Test Path Selection Based on Effective Domains*

Liang-Seng Koh and Ming T. Liu

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210-1277, USA

Abstract

In this paper, a method is proposed to produce test paths that check both data flow and control flow for a protocol specified in the Extended Finite State Machine (EFSM) model. The method first identifies a set of paths from a given specification to cover a dataflow selection criterion, then it appends state check sequences to some transitions in this set of paths for checking control flow. The criterion that our method employs for selecting these transitions is called effective domain for testing. Effective domain for testing is used to evaluate how effective a transition can be tested in a given path in terms of the range of values that the variables in this transition can have. Since each transition can appear in several paths, our method is to append state check sequences to its occurrences that have distinct effective domains. In addition, our method will compute the path domain for each path and make some inexecutable paths executable.

1 Introduction

Conformance testing aims at demonstrating that a protocol Implementation Under Test (IUT) conforms to the specification that it implements. Since an IUT is treated as a blackbox, test sequences that are sequences of input/output are needed for a tester to infer the properties of the IUT. For the purpose of generating test sequences, paths that are believed to be most error revealing must be selected from the corresponding specification. In the case that the specification is written in the Finite State Machine (FSM) model, the paths are, indeed, the test sequences. However, if the specification is specified in the Extended Finite State Machine (EFSM) model, test sequences will need to be constructed from these paths by substituting appropriate values for the input variables of each path.

*Research reported herein was supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

In the literature, protocols specified in the FSM are always tested by making sure that there is no control flow errors associated with each transition. For such a purpose, state check sequences such as Unique Input/Output (UIO) [1] and Characterizing set (W-set) [2] are employed to identify the tail state of a transition. However, these techniques are inadequate for generating paths that can effectively test a protocol specified in the EFSM since the model describes not only the control aspect of the protocol, but also the data aspect of the protocol. For testing an IUT specified in the EFSM, some methods [3, 4, 5] have also been proposed. These methods employ the techniques of dataflow analysis to analyze the data interactions among the variables and then to choose paths from the specification to cover these interactions. Although the data flow of a protocol can be effectively examined by these methods, the control flow is ignored by these methods. In addition, these methods only concern with the issue of path selection; two other important issues, namely deciding path executability and identifying path domains for each selected path, are not addressed.

In this paper, a method is proposed to combine data flow testing with control flow testing. In our method, a set of paths is first selected to fulfill certain dataflow selection criterion, then state check sequences are appended to the transitions in the set for checking control flow errors. As a transition can appear in several paths, *effective domain for testing* is introduced as a criterion for evaluating the occurrences of a transition. The concept of effective domains is employed in this paper as a criterion for measuring how extensive a transition can be tested in a given path in terms of the range of possible values that the variables of this transition can have in the path. For each transition, our method will append state check sequences to its occurrences that have distinct effective domains. In addition to generating test paths for testing both control flow and data flow, our method also generates path domains for the respective test paths and makes some inexecutable test paths executable. As mentioned above, these two problems are very important problems; however, hardly any in depth discussions are provided in

0-8186-6685-4/94 $4.00 © 1994 IEEE

the literature.

This paper is organised as follows: In Section 2, a model of the EFSM and the concept of dataflow testing are introduced. For ease of presenting our approach in the subsequent sections, a dataflow criterion called *all-du-paths* criterion is described in detail. The concept of effective domain for testing is discussed in Section 3. Then, assuming that a set of paths for covering all-du-paths criterion is given, a procedure for path selection is developed in Section 4. In Section 5, two related works recently published in the literature are compared with our work.

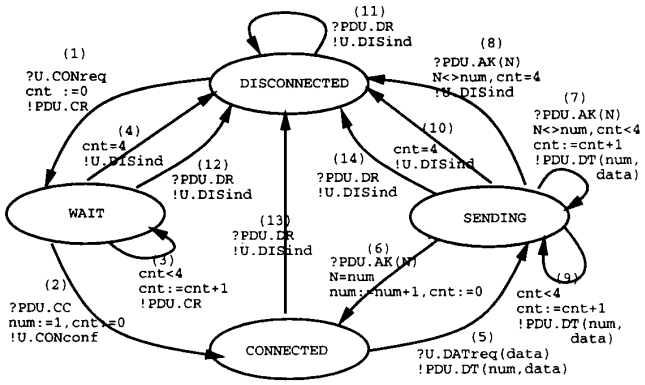

Figure 1: The INRES protocol

2 Background

2.1 Extended Finite State Machine

The EFSM model extends the FSM model by using variables to describe the data aspect of a protocol. In this paper, a protocol entity is modeled as an EFSM that is specified by a state-transition graph. In the graph, a state is represented by a vertex and a transition by a directed edge. Furthermore, it is assumed that every transition in the EFSM includes the following four parts that will be executed in this order: input, enabling conditions, program segment, and output.

Each of the four parts can have zero or more statements. Note that the order of statements given in each part must be observed. The input and output parts specify the interaction of a protocol with its environment. An input statement is prefixed by a '?' and an output statement is prefixed by a '!'. The enabling conditions part specifies the conditions under which a transition is fired when all the conditions are evaluated to be true. Thus, an enabling condition is a conjunctive predicate of all the conditions and is written as $p_1, p_2, \cdots, p_n$ where each p_n is a condition. Note that if the enabling conditions part has no statement, a transition can always be executed. The program segment is a sequence of assignment statements specifying the actions to be performed by a transition. An example of the INRES protocol [6] specified using this model is shown in Figure 1.

2.2 Data Flow Testing

In software engineering, dataflow testing is concerned with testing the data interactions in a program. Strategies based on dataflow testing, which are normally referred as dataflow selection criteria, look at interactions involving definitions for program variables and subsequent references that are affected by these definitions and require that certain such interactions be tested. Several of these criteria have been used in conformance testing for testing the data aspect of a protocol. For instance, the methods proposed in [3, 4] use *all-du-paths* [7]; whereas, the method proposed in [5] uses *all-simple-OI-paths* [8]. Since our method is to augment a set of paths generated to cover a dataflow selection criterion with some subsequences to cover the control aspect of a protocol, any of these dataflow criteria can be used in our method for selecting a set of paths to cover the data interactions. For ease of discussing our algorithm, the all-du-paths is used in this paper.

Before discussing the all-du-paths criterion, several terms need to be defined. In dataflow terminology, a variable, x, is said to be a definition (*def*) in a transition, t, if x appears in the input portion of t or x appears at the left hand side of an assignment statement in the program segment of t. However, when a variable, x, appears on the right hand side of an assignment statement s of t or in an output statement s of t, x is both a computation-use (*c-use*) variable of t and s, respectively. If x is used in an enabling condition statement s of t, then it is a predicate-use (*p-use*) variable of t and s, respectively. In addition, p-use(t) denotes a set of p-use variables in t and c-use(t) denotes a set of c-use variables in t. Similarly, p-use(s) and c-use(s) are for s. A *global use* of x is a use of x whose *def* occurs in some other transitions; otherwise, it is a *local use*. Similarly, a *global use* of x is a def of x for which there is a global use in some other transitions; otherwise, it is a *local def*.

A *path* is a sequence of transition. A path is a *simple path* if all transitions, except possibly the first and the last, are distinct. A path is a *loop-free* path if all transitions are distinct. A path $(t_1, \cdots, t_n)$ is a *def-clear* path with respect to x if $t_2, \cdots, t_{n-1}$ do not contain def of x. A path $(t_1, \cdots, t_n)$ is a *du-path* with respect to a variable x if t_1 has a global def of x and either: 1) t_n has a global c-use of x and the path is a def-clear simple path with

Transition	def Variables	du-paths	Test Paths
1	cnt	1 → 3	(1,3,4)
		1 → 4	(1,4)
2	cnt	2 → 5 → 10	(1,2,5,10)
		2 → 5 → 8	(1,2,5,8)
		2 → 5 → 9	(1,2,5,9,10)
		2 → 5 → 7	(1,2,5,7,10)
	num	2 → 5, 2 → 5 → 6	(1,2,5,6,5,8)
		2 → 5 → 8	(1,2,5,8)
		2 → 5 → 9	(1,2,5,9,8)
		2 → 5 → 7	(1,2,5,7,8)
		2 → 5 → 9 → 7	(1,2,5,9,7,8)
		2 → 5 → 7 → 9	(1,2,5,7,9,8)
3	cnt	3 → 3	(1,3,3,4)
		3 → 4	(1,3,4)
5	data	5 → 9 → 7, 5 → 9	(1,2,5,9,7,14)
		5 → 7 → 9, 5 → 7	(1,2,5,7,9,14)
6	cnt	6 → 5 → 10	(1,2,5,6,5,10)
		6 → 5 → 8	(1,2,5,6,5,8)
		6 → 5 → 7	(1,2,5,6,5,7,10)
		6 → 5 → 9	(1,2,5,6,5,9,10)
	num	6 → 5 → 8, 6 → 5	(1,2,5,6,5,8)
		6 → 5 → 7	(1,2,5,6,5,7,8)
		6 → 5 → 9	(1,2,5,6,5,9,8)
		6 → 5 → 7 → 9	(1,2,5,6,5,7,9,8)
		6 → 5 → 9 → 7	(1,2,5,6,5,9,7,8)
7	cnt	7 → 7	(1,2,5,7,7,10)
		7 → 9	(1,2,5,7,9,10)
		7 → 8	(1,2,5,7,8)
		7 → 10	(1,2,5,7,10)
9	cnt	9 → 7	(1,2,5,9,7,10)
		9 → 9	(1,2,5,9,9,10)
		9 → 8	(1,2,5,9,8)
		9 → 10	(1,2,5,9,10)

Table 1: Test Paths for Testing Data Aspect of INRES

respect to x or 2) t_n has a global p-use of x and the path is a def-clear loop-free path with respect to x. The all-du-paths criterion requires that all the du-paths for each variable in a given EFSM be covered by some paths in a test set. Therefore, selecting a set of paths to fulfill the all-du-paths criterion involves: 1) computing all the du-paths for each variable, and 2) finding a set of paths to cover these du-paths.

Table 1 shows all the du-paths extracted from the INRES protocol and a set of paths to cover these du-paths. Note that, du-paths that correspond to the same subsequence have only a single entry in this table. For instance, a subsequence $1 \rightarrow 3$ that is a du-path for p-use and a du-path for c-use of *cnt* has a common entry in the table. The main reason is that both the du-paths can be covered by the same test path. Each du-path d is covered by a path $(pre, d, post)$, where pre is the shortest path from the initial state of the protocol to the first transition of d and $post$ is the shortest path from the last transition of d to the initial state. Normally, not all paths in this set are executable due to the enabling conditions along the paths. In addition, since the dataflow technique is focused only on path selection, no clue is given to how path domains can be generated for developing test cases from these paths.

ETP No.	Executable Test Path	Corresponding Test Paths
1	(1,$\mathbf{3^4}$,4)	(1,3,4), (1,4), (1,3,3,4)
2	(1,2,5,$\mathbf{7^4}$,10)	(1,2,5,10),(1,2,5,7,10),(1,2,5,7,7,10)
3	(1,2,5,$\mathbf{7^4}$,8)	(1,2,5,8),(1,2,5,7,8)
4	(1,2,5,$\mathbf{9^4}$,10)	(1,2,5,9,10),(1,2,5,9,9,10)
5	(1,2,5,$\mathbf{9^4}$,8)	(1,2,5,9,8)
6	(1,2,5,6,5,$\mathbf{7^4}$,10)	(1,2,5,6,5,7,10),(1,2,5,6,5,10)
7	(1,2,5,6,5,$\mathbf{7^4}$,8)	(1,2,5,6,5,8),(1,2,5,6,5,7,8)
8	(1,2,5,6,5,$\mathbf{9^4}$,10)	(1,2,5,6,5,9,10)
9	(1,2,5,6,5,$\mathbf{9^4}$,8)	(1,2,5,6,5,9,8)
10	(1,2,5,7,$\mathbf{9^3}$,10)	(1,2,5,7,9,10)
11	(1,2,5,7,$\mathbf{9^3}$,8)	(1,2,5,7,9,8)
12	(1,2,5,9,$\mathbf{7^3}$,10)	(1,2,5,9,7,10)
13	(1,2,5,9,$\mathbf{7^3}$,8)	(1,2,5,9,7,8)
14	(1,2,5,6,5,7,$\mathbf{9^3}$,8)	(1,2,5,6,5,7,9,8)
15	(1,2,5,6,5,9,$\mathbf{7^3}$,8)	(1,2,5,6,5,9,7,8)
16	(1,2,5,7,9,14)	(1,2,5,7,9,14)
17	(1,2,5,9,7,14)	(1,2,5,9,7,14)

Table 2: Executable Test Paths for Testing Data Aspect of INRES

3 Effective Domain For Testing

In order to conform that a path chosen to cover some data interactions in a specification is indeed the same path taken by an IUT during execution, our method requires that every transition of the path be checked for control flow errors. This involves checking the tail state of every transition. Since the tail states cannot be observed directly, state check sequences must be appended to the respective transitions. However, a transition typically can exist in different test paths; therefore, if all the occurrences of a transition must be checked, then the length of the paths will be significantly increased. On the other hand, it is generally not sufficient to conclude that all occurrences of a transition are correct by simply testing one of these occurrences. It is due to the fact that an occurrence may be tested on some aspects of a transition that other occurrences cannot offer. (This point should become obvious after our discussion on effective domains.) In order to reduce the number of testing for a transition, strategies are needed for effectively selecting some occurrences of this transition so that fault coverage is not severely degraded. *Effective domain for testing of a transition in a path* provides a kind of measurement for evaluating the extent to which a transition can be tested in a given path.

The concept of effective domain for testing is based on the intuitive relationships between dataflow testing and domain testing [9]. The primary characteristic of domain

testing is that a program's input domain is divided into subsets, with the tester selecting one or more elements from each subdomain. An *input domain* of a program is a *subset of* the Cartesian product of the respective range of values for each input variable. An element of an input domain is called an *input vector*. In a way, dataflow testing is a type of domain testing. Dataflow testing requires the exercising of path segments determined by combinations of variable definitions and variable uses. The input domain is divided so that there is a subdomain corresponding to each path covering the def-use associations [10]. A given subdomain contains every input vector that causes the particular def-use associations to be satisfied. This subdomain of a path is called the *path domain* of that path.

Let's call a tuple $< (x_1, v_1), \cdots, (x_n, v_n) >$, where $x_1, \cdots, x_n$ are all variables of a program and $v_i \in$ range of $x_i \cup \{\mho\}$, a *state vector* of the program. $\mho$ is a null value. Then, a path domain defines an initial set of state vectors for a path. Based on this set, the execution of the first transition in turn determines the new set of state vectors for the second transition, and so forth. Each of these sets of state vectors before its corresponding transition is executed is collectively called the *executable state set* of that transition. Note that two transitions in a path may not be distinct transitions from a specification. Any possible execution of a transition (or rather an *occurrence* of a transition from a specification) in a path is restricted by its executable state set. Specifically, a tuple in this executable state set determines the respective values of those c-use and p-use variables of the transition. As a result, testing of this transition in the path can only be done on a certain combination of values of its c-use and p-use variables. With a given path, the allowed subsets of Cartesian product of the possible values for the p-use and c-use variables of a transition is collectively called the effective domain for testing of that transition in the path. The effective domain for testing provides a way that a tester can estimate how much a transition of a specification can be tested by its respective occurrences in a given path. Formally, the effective domain for testing is defined as follows:

Definition 1 *For a given path, $p = (t_{p_1}, \cdots, t_{p_n})$, the effective domain for testing of a transition, t_{p_i}, in p, denoted by $ED_{p,t_{p_i}}$, is*

$$\{proj_{x_{j_1},\cdots,x_{j_m}}(e) \mid e \in \textit{execution state set of } t_{p_i}\}$$
where $x_{j_1} \in$ *c-use*$(t_{p_i}) \cup$ *p-use*(t_{p_i})
and $proj_{y_{j_1},\cdots,y_{j_k}} < (y_1, v_1), \cdots, (y_p, v_q) > =$
$< (y_{j_1}, v_{j_1}), \cdots, (y_{j_k}, v_{j_k}) >$
for $1 \leq j_1 < j_2 < \cdots < j_k \leq q$

In the rest of this paper, if no ambiguity arises due to paths or transitions, an effective domain for testing of a transition in a path will simply be stated as an effective domain.

To illustrate the idea of effective domains, let's study a set of executable paths in Table 2, which is obtained from the test paths as shown in Table 1. Note that not all the test paths in Table 1 are executable. Some of the inexecutable paths can be made executable while computing their effective domains. The details will be presented in the next section. At this moment, let's assume that all executable paths have been obtained.

Table 3 shows the effective domains of the executable test paths (ETP) numbered 1, 2, 3, 5 and 7 in Table 2. For simplicity,
$< (x_1, v_1), \cdots, x_i : \{a_1, \cdots, a_n\}, \cdots, (x_m, v_m) >$ is used to collectively denote the following state vectors $< (x_1, v_1), \cdots, (x_i, a_j), \cdots, (x_m, v_m) >$ for $1 \leq j \leq n$. Similarly, for a transition t_i (says, having two variables x and y) from Figure 1, $i < x : [v_1][v_2]\cdots[v_n], y : [u_1][u_2]\cdots[u_n] >$ is used to denote the n occurrences of t_i in a given path, where v_j and u_j are values of x and y, respectively, for the j^{th} occurrence of t_i. If y has the same value u in all n occurrences, then it will be denoted as $i < x : [v_1][v_2]\cdots[v_n], (y, u) >$. For the five paths, the effective domains for transition 1 are not shown. They are $< (cnt, \mho) >$. In ETP 1, each occurrence of transition 3 can be tested with $cnt = 0$, $cnt = 1$, $cnt = 2$ and $cnt = 3$, respectively. This is because the first time transition 3 is tested, the value of cnt has been set to 0 by transition 1. Then, cnt is increased by one by the statement $cnt := cnt + 1$. As a result, the second time transition 3 is tested, the transition is tested with the value of cnt equal to 1. The same argument applies to the third and forth iterations of transition 3.

In Table 2, it also can be seen that transition 8 has different effective domains in ETP 3 and ETP 7. ETP 3 has $< (cnt, 4), (num, 1), N : [2..) >$, but ETP 7 has $< (cnt, 4), (num, 2), N : [3..) >$. However, under many circumstances, a transition that appears in two different paths can have the same effective domain. For instance, transition 8 has the same effective domain of $< (cnt, 4), (num, 1), N : [2..) >$ in both ETP 3 and ETP 5, and transition 9 has the same effective domain of $< cnt : [0][1][2][3], (num, 1), data : [a..z] >$ for ETP 5 and ETP 7. Therefore, ETP 3 and ETP 5 can only provide the same effectiveness in terms of ranges of values for testing transition 8, and ETP 5 and ETP 7 can only provide the same for transition 9.

Thus, using effective domains, the occurrences of a transition in different paths can be qualitatively evaluated. For a transition, our strategy is then to choose a subset of occurrences that can adequately cover all the

ETP No.	Executable Test Path	Effective Domain For Testing
1	1,3^4,4	3 < cnt : [0][1][2][3] > 4 < $(cnt, 4)$ >
2	1,2,5,7^4,10	2 < $(cnt, 0)$ > 5 < $(num, 1), data$: $[a..z]$ > 7 < cnt : [0][1][2][3], $(num, 1), N$: [2..), $data$: $[a..z]$ > 10 < $(cnt, 4)$ >
3	1,2,5,7^4,8	2 < $(cnt, 0)$ > 5 < $(num, 1), data$: $[a..z]$ > 7 < cnt : [0][1][2][3], $(num, 1), N$: [2..), $data$: $[a..z]$ > 8 < $(cnt, 4), (num, 1), N$: [2..) >
5	1,2,5,9^4,8	2 < $(cnt, 0)$ > 5 < $(num, 1), data$: $[a..z]$ > 9 < cnt : [0][1][2][3], $(num, 1), data$: $[a..z]$ > 8 < $(cnt, 4), (num, 1), N$: [2..) >
7	1,2,5,6,5,9^4,8	2 $(cnt, 0)$ > 5 < num : [1][2], $data$: $[a..z]$ > 6 < $(num, 1), (N, 1)$ > 9 < cnt : [0][1][2][3], $(num, 2), data$: $[a..z]$ > 8 < $(cnt, 4), (num, 2), N$: $1 or [3..)$ >

Table 3: Effective Domain of Some Test Paths

distinct effective domains. Since the reasoning for effective domains is parallel to that of domain testing, the commonly used definition for distinct domains in the context of domain testing is adopted in our context; that is, two domains are said to be distinct if they do not have the same set of variables of which each variable has an identical domain. In this sense, two domains such that one is a subset of the other are still considered distinct.

4 Test Path Selection

Assuming that a set of paths, which may contain inexecutable paths, has been selected for fulfilling the all-du-paths criterion, our test path selection algorithm will perform two more tasks. They are computing the effective domains for the paths and selecting the occurrences of the transitions.

4.1 Computing Effective Domains

Since not all the transitions in a specification are included in the set of selected paths, those yet-to-be selected transitions need to be included in this set. For this purpose, a tour containing these transitions will be added to the set.

Our strategy in computing effective domains for a given path is to progressively computing the executable state sets for a newly encountered transition while traversing the path and modifying the executable state sets of some traversed transitions to reflect the constraints imposed by the newly visited transition. Each of the conditional and assignment statements imposes a certain constraint on those variables that appear in the statement. The allowed values for a variable will be represented by the allowed maximum and the allowed minimum values of the variable, and a list of constraints that specifies the relations of this variable with some other variables.

Initially, the executable state set of the initial transition consists of the Cartesian product of the variables. When a transition is encountered, each of its enabling condition t will be evaluated against the current boundaries of the involved variables. This evaluation is done by computing the new boundary of a selected variable v among c-use(t). The selection prefers input variables to context variables. This preference is to reduce the chances of failure and the number of changes required for the executable state sets along the path. Based on this computed boundary D, the logic for further execution is as follows:

If D is not empty then
 If the new constraint conflicts with existing constraints
 then path may be inexecutable
 else
 If D is a subset of the old domain D' of v
 then do nothing
 else
 D:=$D \cap D'$
 propagate D to the executable state sets of which the corresponding transitions have variables directly or indirectly dependent on v and update the constraints of these transitions
else path may be inexecutable

For checking the consistency of the new constraint against the existing constraints, only those constraints directly or indirectly related to those variable in c-use(t) need to be examined. Generally, the constraints for communicating protocols are in simple forms, such as linear

form, and a list of the relavent constraints are normally very short. For instance, at most of the time, the constraint list for most of the variables in INRES has only a single constraint. In addition, since the domains of the variables are known, consistency of the relevant constraints can be easily checked by the constraint satisfaction problem (CSP) technique [11].

In the context of communicating protocols, most of the inexecutable paths can be made executable if some sequences of transitions are inserted into the original paths. This always involves inserting a cycle to a particular state of an original path to change the values of v. A loop that can alter the values of v will be chosen. The number of times this loop must be unfolded is determined by the number of execution needed to bring the values of v within D. This loop will be unfolded a number of times as required and inserted into the given test path. The constraints of the related transitions will be updated accordingly.

An assignment statement is not only to define the value of its def variable, but also to impose constraints on the def and c-use variables of the statement. This constraint will be used to restrict some c-use variables if the possible values of the def variable has to be restricted based on some constraints imposed by later statements. An output statement does not impose any constraint on the variables involved.

Based on an executable state set, the corresponding effective domain can be computed as the boundaries of the p-use and c-use variables, and constraints relevant to these variables. It should be noted that for most of the test sequence generation methods proposed in the literature, the test path selection and test case selection are separated. For those test sequence generation methods based on dataflow criteria, a step which may be manual walkthrough or computed automatically by some symbolic execution methods, is unavoidable for computing the path domains and choosing test cases based on the paths in the given test set. In our method, the path domain as well as the constraints on the input variables are computed. These two pieces of information will help in generating input vectors.

4.2 Selecting Occurrences of a Transition

After computing effective domains for the occurrences of each transition, the next task is to select those occurrences that have distinct effective domains for a transition. Identical effective domains of a transition are its occurrences that have identical boundaries and constraints for the relevant variables. However, some care must be taken to reduce the overall length of the resulted test paths. There are two reduction rules that can be employed to reduce the number of occurrences of a transition before making selection. The first rule is to be applied to a set of occurrences that have distinct domains, and the second rule is to be applied to a set of occurrences that have identical effective domains. The first reduction rule is as follows:

R1 **Redundant Domains**: An effective domain, $ED_{p_j,t}$ can be eliminated if there exists some $ED_{p_i,t}$ (says n) for $1 \leq i \leq n$ and $i \neq j$ such that

$$ED_{p_j,t} = \cup_{i=1}^{n} ED_{p_i,t}$$

where $\cup$ is defined for a tuple of range of values as follows:

$$\cup_{i=1}^{n} < D_{i1}, \cdots, D_{im} > = < \cup_{i=1}^{n} D_{i1}, \cdots, \cup_{i=1}^{n} D_{im} >$$

In R1, the effective domain $ED_{p_j,t}$ is made redundant by the n effective domains $ED_{p_i,t}$ since testing of t in p_j with any possible values for variables of t before t is executed can also be conducted with some p_i.

Before describing the second reduction rule, let's discuss the problems of appending state check sequences to transitions. For each transition, state check sequences must be appended to the tail states of those selected occurrences. To ensure that the fault coverage of these paths is not severely affected after state check sequences are introduced, these newly added state check sequences must be selected with care. In general, introducing more transitions to a path of an FSM cannot degrade the fault coverage of that path, but introducing more transitions in a path of an EFSM may degrade the fault coverage of that path. A simple example is that a newly introduced segment may redefine some variables in a path that is to cover a def-clear segment. Three types of state check sequences can be used to serve the purpose of identifying a tail state. The preference that it is chosen for a tail state is according to the following order.

Type 1 An unique check sequence such as an UIO sequence for a state such that this sequence overlaps with the subsequent transitions.

Type 2 A sequence that concatenates 1) a unique check sequence for a state with 2) a transfer sequence to bring the flow back to the tail state, and does not redefine the def variables of any def-use association that the original path is supposed to cover.

Type 3 A sequence that concatenates a unique check sequence for a state with a transfer sequence to bring the flow back to the initial state and then back to the tail state.

Obviously, state check sequences of Type 1 do not alter the effective domains of transitions in a path. In addition,

States	UIO Sequences
DISCONNECTED	1
WAIT	3 or 2
CONNECTED	5
SENDING	7 or 6 or 8 or 9

Table 4: State Check Sequences for INRES

Type 1 sequences do not increase the length of a path. Type 3 requires that all the variables be reset after a state check sequence is applied. It is, thus, less preferred than the other two types of check sequences.

The second reduction rule is based on the Type 1 state check sequences. It deals with a set of occurrences of a transition that have an identical effective domain. If there exists an occurrence in this set that has an UIO of the transition overlapping with its path, this occurrence can be chosen as the representative for this set of occurrences. However, since this occurrence has an UIO of the transition overlapping with the path, no extra checking is needed at all. Thus, the second rule can be formulated as follows:

R2 **Overlapping Transitions**: No checking is needed for a set of occurrences of a transition t that has an identical effective domain if there exists a path p_i in this set such that p_i contains a subsequence (t, U), where U is a state check sequence for the tail state of t.

An application of this rule to some paths shown in Table 2 is as follows: As shown in Table 3, both ETP 2 and ETP 3 have the same effective domain of $< (cnt, 3), (num, 1), N : [2..), data : [a..z] >$ for the forth iteration of transition 7. By applying R2, the forth iteration of transition 7 in ETP 2 needs not be checked. It is because ETP 3 contains a subsequence (7,8). As shown in Table 4, transition 8 is an UIO of the tail state of transition 7.

Some caution of words must be given here. By applying R1 to a set of test paths, the fault coverage for the resulting set cannot be degraded. However, applying R2 may degrade the fault coverage of a set. According to [12], the fault coverage of UIOs for a set of transitions can be degraded if these UIOs are overlapped with the subsequent transitions in a test path that is supposed to test this set of transitions. Since the tail state of a transition is always tested a number of times by our method in different occurrences, the chances of degrading are reduced. Nevertheless, Miller and Paul [13] pointed out that if there is no *converging state* in a path, fault coverage is not compromised by overlapping the UIOs of the tail states of the intermediate transitions. Converging states are those states which have outgoing transitions with identical input/output that go to the same state.

ETP No.	Final Test Path
1	$(1, 3^4, \underline{2, 13}, 1, 3^4, 4, \underline{1, 12})$
2	$(1, 2, 5, 7^4, 10, \underline{1, 12})$
3	$(1, 2, 5, 7^4, 8, \underline{1, 12})$
4	$(1, 2, 5, 9^4, 10)$
5	$(1, 2, 5, 9^4, 8)$
6	$(1, 2, 5, 6, 5, 7^4, 10)$
7	$(1, 2, 5, 6, 5, 7^4, 8, \underline{1, 12})$
8	$(1, 2, 5, 6, 5, 9^4, 10)$
9	$(1, 2, 5, 6, 5, 9^4, 8)$
10	$(1, 2, 5, 7, 9^3, 10)$
11	$(1, 2, 5, 7, 9^3, 8)$
12	$(1, 2, 5, 9, 7^3, 10)$
13	$(1, 2, 5, 9, 7^3, 8)$
14	$(1, 2, 5, 6, 5, 7, 9^3, 8)$
15	$(1, 2, 5, 6, 5, 9, 7^3, 8)$
16	$(1, 2, 5, 7, 9, 14, \underline{1, 12})$
17	$(1, 2, 5, 9, 7, 14)$
18	$(1, 3, 12, \underline{1, 12}, 11, 1, 2, 13, \underline{1, 12})$

Table 5: A Final Set of Test Paths for INRES

After applying these two rules, it will be easy to decide for which the occurrences of a transition, state check sequences are needed. For those domains that have only one occurrence, these occurrences will be checked. For each of those domains that have more than one occurrences, only one is needed. Of all these selected occurrences, Type 2 or Type 3 sequences can be used for each of them.

4.3 An Example

Based on the aforementioned algorithm, a final test set that can test both the data flow and the control flow of INRES is given in Table 5. The state check sequences are underlined. Except that the state check sequences for transition 3 in ETP 1 are for the WAIT state, the others are all for the DISCONNECTED state. Even though transition 8 appears 8 times, the tail state of transition 8 is only tested at ETP 3 and ETP 7. Similarly, the tail state of transition 10 is tested once at test sequence 2 even though transition 10 appears 6 times. Indeed, transitions 8 and 10 use the same state check sequence; it is $(1, 12)$ and is of Type 2. Transition 1 is an UIO of DISCONNECTED and transition 12 is a transfer sequence to bring the flow back to DISCONNECTED. This same state sequence is also used for testing transition 13 in ETP 13 and transition 14 in ETP 16.

In ETP 1, the last execution of transition 3 that has an effective domain of $< (cnt, 3) >$ is tested by a Type 3 state check sequence. This sequence consists of transition 2 and a subsequence, (13,1,3^4). Note that the correctness of transition 3 is checked at the first three iterations by

a Type 1 state check sequence which is (3). Since transition 3 is tested in ETP 1 with an effective domain of $< (cnt, U) >$, it is not tested in ETP 18 again.

5 Conclusion

In this paper, an approach is proposed for generating paths that check both data and control flow of a protocol. After generating a set of paths to cover a dataflow selection criterion, our method uses the concept of effective domains to selectively augment the transitions in the set with state check sequences so that the control flow can be ensured and the dataflow coverage can still be preserved. Our method also computes path domains for the paths and makes some inexecutable paths executable.

In terms of fault detection capability, our method preserves the fault coverage of the dataflow analysis technique since the def-use associations of a criterion that a selected path is supposed to cover are not disturbed by any of the three types of state check sequences. In addition, our approach also ensures that every transition and its tail state is tested at least once. This is the minimum requirement in the conventional sense for checking control errors. However, as mentioned in Section 3, it is not sufficient for the EFSM. The resulting test sets further ensure that every distinct effective domain is covered. For a transition, our algorithm only appends state check sequences to those occurrences that have distinct effective domains, thereby reducing the number of state check sequences to be appended to a test set.

To the best of our knowledge, there are only two works [14, 15] that have dealt with the issue of combining control flow testing with dataflow testing. But, neither has proposed ways for computing path domains for the respective paths. In [14], a combining method is used for generating a test path for a given mutant. Basically, this method uses segment overlapping technique to shorten the overall length of a test path and only requires that a transition be tested once. As mentioned in Section 3, a transition in an EFSM generally must be tested more than once for revealing errors that may only occur in certain paths. In [15], a method is proposed to augment a set of paths generated to cover the all-du-paths criterion with cyclic characterizing sequences (CCS). A CCS is the same as a Type 2 state check sequence. By applying Miller's result reported in [13] for the FSM, this method only inserts a CCS to the last state of a path if there is no converging state along the path. In the case that several converging states appear in a path, each converging state is replaced by a CCS. In our method, these two cases are taken care of by the Type 1 and Type 2 sequences. In addition, our method do not check a transition whose tail state is a converging state more than once if its occurrences have the same effective domain. In our example, if transition 7 is not an UIO for SENDING, then our method will only check transition 7 for its four iterations in ETP 2. However, the method of [15] will have to check all occurrences of transition 7 in various paths.

Regarding our future work, strategies are being investigated for using the underlying control structures of a specification to reduce the number of test cases needed for data flow testing.

References

[1] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.

[2] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, March 1978.

[3] B. Sarikaya, G. v. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 5, pp. 518–531, May 1987.

[4] H. Ural, "Test sequence selection based on static data flow analysis," *Computer Communications*, vol. 10, no. 5, pp. 234–242, 1987.

[5] H. Ural and B. Yang, "A test sequence selection method for protocol testing," *IEEE Trans. on Communications*, vol. 39, no. 4, pp. 514–523, April 1991.

[6] D. Hogrefe, "OSI formal specification case study: The INRES protocol and service," tech. rep., IAM 91-012, University of Berne, Institute of Computer Science and Applied Mathematics, 1991.

[7] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. on Software Engineering*, vol. 11, no. 4, pp. 367–375, April 1985.

[8] H. Ural and B. Yang, "A structural test selection criterion," *Infomation Processing Letter*, vol. 28, no. 3, pp. 157–163, 1988.

[9] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Trans. on Software Engineering*, vol. 6, no. 3, pp. 247–257, May 1980.

[10] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Trans. on Software Engineering*, vol. 17, no. 7, pp. 703–711, July 1991.

[11] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[12] M.-S. Chen, Y. Choi, and A. Kershenbaum, "Approaches utilizing segment overlap to minimize test sequences," in *Proc. of 10th IFIP Symp. on Protocol Specification, Testing, and Verification*, pp. 67–84, 1990.

[13] R. E. Miller and S. Paul, "Generating minimal length test sequences for conformance testing of communication protocols," in *Proc. IEEE INFOCOM '91*, pp. 970–979, 1991.

[14] R. E. Miller and S. Paul, "Generating conformance test sequences for combined control and data of communication protocols," in *IFIP Trans. Protocol Specification, Testing, and Verification, XII*, pp. 1–15, North-Holland, 1992.

[15] S. T. Chanson and J. Zhu, "A unified approach to protocol test sequence generation," in *Proc. IEEE INFOCOM '93*, pp. 106–114, March 1993.