are needed. As there is normally a trade-off relation between the amount of test cases and the fault coverage of test cases, optimizations may drop the fault coverage. We propose an optimization scheme with the desired fault coverage defined in the previous chapter. This optimization scheme is depicted in Fig.3.

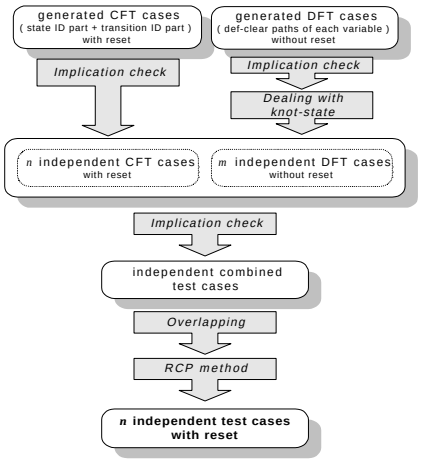

Fig.3 The proposed optimization procedure

Each implication check makes test cases independent together as shown in Fig.3. We only adopt the overlap method and RCP method[1] for optimization. We do not use MUIO(Multiple UIO) method[10] because it needs more test cases for full fault coverage[2]. The serious defect of UIO method, not full applicability, can be almost overcome by applying properties of EFSM and adopting delay as an input declaration. Transitions with delay statements keep the states from having their UIO sequences which have the same output declarations. In fact, however, the null output which outputs nothing is used as an output in spite of the similar timing problem. Therefore we also use delay as an input declaration, which can extend the test coverage of control flow test cases.

The generated control flow test cases for both the state identification and the transition identification are checked if a test case is implicated by another and the implicated test cases are discarded. Def-clear paths for all the variables compose data flow test cases, where the implication check is also run. In the EFSM, a major factor increasing test cases is the edges appended for completeness of the specification, which have null actions and null output declarations. These edges are represented as loops in the expanded EFSM as well as in the normal EFSM. In the states having these edges or many ingoing and outgoing edges, in fact most of states, the needed number of data flow test cases get multiplied as the number of edges. In the expanded EFSM with the assumption of linearity, we solve the problem of those states as follows.

Definition 4. A state is called a *knot-state* if it lies in both more than 2 distinct ingoing def-clear paths and more than 2 distinct outgoing def-clear paths in an expanded EFSM. A knot-state can have self loops without data element.

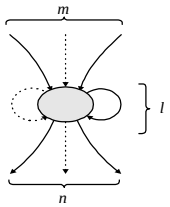

Fig.4 A model of knot state

Theorem 2. Let test cases have a common knot-state. When the knot-state connects m inward def-clear paths and n outward def-clear paths and it has l self loops without data element, selected $m+n+l-1$ def-clear paths among $m*n*l$ can cover all-du-paths criterion, if only linear faults in actions are allowed.

Proof. If an edge in Fig.4 has a fault in a side, the upper side or the lower side, of the knot state, a test case passing this edge may detect the fault. But the test case has another fault in the other side of the knot state, it may not detect faults. Then another test case passing the edge having the latter fault may detect the fault. And the process can be repeated one after another. If no test case can find any fault, this faulty specification is functionally equivalent to the original one. In other words, if all test cases can be connected in the edges without making any partition, they can find out the probable faults which can be detected by test cases satisfying all-du-paths criterion. A Loop at a knot state has the same property because it forms both ingoing and outgoing edge except it is made of only one edge. The minimum number of paths that can connect one side having $n+l$ elements and the other side having $m+l$ elements is $n+m+2l-1$. As loops are counted twice there, the minimum number is $n+m+l-1$. □

In the expanded EFSM, knot-states are naturally connected together. In other words, a fully extended def-clear path can contain multiple knot-states. In this case, the following theorem is valid.

Theorem 3. In a part of an expanded EFSM, where there are p edges composing def-clear paths, q knot states dependent together, and r o-uses for a variable, selected $p-q+r$ def-clear paths can cover all-du-paths criterion in the part, if only linear faults of the variable in actions are allowed.

Proof. In a knot state having n ingoing edges, l loops and m outgoing edges the needed number of test cases is $n+l+m-1$. This can be regarded as the number of ingoing edges of the following knot state. If we repeat this process to the knot-states with outgoing edges having o-uses, the minimum required number of test cases will be $p-q'$ where p is the number of edge composing the def-clear paths and q' is the number of knot-states which def-clear paths pass through. In the last knot state, the needed number of test cases is $(p-q')+r-1$. If we count the last knot-state, the number gets $p-q+r$, where $q=q'+1$. □

5. Empirical results and conclusions

To estimate the usefulness of the proposed method, we generated the test cases of a simple protocol, Inres[7] used often for the reason. Initiator module in Inres protocol is the target EFSM, where delay is treated as an input and the reset input is used. The original EFSM of Initiator module freed of edges for completely specified condition has 4 states and 14 edges. The complete specified transformed expanded EFSM has 18 states and 147 edges, but the number of edges except the edges added for comlpeteness of the specification is 57. Among them one edge is left not omitted, because it is a part of a UIO sequence. We finally generated 46 test cases for control flow test and 3 test cases for data flow test by the proposed optimization scheme. A large amount of test part detecting faults of data flows is transferred to control flow test cases by the proposed method. 3 data flow test cases are linked to the control flow test cases by RCP method.

By now, there has been little study on the fault model for data flows and the optimization of data flow test cases. The major reasons are the complexity of data flows and inter-dependency between control and data flows. To resolve these problems, we transformed an original EFSM into the equivalent expanded EFSM which is good to analyze without being interfered by those problems. With the advantages of the expanded EFSM, we proposed a test case generation method by fault coverage analysis. The analysis shows that control flow test cases with full fault coverage and data flow test cases satisfying 'all-du-paths' criterion are needed to guarantee high fault coverage. We also proposed an optimization scheme with some assumptions. We think that the efficient point on the nearly proportional curve between fault coverage and cost can be decided by the characteristics of the target protocol, which we are now studying.

References

[1] Alfred V. Aho *et al*., "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours", *Protocol Specification, Testing and Verification '88*, Atlantic City, USA, pp.75-86, 1988

[2] Ricardo Anido *et al*., "Guaranteeing full fault coverage for the UIO-based testing methods", *Int. Workshop on Protocol Test Systems '95*, Evry, France, pp.221-236, 1995. 9

[3] Samuel T.Chanson *et al*., "A unified approach to protocol test sequence generation", *IEEE INFOCOM '93*, San Francisco, USA, pp.106-114, 1993

[4] J. H. Park, J. P. Hong, Jai Yong Lee, "A conformance testing framework for applying test proposes", *95 Protocol Test Systems, Chapman & Hall*, 1995

[5] Iksoon Hwang,Taehyong Kim, Minseok Jang, Jaiyong Lee, Sangbae Lee, "A searching algorithm for generating test sequences in conformance testing", *JCCI '96*, Kwangju, Korea, pp.722-726, 1996.4

[6] Taehyong Kim, "Automatic generation of observation-based and length-optimized test cases for EFSM model in conformance testing", *Master thesis, Yonsei Univ.*, 1995. 8

[7] Do Young Lee, Jai Yong Lee, "A well-defined Estelle specification for the automatic test generation", *IEEE trans. on Computer*, Vol.4, No.4, pp.526-542, 1991.4

[8] Do Young Lee, Jai Yong Lee, "Test generation for the specification written in Estelle", *PSTV*, Vol.XI, Elsevier Publishing (North Holland), 1991

[9] Xiangdong Li *et al*., "Automatic generation of extended UIO sequences for communication protocols in an EFSM model", *Int. Workshop on Protocol Test Systems '94*, Tokyo, Japan, pp.213-228, 1994. 11

[10] Y. N. Shen *et al*., "Protocol conformance testing using multiple UIO sequences", *Protocol Specification, Testing and Verification '90*, Ottawa, Canada, pp.131-143, 1990

[11] Hasan Ural *et al*., "A test sequence selection method for protocol testing", *IEEE trans. on Comm.*, Vol.39, No.4, pp.514-523, 1991. 4

[12] Elain J. Weyuker *et al*., "Selecting software test data using data flow information", *IEEE trans. on SE*, Vol.11, No.4, pp.367-375, 1985. 4

[13] Byungmoon Chin, Taehyong Kim, Minseok Jang, Iksoon Hwang, Jaiyong Lee, Sangbae Lee, "Generation of Reliable and Optimized Test Cases for Data Flow Test with a Formal Approach", ICOIN-11, Taipei, Taiwan, 1997. 1

[14] O. Henniger *et al*., "Transformation of Estelle modules aiming at test case generation", *Int. Workshop on Protocol Test Systems '95*, Evry, France, pp.45-60, 1995. 9

[15] Iksoon Hwang, Taehyong Kim, Minseok Jang, Jaiyong Lee and Sangbae Lee, "A Method to Derive a Sing-EFSM from Communicating Multi-EFSM for Data Part Testing", *Int. Workshop on Testing of Communicating Systems '97*, Cheju, Korea, 1997. 9