A Novel Framework for Test Domain Reduction using Extended Finite State Machine

Nutchakorn Ngamsaowaros
Department of Computer Science,
Faculty of Science and Technology,
Thammasat University (Rangsit Campus),
Patumthani, 12121, Thailand
nng@cs.tu.ac.th

Peraphon Sophatsathit
Advanced Virtual and Intelligent Computing
(AVIC) Center, Faculty of Science,
Chulalongkorn University,
Bangkok, 10330, Thailand
peraphon.s@chula.ac.th

Abstract

Test case generation is an expensive, tedious, and error-prone process in software testing. In this paper, test case generation is accomplished using an Extended Finite State Machine (EFSM). The proper domain representative along the specified path is selected based on fundamental calculus approximation. The pre/post-conditions of class behavior is derived from a continuous or piece-wise continuous function whose values are chosen from partitioned subdomains. Subsequent test data for the designated class can be generated from the selected test frames. In so doing, the domain is partitioned wherein reduced test cases are generated, yet insuring complete test coverage of the designated test plan. The proposed modeling technique will be conducive toward a new realm of test domain analysis. Its validity can also be procedurally proved by straightforward mathematical principles.

Keywords : Software Testing, Domain Analysis, Test Partitioning, Path Testing.

1. Introduction

Testing is an important stage of software development and maintenance. It provides a method to establish confidence in the reliability of software. It is an expensive process in software engineering. Many researchers are interested in this cost reduction. A difficult problem of testing is automated test data generation in a small test suit that satisfies the designated testing criteria.

Several approaches have been proposed in the literature to reduce the number of test cases and to automate the test case generation process, such as Combinatorial Black-Box Testing Techniques [11] [4], Finite State Machine Based Testing [5] [13] [9] [7], and Domain Testing [14] [10] [6]. However, manual generation of such test cases can be time-consuming and error prone. The primary goal of this work is to reduce the size of the input domain and be able to detect faults in the same manner as combinatorial techniques.

Our approach is based on model-based test generation (MBTG) which can be modeled using formal description language like Specification and Description Language (SDL). The model describes operations in terms of (1) System state S, (2) Input/Output parameters X, Y, and (3) Pre- and postconditions on S, P. An Extended Finite State Machine (EFSM) can be extracted from the underlying model. There are many approaches such as [8], [3], [12], and [1] that make use of an EFSM/FSM representation from MBTG. Then, test cases can be generated using structural criteria on the extracted EFSM.

This paper proposes a new approach to generate test data that combining black-box and white-box testing. We use black-box testing which is based on domain partitioning by means of a software function f that maps its input to output. We use structural criteria on the EFSM, represented by predicates, to partition and reduce input domains in the same equivalence class to the same output, i.e., f(x) = f(y). This implies that one input from each equivalence class provides 100% output coverage.

This paper is organized as follows. Section 2 describes some basic concepts used in the proposed approach, EFSM model, partition testing, and the fundamentals of domain reduction. Section 3 elaborates the proposed technique. The essence of model application is given in Section 4. Some benefits and possible extensions are summarized in the Conclusion. Future Work furnishes some inferences for new courses of action.

2 Basic Concepts and Definitions

In this section, we describe some basic concepts which will be used in the proposed approach. A few relevant definitions, notations, and assumptions are also provided.

2.1 System Model

The model-based test generation is an approach to generate test cases from a model-based specification which is formal specification. It can be used as a prototype to bridge the gap between user requirements and formal specification. The model can serve both the purpose of specifying how the system should respond to inputs from its environment and to guide the selection of test cases. The model formulation is elucidated below.

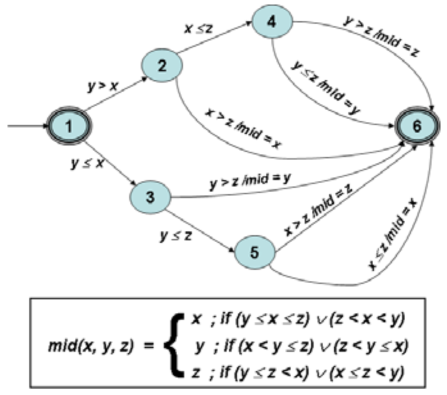
2.1.1 Extended Finite State Machine

Definition 1 Let X and Y be finite sets of input and output parameters, and V be a finite set of local and external variables. Denote the domain of X, Y, and V by D_X, D_Y, and D_V, respectively.

An *Extended Finite State Machine* (EFSM) M over X, Y, V, D_X, D_Y, and D_V is a tuple (S, s, f, T), where

1. S is a finite set of states,
2. s is the initial state,
3. f is the exit state,
4. T is a finite set of transitions from states $i \rightarrow j$, provided that $i, j \in S$. A transition $t \in T$ is a 7-tuple $(a, x, v, \rho, \Delta, y, a')$, where
   - $a, a' \in S$ are the initial and final states of the transition, respectively.
   - $x \in X$ is the input to a and $y \in Y$ is the output of a
   - $\rho : D_X \times D_V \rightarrow \{True, False\}$ is a predicate function of the transition.
   - $\Delta : D_X \times D_V \rightarrow D_Y$ is the output parameter function of the transition.

An EFSM consists of states (including an initial state and a final state) and transition between states. A state transition is triggered by an event provided that the enabling condition is satisfied. When a transition is traversed, the corresponding output parameter functions may be performed. An output parameter function may manipulate variables, read input, or produce output. An enabling condition is a boolean predicate that may use EFSM variables and must evaluate to TRUE in order to transit to next state. An EFSM can be represented as directed graphs where states are denoted by nodes and transitions by directed arcs. Figure 1 shows a graphical illustration of an EFSM representing the specifications of *mid()* function. $State$ 1 and $state$ 6 denote initial and exit state, respectively.



x ≤ z
y > z /mid = z
y > x
y ≤ z /mid = y
x > z /mid = x
y ≤ x
y > z /mid = y
x > z /mid = z
y ≤ z
x ≤ z /mid = x
1
2
3
4
5
6
mid(x, y, z) = x ; if (y ≤ x ≤ z) ∨ (z < x < y)
y ; if (x < y ≤ z) ∨ (z < y ≤ x)
z ; if (y ≤ z < x) ∨ (x ≤ z < y)

Figure 1. EFSM representing the specifications of *mid()* function.

2.1.2 Domain Mapping Function

As EFSM operations commence, the state changes while executing the input parameters that subsequently transformed into output parameters. These output parameters will in turn become input parameters of the next state. Thus, some input domain portion may be cut off during such transition through the path of execution using predicate function (ρ) as shown in Figure 2 while traveling through the path 1-2-4-6.

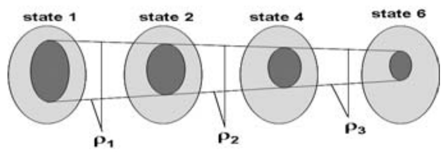


Figure 2. Domain perspective in an EFSM model ($D_X \overset{y>x}{\longmapsto} D'_X \overset{x>0}{\longmapsto} D''_X$)

A *predicate function* is an algebraic expression of input and local variables related by one of the conditional operators $\{>, <, =, \geq, \leq, \neq\}$. The output value of this function is $TRUE$ or $FALSE$. If the value is $TRUE$, then it maps the valid domain from node i to the next node $i + 1$. Thus, a predicate function restricts the space of program variables to certain portions of the input domain. For example, the predicate $x > 10$ describes the portion of the input domain of the incoming node that must be greater than 10.

To simplify the problem, predicate functions in this paper are based on the following assumptions:

- The predicate functions are of numeric relationships.
- Domains are either finite integer (piece-wise) or continuous.
- predicate functions are of the forms

- $x \Theta y$,
- $x \Theta c$, or
- $x \Theta exp$

where Θ denotes a relational operator $\{>, <, =, \geq, \leq, \neq\}$, c is a constant, and exp is an expression.

2.2 Fundamentals of Domain Reduction

From the previous example, an input domain may be reduced by getting rid of invalid subdomains. From Figure 2, the predicate function ρ maps the relevant domain from node i to the next node, reducing the size of the domain. The invalid portions are thus discarded from the original domain during such transitions.

To further elaborate domain reduction concepts, consider a predicate function of two variables x and y. Let $[l_x, u_x]$ and $[l_y, u_y]$ be the domains of x and y. And suppose that the precondition is $x > y$. The six possible domain arrangements of x and y are shown in Figure 3. The invalid portions of x and y satisfying the predicate $x > y$ are highlighted in the Figure. From this example, additional premises can be inferred that a domain may be reduced if

- variables are in the same convex domain, and
- variables are *dependent*.

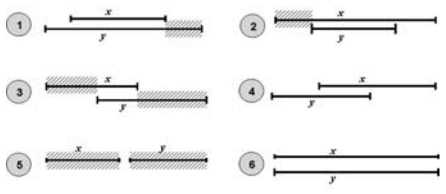


Figure 3. The valid domains satisfying the predicate $x > y$. The shaded areas are invalid domains

2.3 Hoare Triple Sequence

We utilize Hoare tipple to describe the formulation of the proposed EFSM model.

Definition 2 Let $a \in S$ be an arbitrary state. An assertion at a, denoted by $g(a)$, can be evaluated to true or false. An assertion of the preceding state a is called a *precondition* P, and an assertion of the succeeding state a is called a *postcondition* Q. The set of precondition and postcondition are denoted by $\{P\}$ and $\{Q\}$, respectively.

Definition 3 An execution sequence is a **Hoare triple** denoted by $\{P\}M\{Q\}$.

The precondition $\{P\}$ is the condition representing selected input value while the postcondition $\{Q\}$ describes the output domain. M is the assertion statements $g(a)$ under investigation.

Figure 4 demonstrates how Hoare triple is applied to the proposed model. $\{P, D_X^0\}$ selects the input domain for use by the EFSM, which in turn produces $\{Q, D_Y\}$. Further analysis of the EFSM (M_0) reveals the path 1-M_1-6 formed by $\{(y > x), D'_X)\}M_1\{Q, D_Y\}$. Using the precondition as a domain mapping function and focusing on the dependent variables, a series of progressive transitions can be established to derive some expressions (predicates) representing the path expression, or simply put *state transition from* i *to* j. One such expression/predicate is the output of node 1 as $y > x$ which becomes the input of M_1. Subsequent analysis of M_i can be carried out in the same manner.

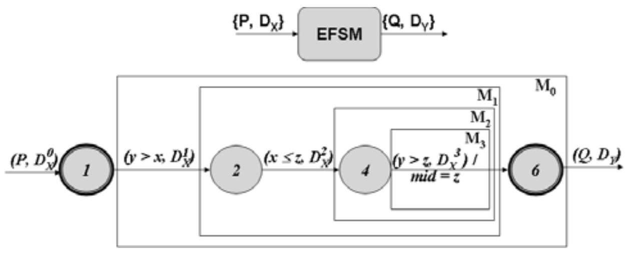


Figure 4. Hoare Triple being applied to an EFSM model

3 Domain of Partition Testing

Partition testing is a popular approach to generate test cases from a model-based specification. This study employs the technique to divide the input domain into subdomains, whereby representatives of each sub-domain are selected to generate the desired test cases from proper combination of subdomains. Two input values are considered to be equivalent if and only if the operations have the same behavior satisfying the same preconditions of the specification. We will introduce the data structure and its construction rules used in the domain partitioned model. Figure 3 demonstrates one-dimensional domain partitioning from the given predicate. We will show how the same modeling approach is extended to higher dimensions and more complicated predicate functions in the sections that follow.

3.1 Domain Partitioned Model

The basic representation of input domain partitioning to subdomains employs conventional tree notion. As shown in Figure 5, the input domain denotes the root of the tree. All intermediate nodes represent various hierarchies of partitions and sub-partitions. The leaf nodes denote test frames generated from their corresponding parent partitions. The data structure of such representation can be written as a 2-tuple sequence of the form (ρ, D), where ρ is a predicate function serving as the partition function and D is the set of variables. The partition function is either predicates in an EFSM or preconditions $\{P\}$ from Hoare triple sequence. The resulting domain subspace Z can be defined as

$$Z_{ij} = \{(P, D_X^0), (\rho_i, D_i^1), (\rho_{ij}, D_{ij}^2)\}$$

where $i \in \{1...m\}$ and $j \in \{1...n\}$.

The i^{th} domain at level k partitioned in n pieces can be defined as

$$D_{X,i}^k = \bigcup_{j=1}^{n} D_{X,ij}^{k+1}$$

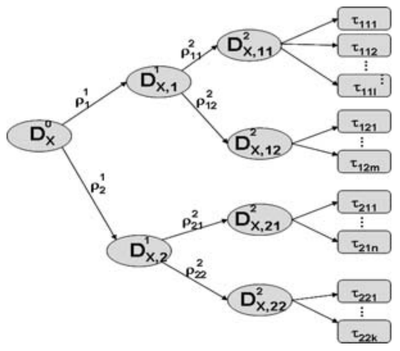


Figure 5. Domain partitioning tree representation

3.2 Test Frame Derivation

Analysis of partition function on a given input domain utilizes principles of Calculus. The underlying input specification, preconditions, and predicates are taken into derivation. If the set of variables is continuous over the entire input domain, the corresponding partition function will be easily derived. On the contrary, should the above governing mandates do not come in concert with one another, the set of variables may not be entirely continuous, but piece-wise continuous. In which case, the representative function must be separately derived, resulting in partitioned domain functions. We illustrate the derivation by a few examples as shown in Figure 6, where 6a and 6b demonstrate frame derivation on linear and quadratic domains. The third example (6c) depicts the case of piece-wise continuous partition function.

Thus, the valid domain can be estimated by the function:

$$\sum_{i=1}^{n}(u_y - f(i'\Delta x))\Delta x$$

where $\Delta x = \frac{u_x - l_x}{n}; i' = i - 1$

However, it is not necessary to divide the rectangle in the same width. The proposed partitioning technique by rectangular approximation merely rests on Calculus principles that are simple to automate the test process.

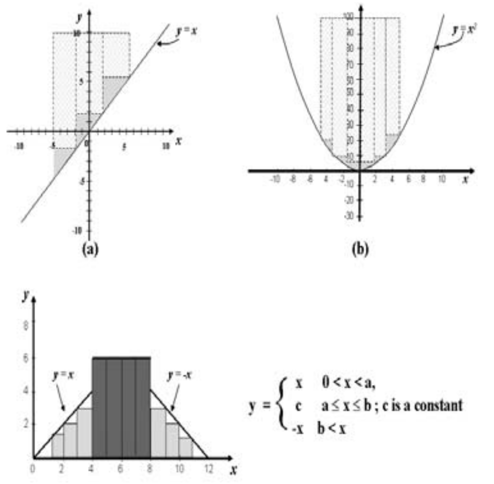


Figure 6. Examples of test frames derivation using rectangular approximation

During rectangular approximation, each subdivision D_X^0, $D_{X,i}^1$,$D_{X,ij}^2$,$D_{X,ijk}^3$,... determines not only how close the approximation will be, but the granularity of test partitions as Δx is successively refined, or equivalently deeper traversal into domain partition tree. Upon stopping of subdivisioning, the path from D_X to each leaf node will denote an expression of the designated test frame. All variables, preconditions, and input specification of that test frame will constitute the desired test cases.

Each test frame can be described from the root node D_X to its leaf node τ_{ij} as follows:

$$\tau_{ij} = (l_x \le x \le u_x) \wedge (f(x) \le y \le u_y)$$

where $f(x) = \rho$ is a predicate function, l_x is the lower bound of x satisfying the function $f(x)$, and u_y is the upper bound of y satisfying the function $f(x)$.

A test case, t, generated from a given test frame, τ_{ij}, can be expressed as

$$t = \{(x, y) \in \Re^2 | l_x \le x \le u_x, f(x) \le y \le u_y\}.$$

Once the test frame and its corresponding test cases are obtained, a test path that forms the domain subspace Z is applied to the EFSM for cross-validation on all preconditions $\{P\}$, statements M, and postconditions $\{Q\}$.

The following comprehensive example describes the process of test frame generation from domain partitioning.

Example Consider the test problem of Figure 1 which is further elucidated in Figure 4. Suppose we want to generate test cases which exercise the path 1-2-4-6. Let the initial domains of x, y, and z be x,y,z∈ [0,...,99]. If we map the domain to two dimensional graph, the initial domain covered by x and y will be the area bounded by the lines x = 0, x = 99, y = 0, and y = 99 which form a rectangle shown in Figure 7(a). This domain is the input domain D_X^0 of M_0. The predicate $y > x$ (ρ_1) enables the transition to the exit state of M_0. This predicate function partitions D_X^0 into two subdomains, namely $D_{X,1}^1$ satisfying the condition $y > x$ and $D_{X,2}^1$ satisfying $y \leq x$, as depicted in Figure 7(b). The reduced domain D^1 thus serves the test path 1-2.

Consider the next test path 2-4, rather than taking all $D_{X,1}^1$ to be the input domain for M_1, we further divide $D_{X,1}^1$ into four test frames $D_{X,11}^1$, $D_{X,12}^1$, $D_{X,13}^1$, and $D_{X,14}^1$ using *rectangular estimation* principle of Calculus as shown in Figure 7(c). Figure 7(d) and (e) illustrate the domain partition on condition $z \geq x$.

The final transition from state 4 to 6 considers the relationship between the domains of y and z, superimposed on the previous domain in Figure 7 (e). The final partitioning is degenerated to lines bounding previously defined partitions that represent only valid z.

It is worth noting that domain partitioning using rectangular approximation can be performed at different level of granularity. The above example utilizes four equal width partitions $D_{X,11}^1$, $D_{X,12}^1$, $D_{X,13}^1$, and $D_{X,14}^1$ for illustrative purpose only. In actual analysis, Δx should be small enough to closely approximate the predicate function in the same manner as that of Calculus.

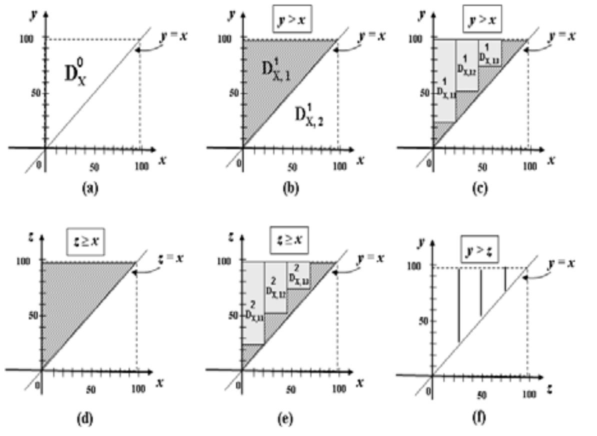


Figure 7. Test frame generation for the path 1-2-4-6

3.2.1 Representation of Domain

For each execution state of the EFSM, the set of input parameters, as well as output parameters can be denoted by X_i, and Y_i, respectively, where $X_i = \{X_1, \ldots, X_m\}$, and $Y_i = \{Y_1, \ldots, X_n\}$. The region of each domain is represented by

$$\begin{array}{ccccccccccc} l_{x_1} & \leq & x_1 & \leq & u_{x_1}, & l_{y_1} & \leq & y_1 & \leq & u_{y_1} \\ l_{x_2} & \leq & x_2 & \leq & u_{x_2}, & l_{y_2} & \leq & y_2 & \leq & u_{y_2} \\ & & \vdots & & & & & \vdots & & \\ l_{x_m} & \leq & x_m & \leq & u_{x_m}, & l_{y_n} & \leq & y_n & \leq & u_{y_n} \end{array}$$

in matrix notation

$$\begin{bmatrix} l_{x_1} \\ l_{x_2} \\ \vdots \\ l_{x_m} \end{bmatrix}_i \leq \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}_i \leq \begin{bmatrix} u_{x_1} \\ u_{x_2} \\ \vdots \\ u_{x_m} \end{bmatrix}_i,$$

$$\begin{bmatrix} l_{y_1} \\ l_{y_2} \\ \vdots \\ l_{y_n} \end{bmatrix}_i \leq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_i \leq \begin{bmatrix} u_{y_1} \\ u_{y_2} \\ \vdots \\ u_{y_m} \end{bmatrix}_i$$

or in final matrix form

$$L_X \leq X \leq U_X, L_Y \leq Y \leq U_Y$$

3.2.2 Predicate Function Representation

In the proposed EFSM model, a predicate function of an edge between state i and $i{+}1$ can be represented in a general equation as follows:

$$A_1\vec{X_i^1} + A_2\vec{X_i^2} + ... + A_n\vec{X_i^n}\Theta C$$

where Θ is one of conditional operators $\{>, <, =, \geq, \leq, \neq\}$, $\vec{X_i^n}$ is a vector of variables x_k^n at state i, and A_n is a coefficient matrix of vector $\vec{X}$, $A_n \in C^{n \times n}$ of the form

$$\vec{X_i^n} = \begin{bmatrix} x_1^n \\ x_2^n \\ \vdots \\ x_m^n \end{bmatrix}_i, A_n = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1m} \\ a_{21} & a_{22} & ... & a_{2m} \\ & & \vdots & \\ a_{m1} & a_{m2} & ... & a_{mm} \end{bmatrix}$$

For example, the predicate function $y > x^2$ represented in equation as $y - x^2 > 0$ denoted in matrix by

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \end{bmatrix} [>] \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

4 Model Evaluation

To achieve the goal of the study, the following questions are addressed:

1. How is the effectiveness measured?

2. How is cost reduction measured?

Two metrics are defined to answer the above questions quantitatively.

The *effectiveness* is measured by the percentage of domain-specification requirements covering the test model, which can be simply estimated by summation of τ_{ij}. The domain-specification requirements are computed by $\int_{l_x}^{u_x}(u_y - f(x))dx$. Thus, the effectiveness (VT) is evaluated as follows:

$$VT = \frac{\sum_{j=1}^{k} \tau_{i,j}}{\int_{l_x}^{u_x}(u_y - f(x))dx}$$

This study employs a simple bounded region which is bounded by the lines $x_1 = l'_x$, $x_2 = u'_x$, $y_1 = l'_y$, and $y_2 = u'_y$. Such a premise lends itself to exploit the benefits from rectangular approximation as shown in the above example. Each rectangular region (τ_{ij}) can be computed by

$$\tau_{ij} = (u'_x - l'_x) \times (u'_y - l'_y)$$

Table 1. VT of predicate function $y > x$

No.of τ	$x, y \in [-20, ..., 20]$	$x \in [-6, ..., 6]$, $y \in [0, ..., 90]$
1	13.0435	69.2308
2	56.5217	69.2308
4	78.2609	86.5385
8	89.1304	93.7500
16	94.4293	96.9952
32	96.9769	98.2359
64	98.2507	99.2713

The VT indicates the coverage of domain testing. Table 1 shows the resulting VT approaching 100% as the number of τ_{ij} increases. A corresponding test case for each τ_{ij} is generated. Therefore, if the results yield high VT, more number of test cases must be generated accordingly.

The *cost reduction* measures how much the size of the domain is reduced. To evaluate cost reduction, the selected domain is compared with the initial domain. The ratio of selected domain to the initial domain is therefore

$$VS = \frac{\sum_{j=1}^{k} \tau_{i,j}}{(u_x - l_x) \times (u_y - l_y)} \times 100$$

5. Model Application

To generate test frames from an EFSM, a path from the initial state to the exit state and the corresponding input domain must be specified. Preconditions are used to scope the input domain. Predicates along the selected path are also applied to progressively partition the domain, as well as to check if the derived test frames are valid domain for the next state. An algorithm for test frame generation is given below.

```
Input: Input domain $D_X$, specified path from initial
       state to exit state.
Output: Set of test frames
CurrentState = $s$;
$T_{CurrentState} = \emptyset$; {Set of test frames of current
state}
Use $precondition$ of transition to reduce the domain
of test frame;
while $CurrentState \neq ExitState$ do
    if $T_{CurrentState} = \emptyset$ then
        Partition domain into $n$ test frames with
        predicate function using rectangular
        estimation method;
        Add partition test frame into $T_{CurrentState}$
        if Each test frame size is less than $\epsilon$ then
            if $CurrentState \neq a_0$ then
                CurrentState = PreviousState
            else
                No solution found {not possible
                domain partition}
            end
        end
    end
    else
        Select some test frames to be the domain
        of the next state and remove them from
        $T_{CurrentState}$;
        CurrentState = NextState;
    end
  end
end
```

Algorithm 1: The test frame generation algorithm

Based on the definitions and representations of the proposed domain reduction framework established so far, the final EFSM of the sample program culminates to the following derivations which are summarized in Figure 8.

$$\{P, D_X^0\}M_0\{\infty, \emptyset\}$$

$$\{(y > x), D_{X,1}^1\}M_1\{\infty, \emptyset\}$$

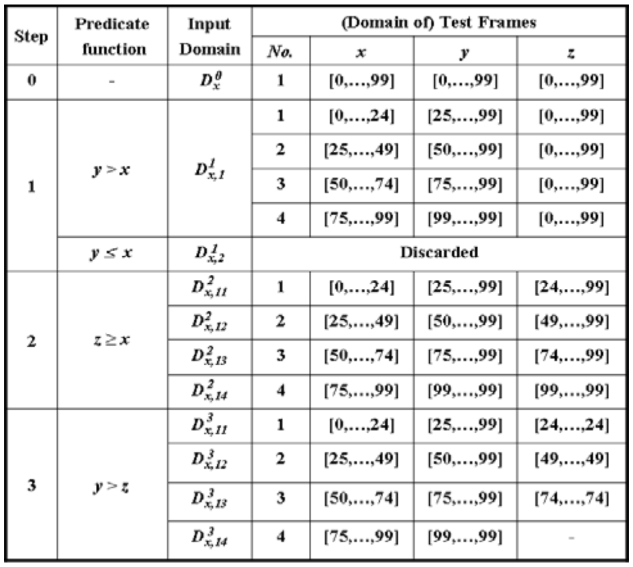
$$\{(x \leq z), \{D_{X,11}^2, D_{X,12}^2, D_{X,13}^2\}\}M_2\{\infty, \emptyset\}$$

$$\{(y > z), \{D_{X,11}^3, D_{X,12}^3, D_{X,13}^3\}\}M_3\{(mid = z, D_Y\}$$

The initial input domain, under the given predicate function, is reduced by the ratio *VS*. The output parameter *mid*, which is initially undefined, takes on designated intermediate results as the transition progresses, and is eventually stabilized as the domain shrinks down by the ratio VT. Execution of the EFSM terminates as it reaches the exit state.

The percentage of domain reduction is computed by

$$VS = \frac{75}{1,000,000} = 0.000075$$



Step	Predicate function	Input Domain	(Domain of) Test Frames			
			No.	*x*	*y*	*z*
0	-	D_x^0	1	[0,…,99]	[0,…,99]	[0,…,99]
1	$y > x$	$D_{x,1}^1$	1	[0,…,24]	[25,…,99]	[0,…,99]
			2	[25,…,49]	[50,…,99]	[0,…,99]
			3	[50,…,74]	[75,…,99]	[0,…,99]
			4	[75,…,99]	[99,…,99]	[0,…,99]
	$y \le x$	$D_{x,2}^1$	Discarded			
2	$z \ge x$	$D_{x,11}^2$	1	[0,…,24]	[25,…,99]	[24,…,99]
		$D_{x,12}^2$	2	[25,…,49]	[50,…,99]	[49,…,99]
		$D_{x,13}^2$	3	[50,…,74]	[75,…,99]	[74,…,99]
		$D_{x,14}^2$	4	[75,…,99]	[99,…,99]	[99,…,99]
3	$y > z$	$D_{x,11}^3$	1	[0,…,24]	[25,…,99]	[24,…,24]
		$D_{x,12}^3$	2	[25,…,49]	[50,…,99]	[49,…,49]
		$D_{x,13}^3$	3	[50,…,74]	[75,…,99]	[74,…,74]
		$D_{x,14}^3$	4	[75,…,99]	[99,…,99]	-

Figure 8. Test frame derivation for the path 1-2-4-6

The VS indicates that the size of the initial input domain is reduced by 99.99%. Since there are only three valid test frames, three test cases representing such test frames from the above derivation of test path 1-2-4-6 are generated. They are

1. x = 22, y = 25, z = 24
2. x = 35, y = 50, z = 49
3. x = 50, y = 75, z = 74

The VT can be computed by

$$VT = \frac{\sum_{j=1}^{4} \tau_{1,j}}{\int_0^{99} (99 - x)dx} = \frac{377,500}{500,000} = 0.755$$

which implies that only 75.5% of the approximated subpartition done at state 1 are used along the test path.

6. Conclusion and Future Work

The proposed domain reduction technique, simple as it may seen, offers an effective means to carry out test domain analysis. This is due entirely to the simplicity yet well-established of various fundamental principles, ranging from Calculus, logic, to Linear Algebra. Such basic building blocks furnish not only proven bases of formulation and derivation, but straightforward application to untangle some recalcitrant test problems that otherwise difficult to overcome by sole application of conventional methods. Nonetheless, the limitations of the proposed technique lie in nature of input domain which must exhibit some continuity (either continuous or piece-wise continuous). We envision that higher mathematics can be introduced to accommodate such shortcomings, as well as an addition of preprocessing stage to cluster discrete or non-numeric data so as to be ready for subsequent processing, thereby continuity requirement can be dropped.

References

[1] S. Chanson and J.Zhu. A unified approach to protocol test sequence generation. In *IEEE Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future.*, pages 1d.11–1d.19, 1993.

[2] T. Chen, P. lok Poon, and T. Tse. A choice relation framework for supporting category-partition test case generation. *IEEE Transactions on Software Engineering*, 29(7):577–593, July 2003.

[3] K.-T. Cheng and A. S. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. Des. Autom. Electron. Syst.*, 1(1):57–79, 1996.

[4] D. M. Cohen, S. R. Dalal, M. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[5] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *Proceedings of International Symposium on Software Testing and Analysis*, pages 112–122, 2002.

[6] D. Hamlet. On subdomains: Testing, profiles, and components. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 71–76, 2002.

[7] D. Lee. Principles and methods of testing finite state machines - a survey. *Proceedings of IEEE*, 84(8):1090–1123, August 1996.

[8] D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Trans. Comput.*, 43(3):306–320, 1994.

[9] A. Petrenko, S. Boroday, and R. Groz. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, 30(1):29–42, January 2004.

[10] S. D. Stoller. Domain partitioning for open reactive systems. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 44–54, July 2002.

[11] K.-C. Tai and Y. Lei. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, January 2002.

[12] H. Ural and B. Yang. A test sequence selection method for protocol testing. *IEEE Trans. Communications*, 39(4):514–523, 1991.

[13] C.-J. Wang and M. T. Liu. Generating test cases for EFSM with given fault models. In *Proceedings of IEEE INFOCOM*, pages 774–781, 1993.

[14] L. J. White and E. I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, 6(3):247–257, May 1980.