A Method for Executable Protocol Conformance Test Sequences Generation using Parametric Executable Analysis Tree

Ting SHU*, Guanghui YANG, Jinsong XIA

School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China

Abstract

Automatic generation of executable test sequences is one of the key issues in conformance testing for protocols specified in extended finite state machines (EFSM). The traditional test generation methods based on the executable analysis tree, generally require the initial assignment of variables in the EFSM under testing. Unfortunately, how to select the initial value for input variables is a challenging problem. To address the above issues, this paper presents a novel method which automatically generates an executable test sequence without the initial value assignment using parametric executable analysis tree (PEAT). Our method introduces a variable constraint equation group (VCEG) and its standardization technique to derive the constraint relationship among variables regarding to a generated test sequence. Then, relying on a VCEG, a PEAT is constructed to develop the executable test sequences with a suite of variables constraints. As a result, the problem of executable test sequences generation is reformulated to a PEAT exploration problem. The experimental results show the effectiveness of the proposed approach for executable test sequences generation.

Keywords: Conformance Testing; Test Sequences Generation; Parametric Transition Analysis Tree; Variable Constraint Equation

1 Introduction

A protocol conformance testing is an activity to check whether the implementation of a given protocol under test (IUT) conforms to its specification [1]. For model-based protocol conformance testing, the EFSM has been widely used as the formal modeling approach for the specification under testing. The process of EFSM-based conformance testing is to provide the IUT with a suite of inputs and observe its outputs. These inputs and outputs are known as test sequences represented by transition paths of the EFSM. Using the EFSM model, automatic test sequences generation can facilitate protocol testing. However, due to the condition conflicts among transitions of the EFSM under testing, the generated test sequences usually encounter the unexecutable
problem. That is, there does not exist feasible test data to trigger the generated conformance test sequences.

The main reason of test sequences’ unexecutable problem is that some D-use of variables in transitions may lead to the unsatisfied predicate conditions of other transitions, with P-use of the corresponding variables. Commonly, data dependence relationship contributes to the condition conflicts among transition sequences known as D-P path [2]. To solve this problem, some methods have been proposed during the last decades [3, 4, 6, 7]. Huang [2] presents an executable EFSM-based test sequence generation method using executable analysis tree with initial variables assignment. But he did not provide a feasible approach for initial value selection. Although Duale [5] transforms the EFSM under testing to a new one without infeasible paths, his method can only work on EFSMs in which all operations and predicate conditions are linear. In [8-10], the problem of test sequences generation is converted into an optimization one to be solved using the genetic algorithm. However, the genetic-based methods does not guarantee there must exist a solution, which depends on the choice of their parent test paths. In addition, the selection of their test data to trigger the generated test sequences is a challenging problem.

In this paper, we propose a novel method for automatically generating test sequences. The proposed method introduces a PEAT to derive a test sequence without the initial variables assignment. In addition, the feasible constraints of initial variables regarding to the generated test sequences are available at the end of test generation. Hence, the problem of automatic executable test sequences generation is reformulated to a PEAT exploration problem. Finally, we design a test generation algorithm for D-P coverage criteria and a case study on two classical protocols (Simplified Inres [7] and Network Monitor [2]) is made to show the effectiveness of the new method.

2 Preliminaries

An EFSM is formally described as a 7-tuple $M = < S, V, I, O, P, A, T >$. $S$ is the finite state set of the EFSM. $V$ is $M$’s finite variables set. $I$ and $O$ represent the input event set and the output event set, respectively. $P$ is the set of predicates on variables. $\forall p \in P$, $p = E(v_1, v_2, ..., v_n, op^s)$ where $v_1, v_2, ..., v_n$ is the list of variables, $v_i \in V$ and $op^s \in \{<, >, \leq, \geq, \neq\}$ is the operator symbol. $A$ is the set of actions operated on variables, such as variable assignment. $T = \{t_i = (s^h, s^t, i, o, p)|s^h$, $s^t \in S \land i \in I \land o \in O \land p \in P\}$ represents the transition set. $s^h = HeadState(t_i)$ and $s^t = TailState(t_i)$ are the head state and tail state, respectively.

![Fig. 1: An EFSM $M_1$](image)

The modified EFSM $M_1$ [9] shown in Fig. 1 will be utilized as an example throughout this paper.
to describe the test sequence generation method. In $M_1$, a circle represents a specific state and a straight or curve line with an arrow indicates a state transition from the tail state to the head state. In this example, $S = \{S_1, S_2, S_3\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ and $V = \{v_1, v_2, v_3, p_1, p_2, p_3\}$. For instance, when the state $S_1$ receives an input “aa” and the predicate conditions “$p_1 \geq 10 \& \& p_1 \leq 20 \& \& p_2 \geq 0 \& \& p_2 \leq 10$” are satisfied, $t_1$ will execute the actions “$v_1 := p_1; v_2 := p_2$” and give the output “00”. In this situation, $t_1$ is called an executable transition.

To facilitate the description of our approach, we give the following definitions.

**Definition 1** A transition path (TP) can be represented as $R = t_1, t_2, \ldots, t_i, t_{i+1}, \ldots, t_n$, where $t_i \in T \& \& \text{TailState}(t_i) = \text{HeadState}(t_{i+1})$, $1 < i < n$. $|R|$ is the length of the transition path. $	ext{HeadState}(R) = \text{HeadState}(t_1)$ and $\text{TailState}(R) = \text{TailState}(t_n)$ is $R$’s head state and tail state, respectively. When $\wedge_{i=1}^{n} t_i, p_{i} = \text{true}$, the TP is called an executable transition path (ETP).

For example, $R_i = t_2, t_3, t_4$ is a transition path of $M_1$. When an initial assignment of 5, 15, 5 for the variable $v_2$, $p_1$, $p_2$, respectively, the predicate conditions of every transition in $R_i$ is satisfied. That is, $R_i$ is an ETP.

**Definition 2** A variable constraint equation group (VCEG) can be represented as $FG = \{IE_1, IE_2, IE_3, \ldots, IE_n\}$ where $IE_i$ represents the specific variable constraint equation regarding the $i$-th transition $t_i$ in the ETP.

The VCEG provides a bird’s-eye view of the constraint relationship between variables corresponding to a transition path. For the $R_i$ given above, its $FG_i$ can be represented as $\{v_3 > 0, v_1 > v_2, p_1 > p_2 + 2\}$ corresponding to transition $t_2, t_3, t_4$. However, transitions may have combined conditions such as $p_1 \geq 10 \& \& p_1 \leq 20$ in $t_1$. For this situation, the detail discussion and solution are given in Section 3.1.

**Definition 3** An extended state configuration (ESC) can be denoted as $N_i = \{S_i, R_i, FG_i, V_i\}$ where $S_i$ is the name of current state, $S_i \in S$. $R_i$ is the transition path from the initial state $\text{HeadState}(R_i)$ to the state $S_i$. $FG_i$ is the variable constraint equation group of $R_i$. $V_i \subseteq V$ represents the involved variables in $FG_i$.

For the $R_i$ given above, the $N_i$ of $\text{TailState}(R_i)$ is $N_i = \{S_1(t_2, t_3, t_4), \{v_3 > 0, v_1 > v_2, p_1 > p_2 + 2\}, \{v_1, v_2, v_3, p_1, p_2\}\}$.

**Definition 4** A parametric executable analysis tree (PEAT) can be denoted as a 3-tuple $TT^{PEAT} = < N_0, NS, ES >$. $N_0$ is the root node. $N_i \in NS$ is the tree node and $NS$ is the node set. $ES$ is the edge set of $TT^{PEAT}$ where $ES = \{t_i|t_i, p = \text{true} \& \& N_j \in NS \& \& t_i \in T \& \& N_i, S_i = \text{HeadState}(t_i) \& \& N_j, S_j = \text{TailState}(t_i)\}$.

With regard to the initial variables assignment, the PEAT distinguishes from the traditional executable analysis tree. Using the PEAT, we can generate an executable spanning tree based on the EFSM under testing without the initial values assignment to variables. An example of PEAT of $M_1$ is shown in Fig. 2. The root node that is the initial extended state configuration $N_0$ is $\{S_1(), \}, \}$ and the tail state of the bold transition path is $N_0 = \{S_3(t_2, t_3, t_6), \{v_3 > 5, p_1 > p_2\}, \{v_3, p_1, p_2\}\}$. 

Fig. 2: M₁’s PEAT with the initial state S₁

3 Construction of VCEG

VCEG is an important description method regarding to the variable constraint relationship between transitions in a TP, which can facilitate us deriving the feasible variable domain during the test sequence generation. Generally, there exists various predicate condition forms in a transition. For convenience, we should standardize and merge all the condition inequalities in a VCEG.

3.1 Variable constraint inequality standardization

A FG consists of a group of IE{s}, each of which may have a different condition expression. The standardization of an IE need convert it into the uniform \( IE^s = CRV \times VV \ op \ c \). The \( CRV \) is a coefficient row vector in which element values are not all zero and \( VV = (v_1, v_2, ..., v_n)^T \) is a variable vector in regard to the \( IE \). \( op \in \{<, \leq, =, \neq\} \) is the comparison operator. \( c \) represents a constant.

For M₁, the default order of variables is \( v_1, v_2, v_3, p_1, p_2, p_3 \) in this paper, unless otherwise noted. Hence, the \( VV \) of M₁ can be represented as \( (v_1, v_2, v_3, p_1, p_2, p_3)^T \). For example, for the transition \( t_6 \), one of its variable constraint inequalities: \( p_1 > p_2 \) can be converted to \( p_2 - p_1 < 0 \) in the standardization process. Therefore, the \( CRV \) of \( t_6 \) can be represented as \( (0, 0, 0, -1, 1, 0) \).

A transition may have combined predicates linked by “&&” and “∥”. For the former, we divide them into several atomic inequalities. For the latter, a FG can be separated into several new FG{s}. For instance, we assume that the \( FG_i \) contains an inequality \( IE_i = \{IE_{i1}||IE_{i2}\} \). Then, the \( FG_i \) can be divided into \( FG_{i1} = \{IE_{i1}, IE_{i2}, IE_{i3}, ..., IE_{i1}\} \) and \( FG_{i2} = \{IE_{i1}, IE_{i2}, IE_{i3}, ..., IE_{i2}\} \). If \( IE_i = \{IE_{i1}&&IE_{i2}\} \), then the two inequalities \( IE_{i1} \) and \( IE_{i2} \) should be appended to \( FG_i \) to construct a new \( FG_i' = \{IE_{i1}, IE_{i2}, IE_{i3}, ..., IE_{i1}, IE_{i2}\} \).

3.2 Variable constraint inequality merger

To merge inequalities, we assume that all IE{s} have been standardized. For the \( CRV_i \) of an \( IE_i \), we separate it to several unit vectors \( \vec{a}_1, \vec{a}_2, ..., \vec{a}_i, \vec{a}_{i+1}, ..., \vec{a}_n \) where every \( \vec{a}_i \) has only a non-zero element according to the non-zero variable coefficient. If the coefficient of a variable in \( CRV \)
is a positive number, the element in $\vec{a}_i$ is 1. Otherwise, it is -1. For instance, the CRV of $t_6$ mentioned above can be separated into two vectors $\vec{a}_1 = (0, 0, 0, -1, 0, 0)$ and $\vec{a}_2 = (0, 0, 0, 0, 1, 0)$. Thus CRV can be represented as $CRV_i = \vec{a}_1 + \vec{a}_2$. Let $op_i$ be the operator of $IE_i$. Suppose that $IE_j$ is another constraint inequality in $FG_k$ with an operator $op_j$. Then the merging operation of $IE_i$ and $IE_j$ which is also the construction of a $FG_k$ can be presented as an algorithm $MergeIE$ shown as following steps:

**Step 1** Extract the CRV of $IE_i$ and separate CRV to a unit vector group $\vec{a}_1, \vec{a}_2, ..., \vec{a}_i, \vec{a}_{i+1}, ..., \vec{a}_n$;

**Step 2** For the CRV $j$ of an $IE_j$ in $FG_k$, if there exists a group of coefficients ($k_1, k_2, ..., k_n$) which consists of positive integer and zero but not all zero, $CRV_j$ can be represented as $CRV_j = k_1\vec{a}_1 + k_2\vec{a}_2 + ... + k_n\vec{a}_n$. Then the merging procedure will be performed as following:

a. When $op_i \in \{<, \leq\}$ and $op_j \in \{<, \leq\}$, the intersection of $IE_i$ and $IE_j$ will be appended into $FG_k$. In addition, $IE_j$ should be removed from $FG_k$;

b. When $op_i \in \{<, \leq\}$ and $op_j \in \{\neq\}$, $IE_i$ should be appended into $FG_k$ directly;

c. When $op_i \in \{\neq\}$ and $op_j \in \{<, \leq\}$, the merging operation equals to situation b;

Otherwise, $IE_i$ will be appended into $FG_k$ directly.

**Step 3** Repeat Step 2 until all $IE_j$ in $FG_k$ have been checked.

For the bold branch denoted as $R_k = t_2, t_3, t_6$ in Fig. 2, the construction of $R_k, FG$ is presented as following. $t_2$ has no constraint inequalities and $t_3$ has a constraint inequality $IE_{k3} = -v_3 < 0$. Thus $R_k, FG = \{-v_3 < 0\}$ after the execution of $t_2$ and $t_3$. The constraint inequalities of transition $t_6$ include $IE_{k2} = -v_3 < -5$ and $IE_{k3} = p_2 - p_1 < 0$. $IE_{k2}$. $CRV_{k2} = (0, 0, -1, 0, 0, 0)$ can be separated into an unit vector $\vec{a}_1 = (0, 0, -1, 0, 0, 0)$, that is, $CRV_{k2} = \vec{a}_1$. In addition, according to $MergeIE$, we can merge $IE_{k2}$ into $IE_{k1}$ and the merging result “$-v_3 < -5$” should be appended into $R_k, FG$. For $IE_{k3}$, it will be appended into $R_k, FG$ directly. As a result, $R_k, FG = \{-v_3 < -5, p_2 - p_1 < 0\}$.

4 A PEAT-based Method for Executable Test Sequences Generation (PETSG)

This section describes the PETSG method. Our method can generate an executable test sequence and a suite of the related variables constraints through expanding a PEAT tree, without the initial value assignment. We assume that each transition of a PEAT $TT^{PEAT}$ is executable under a specific variables constraints. Hence the problem of an executable test sequence generation can be reformulated to search a transition path satisfying a specific coverage criteria (such as D-P coverage) in the $TT^{PEAT}$.

The PETSG consists of three main steps: (1) selecting an initial state as the root node of a $TT^{PEAT}$ and specifying a test coverage criteria; (2) expanding the $TT^{PEAT}$ level by level using breadth-first search (BFS) and updating the $FG_i$ of the current node $N_i$ to derive variables constraints relationship; (3) checking whether $N_i, R_i$ satisfies the specific coverage criteria. The termination condition of the procedure is that $R_i$ covers all transitions to be tested regarding to the coverage criteria at least once.
For convenience, some related terms are given to simplify the representation. $TCTS$ is a target coverage transitions set which contains all transitions to be covered in a conformance testing. $N_{\text{init}}$ is the initial extended state configuration used as the root node. $N_{\text{current}}$ represents the current node. $t_k$ is the outgoing transition of $N_{\text{current}}$. $SC_{\text{Queue}}$ is a node queue storing all nodes in the $TT^{\text{PEAT}}$. $R_{\text{final}}$ and $FG_{\text{final}}$ record the generated executable test sequence and its variable constraint conditions, respectively. The PETSG algorithm is described in detail as following.

Step 1 Push $N_{\text{init}}$ into $SC_{\text{Queue}}$;

Step 2 If $SC_{\text{Queue}}$ is not empty, then pop the head element of $SC_{\text{Queue}}$ and assign it to $N_{\text{current}}$; otherwise, the algorithm is terminated;

Step 3 If $N_{\text{current}}.R$ covers all transitions in $TCTS$, then $R_{\text{final}} := N_{\text{current}}.R$, $FG_{\text{final}} := N_{\text{current}}.FG$ and terminate the algorithm; otherwise, go to Step 4;

Step 4 For each outgoing transition $t_k$ of $N_{\text{current}}.S_i$:
   (1) Standardize the variable constraint inequalities of $t_k$;
   (2) Construct the tail node $N_{\text{tail}}$ for $\text{TailState}(t_k)$;
   (3) Call the $\text{MergeIE}$ algorithm (described in Section 3.2) to merge the $N_{\text{tail}}.FG$;
   (4) Compute the $N_{\text{tail}}.FG$ to judge whether there is a solution for its variable constraints to trigger the $N_{\text{tail}}.R$; if it has a feasible solution, then push $N_{\text{tail}}$ into $SC_{\text{Queue}}$;

Step 5 Go to Step 2.

PETSG selects an initial extended state configuration $N_{\text{init}}$ as the root node to expand a $TT^{\text{PEAT}}$. When a new node $N_{\text{current}}$ is constructed in the exploration of the $TT^{\text{PEAT}}$, we need check whether its $R$ covers all the transitions in $TCTS$. If so, $N_{\text{current}}.R$ is a generated executable test sequence and $N_{\text{current}}.FG$ is the related variable constraints. If not, the Matlab toolbox called $\text{Symbolic Math}$ is used to solve the inequalities in $N_{\text{current}}.FG$ such as Step 4(4). If the $N_{\text{current}}.FG$ has a feasible solution for variable constraints regarding to its $R$, $N_{\text{current}}$ is pushed into $SC_{\text{Queue}}$ to continue expanding the $TT^{\text{PEAT}}$ until a node satisfying the test coverage is found. Otherwise, we discard the node $N_{\text{current}}$.

5 Experimental Results

In this section, two classical EFSMs (Simplified Inres EFSM [7] and Network Monitor EFSM [2]) are used as the experimental examples to validate the effectiveness of our proposed approach. We choose D-P coverage criteria to design our experiments, as the condition conflicts in a D-P path is the essential cause for the unexecutable problem. For Simplified Inres EFSM, we set the root node $N_{\text{init}} = \{\text{Disconnect}, (), (), ()\}$. Its $TCTS$ and results are shown in Table 1.

In Table 1, the second column is the test coverage represented as a pair of D-P transitions $< t_i, t_j >$ stored in a $TCTS$. $t_i$ has D-use of a variable $v_i$ and $t_j$ has a P-use of the $v_i$, respectively. Simplified Inres EFSM has twenty D-P pairs, while the results in Table 1 just show seven typical pairs due to the limited space. The third column is the list of executable test sequences derived
Table 1: Results for Simplified Inres EFSM

<table>
<thead>
<tr>
<th>NO.</th>
<th>Test Coverage</th>
<th>Generated test sequences</th>
<th>Constraints of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&lt; t_6, t_9 &gt;$</td>
<td>$t_2, t_6, t_9$</td>
<td>$NULL$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; t_6, t_{15} &gt;$</td>
<td>$t_2, t_6, t_9, t_{15}$</td>
<td>$counter==4$</td>
</tr>
<tr>
<td>3</td>
<td>$&lt; t_{10}, t_{12} &gt;$</td>
<td>$t_2, t_6, t_9, t_{12}, t_{10}$</td>
<td>$counter &lt; 4$</td>
</tr>
<tr>
<td>4</td>
<td>$&lt; t_{10}, t_{15} &gt;$</td>
<td>$t_2, t_6, t_9, t_{10}, t_{12}, t_{15}$</td>
<td>$counter==3$</td>
</tr>
<tr>
<td>5</td>
<td>$&lt; t_{11}, t_{14} &gt;$</td>
<td>$t_2, t_6, t_9, t_{11}, t_9, t_{14}$</td>
<td>$number &lt; 4$</td>
</tr>
<tr>
<td>6</td>
<td>$&lt; t_{12}, t_{13} &gt;$</td>
<td>$t_2, t_6, t_9, t_{12}, t_{13}$</td>
<td>$counter &lt; 3$</td>
</tr>
<tr>
<td>7</td>
<td>$&lt; t_{14}, t_{13} &gt;$</td>
<td>$t_2, t_6, t_9, t_{13}, t_{14}$</td>
<td>$counter &lt; 4 &amp;&amp; number &lt; 4$</td>
</tr>
</tbody>
</table>

by our proposed approach. The last column gives the feasible constraints of variables regarding to each generated test sequence. If a generated test sequence has no condition conflict, we denote its corresponding conditions of constraints as $NULL$, such as the first row in Table 1.

For Network Monitor EFSM, we assume that the initial root node is $N_{init} = \{Idle, , \}$, such as the first row in Table 1.

Table 2: Results for Network Monitor EFSM

<table>
<thead>
<tr>
<th>NO.</th>
<th>Test Coverage</th>
<th>Generated test sequences</th>
<th>Constraints of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&lt; t_3, t_8 &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_8$</td>
<td>$number-no_of_segment &lt; -1$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; t_4, t_7 &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_7$</td>
<td>$number-no_of_segment == -1$</td>
</tr>
<tr>
<td>3</td>
<td>$&lt; t_4, t_8 &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_8$</td>
<td>$number-no_of_segment &lt; -1$</td>
</tr>
<tr>
<td>4</td>
<td>$&lt; t_8, t_7 &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_8, t_7$</td>
<td>$number-no_of_segment &lt; -1 &amp;&amp; number-no_of_segment == -2$</td>
</tr>
<tr>
<td>5</td>
<td>$&lt; t_9, t_{10} &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_9, t_{10}$</td>
<td>$counter-blockbound &lt;= -1$</td>
</tr>
<tr>
<td>6</td>
<td>$&lt; t_9, t_{11} &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_9, t_{11}$</td>
<td>$-counter+blockbound &lt; 1$</td>
</tr>
<tr>
<td>7</td>
<td>$&lt; t_9, t_{12} &gt;$</td>
<td>$t_1, t_2, t_3, t_4, t_9, t_{12}$</td>
<td>$counter-blockbound &lt;= -1$</td>
</tr>
</tbody>
</table>

Obviously, for all the test sequences listed in Table 2, we can find a group of input test data to trigger the corresponding test sequence according to its variable constraints. For example, for the generated test sequence in the first row, we can assign the variables $number$ and $no\_of\_segment$ according to the variable constraint "$number - no\_of\_segment < -1"$. Then we can generate an executable test sequence "$t_1, t_2, t_3, t_4, t_8$" and its input data such as $(2,0,0,4,0)$ corresponding to variables $number$, $counter$, $SDU$, $no\_of\_segment$, $blockbound$, respectively.

6 Conclusions

This paper proposed a novel approach for generating executable test sequences using a parametric executable analysis tree. The proposed method defines a VCEG to formal description of the
overview of the conflict relationship among transitions. Based on the VCEG, a PEAT is constructed to develop the executable test sequences for a specific coverage criteria. Compared with the traditional executable analysis methods, our approach has two characteristic features: (1) an executable test sequence can be automatically generated without the initial input variables assignment; (2) the feasible constraints of initial variables regarding to the generated test sequences are derived during test generation. Future work may focus on introducing a heuristic method for exploring the PEAT tree to direct the test sequences generation instead of the BFS exploration.

Acknowledgement

We would like to appreciate the anonymous reviewers for their constructive comments and suggestions to improve the quality of this paper. This work is partially supported by the National Natural Science Foundation of China (No. 61101111), Science Foundation of Zhejiang Sci-Tech University (ZSTU) under Grant No. 1004839-Y.

References