International Journal of Control and Automation
Vol. 9, No. 5 (2016), pp. 207-218
http://dx.doi.org/10.14257/ijca.2016.9.5.20

A Test Generation Method for EFSM-based Protocols Using the Transitions Feasibility Estimation

Ting Shu[1], Tiantian Ye[1], Xuesong Yin[2], Jinsong Xia[1]

[1]*School of Information and Science Technology, Zhejiang Sci-Tech University, Hangzhou, 310018, China*

[2] *Department of Computer Science & Technology, Zhejiang Radio & TV University, Hangzhou, 310030, China*

shuting@zstu.edu.cn

Abstract

Although extensive studies have been done on the protocol conformance testing based on an extended finite state machine (EFSM) model, the automatic generation of feasible test sequences is still a challenging task. A generated test sequence may be infeasible, due to the conflicts among the transition conditions and context variables of EFSMs. This paper proposed a test sequence generation approach for EFSM-based protocols conformance test by using the transition feasibility estimation. Firstly, our method generates candidate transition paths from a directed graph G which is derived from the EFSM model under testing ignoring all the predicates; Then, we designed a fitness function to guide the test generation with a trade-off among path feasibility, coverage criterion and path length. Finally, an adaptive exploration algorithm is developed to generate executable test sequences through expanding CPs. An experiment was designed to validate the effectiveness of the proposed method with two classic EFSM models. The experimental results show that our approach is more effective by comparing it to the TEA method based on breadth-first search (BFS).

Keywords*: Conformance testing, Adaptive exploration, EFSM, Test sequence, TEA method*

1. Introduction

Protocol implementations are the fundamental elements of network communication, which plays a key role in running and performance of network [1]. Protocol conformance testing is an experimental activity to determine whether the protocol implementation is conformance to its formal specification or not. The implementation under testing (IUT) is commonly regarded as a black box. Testers can only use a suite of external inputs and corresponding observable outputs to give the test verdicts. These inputs and outputs are called test sequences derived from the protocol specification. Unfortunately, generating test sequences satisfying the specific coverage requirements manually is a time-consuming work. Model-based test sequence generation greatly simplifies the protocol testing work and shortens the development time of protocols.

Two most popular formal models for representing protocol specifications are FSMs [2] and EFSMs [3]. An ocean of FSM-based test generation methods [4-6] have been proposed in the last decades. The EFSM model extends the FSM model to enhance its modeling capability concerning data part of a protocol with variables and predicates. In EFSM-based protocol testing, the test sequences are generally denoted as transition paths. Consequently, the existing conflicts among variables and predicates in transitions lead to infeasible test sequences. Thus, the traditional test sequences generation methods for FSM are not suitable for testing protocol specified in EFSMs [7]. In this paper, we focus on deriving feasible protocol conformance testing sequences based on the EFSM model. A favorable method for test sequence generation should consider the following three objectives: (1) Generate test sequences satisfying a test coverage criterion; (2) Guarantee the executability of the generated test sequences; (3) Minimize the length of the test sequences, thereby reducing test cost.

In recent years, genetic algorithm (GA) has been used to generate executable test sequences [8]. Its working procedure is divided into two stages: the first phase generates transition paths (TPs) to meet the specified coverage criterion; the second phase finds input data to fire the TPs produced in the first phase. However, these GA-based methods can’t guarantee the existence of input data which can fire candidate TPs. In contrast, test sequence generation methods using transition executability analysis (TEA) [9] can ensure the executability of the generated test sequence in only single step. What's more, the restriction conditions of EFSM models under test for the TEA methods are less than GA-based ones. However, TEA-based methods have also a deficiency: exploration in a huge TEA tree is prone to a state explosion problem.

In order to alleviate the state explosion problem, this paper presents a new heuristic method for generating executable test sequences based on transitions feasibility estimation. Our approach makes a trade-off among path feasibility, coverage criterion and path length. We also conduct experimental studies to investigate the performance of our proposed method. Experimental results suggest that our method can effectively decrease the probability of the state explosion problem and improve the efficiency of test sequences generation.

In short, the main contributions of this paper are as follows:

1. A new test generation approach based on estimation of feasibility is presented to find a shorter test sequence that meets the test criterion.

2. We introduce a new fitness function to make a trade off among path feasibility, TP length, and coverage criterion.

3. An experiment is designed with two classic EFSM models. The experimental results show that the approach is more effective by comparing it to BFS.

The rest of the paper is organized as follows. Section 2 provides some background information. Section 3 describes the detail of test generation method for EFSM based on estimation of the transition feasibility. Empirical results are discussed in Section 4. Section 5 summarizes the related work of test sequence generation. Conclusion and prospective are described in Section 6.

2. Preliminaries

An EFSM can be described as an eight-tuple <S, s_0, I, P, A, O, V, T>, where S is a non-empty finite set of logical states of the EFSM; s_0 is the initial state; I is the finite set of input; P is the set of predicates that operate on variables; A is the set of actions that operate on variables; O is the finite set of output; V is the set of variables, $V = V_i \cup V_c$, where V_i and V_c represent the input variables and the context variable, respectively; T is the set of transitions, $T = S \times V \times I \rightarrow S \times V \times O \times A$; A transition t can be represented by the six-tuple ($s_s, i_t, p_t, a_t, o_t, s_e$), where s_s is the start state of t, s_e is the end state of t, i_t is the input and o_t is output. A transition can be fired, when the condition part p_t is satisfied after receiving a given input i_t. When t is executed, the corresponding action $a_t \in A$ is executed and logical state is switched from the state s_s to the s_e.

To facilitate the description of the proposed approach, we will take the EFSM M_1 [10] shown in Figure 1 as an example throughout this paper. In M_1, S = {*Disconnect*, *Waiting*, *Connected*, *Sending*, *Block*}, V_c = {*counter*, *input*}, V_i = {*optional*, *SDU*, *number*, *block*, *olddata*, *NUM*}. If, for example, M_1 is at state S_2, the value of *counter* satisfies the guard of t_4 (*counter*<4) then t_4 is fired, the operation (*counter* =: *counter* +1;) is executed, and the operation update the value of *counter*

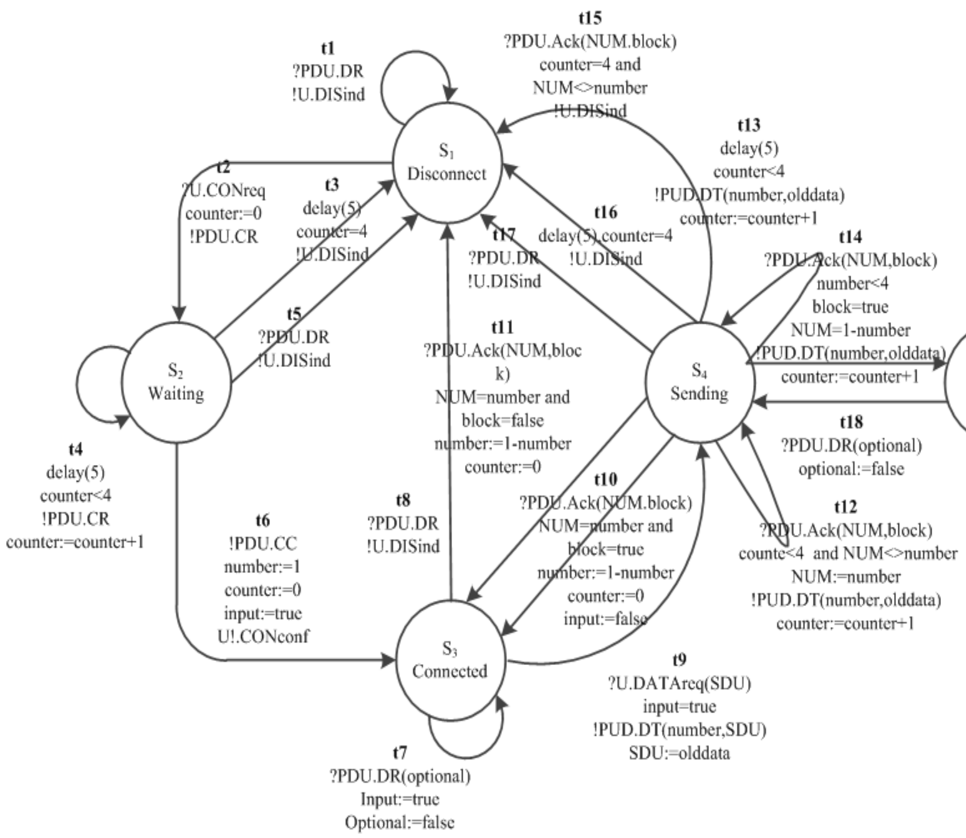


Figure 1. Inres Protocol as an EFSM (M_1)

Before providing a detailed description of our method, we introduce the following definitions:

Definition 1. State configuration (*SC*) can be represented as a tuple $(s_i, \overrightarrow{v_i}) \in S \times V$, where s_i is the current state and $\overrightarrow{v_i}$ is the current value vector of variables, respectively. *SC*= s_i (v_1, v_2, ..., v_n).

Definition 2. Transition configuration (*TC*) can be represented as a tuple $(t_j, \overrightarrow{v_j}) \in T \times V$, where t_j is the name of current transition, $\overrightarrow{v_j}$ is same with the set of variables of s_e. *TC*= t_j (v_1, v_2, ..., v_n).

Definition 3. A transition path (TP) of length *n* is a transition sequence that consists of *n* continuous transitions end to end, TP= t_1, t_2, ... , t_n. TP is feasible (an FTP) if and only if the predicates *p* of each transition t_i can be satisfied, where $1<i<n$.

Definition 4. A path (t_1, ..., t_n) is a DP-path in respect to a variable *v*, if t_1 has a definition-use of *v*, t_n has a predicate-use of *v* and the path is a def-clear path regarding *v*. Then, t_1, t_n in this path is a definition-use pair. DP represents the definition-p-use transition pair in an EFSM model. We use the value of *v*(*DP*) to evaluate the feasibility of a specific transition path.

3. The Proposed Approach

In this section, we present a new test sequence generation approach based on feasibility estimation of transitions. The proposed approach adaptive searches a relatively short FTP regarding the given test criterion for the EFSM-specified protocol. The proposed method descried in this paper consists of two procedures and the flowchart is shown in the Figure 2.

The procedure above the loop generates candidate paths (CPs) to satisfy the transition coverage criteria. To estimate of feasibility of a TP, all DPs in this TP are found and *v*(*DP*) is assigned to each DP, with the *v*(*DP*) depending on the relevant assignment and guard. Before CPs generation, definitions and guards were classified and for each possible combination of guard and definitions type, a pre-determined *v*(*DP*) assigned, which are given in Section 3.1.

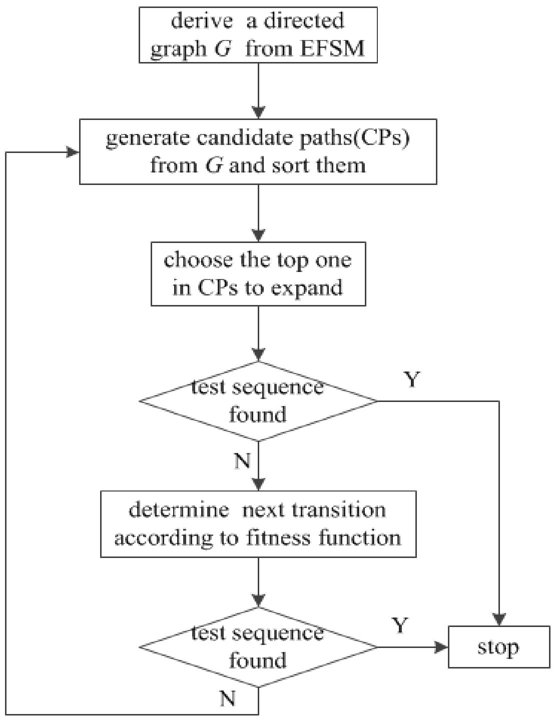

derive a directed graph G from EFSM
generate candidate paths(CPs) from G and sort them
choose the top one in CPs to expand
test sequence found
Y
N
determine next transition according to fitness function
test sequence found
Y
stop
N

Figure 2. Flowchart of Test Sequence Generation Based on Transitions Feasibility Estimation

The procedure in the loop generates test sequence by expanding the CPs generated in Section 3.2. The fitness function used to guide the search is described in detail in Section 3.3.

3.1. DP Classification

The type of definitions and guards that occur in DPs determine the probability that a path from definition transition to p-use transition can be executed. The feasibility of the TP is evaluated by *v*(*DP*). Each definition can be normalized to a form of *v*:= *EP*, where $v \in V_c$, *EP* is an expression. And each p-use can be normalized to a form of $v\Theta EP_r$, where $v \in V_c$, EP_r is an expression and Θ is the operator.

Var(*EP*) and *Var*(EP_r) represent the variables that appear in expression *EP* and EP_r, respectively. According to the types of *Var*(*EP*) and *Var*(EP_r), the DPs through EFSM can be classified, and the resultant *v*(*DP*) used in this work are shown in the Table 1.

Table 1. The Corresponding *v*(*DP*) Used in this Work

Type of EP and EP_r	operator of guard			
	$\neq$	$=$	$\leq$ or $\geq$	$<$ or $>$
$Var(EP) \in c \cup \varnothing$ and $Var(EP_r) \in c \cup \varnothing$	0 if false and 100 otherwise	0 if false and 100 otherwise	0 if false and 100 otherwise	0 if false and 100 otherwise
$Var(EP) \in c \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup \varnothing$	12	24	12	18
$Var(EP) \in c \cup \varnothing$ and $Var(EP_r) \in c \cup V_c \cup \varnothing$	24	60	48	36
$Var(EP) \in c \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup V_c \cup \varnothing$	18	48	36	30
$Var(EP) \in c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup \varnothing$	12	24	12	18
$Var(EP) \in c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup \varnothing$	2	8	4	6
$Var(EP) \in c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_c \cup \varnothing$	8	20	12	16
$Var(EP) \in c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup V_c \cup \varnothing$	6	16	8	12
$Var(EP) \in c \cup V_c \cup \varnothing$ and $Var(EP_r) \in c \cup \varnothing$	24	60	48	36
$Var(EP) \in c \cup V_c \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup \varnothing$	4	16	8	12
$Var(EP) \in c \cup V_c \cup \varnothing$ and $Var(EP_r) \in c \cup V_c \cup \varnothing$	16	40	24	32
$Var(EP) \in c \cup V_c \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup V_c \cup \varnothing$	12	30	16	24
$Var(EP) \in c \cup V_c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup \varnothing$	18	48	36	30
$Var(EP) \in c \cup V_c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup \varnothing$	3	12	6	8
$Var(EP) \in c \cup V_c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_c \cup \varnothing$	12	30	20	24
$Var(EP) \in c \cup V_c \cup V_i \cup \varnothing$ and $Var(EP_r) \in c \cup V_i \cup V_c \cup \varnothing$	8	24	12	20

For example, in Table 1, $Var(EP) \in c \cup \varnothing$ is used to indicate that EP only includes constants, where c is a constant. $Var(EP) \in c \cup V_i \cup \varnothing$ represents EP only includes constants and input variables but no context variables.

3.2. Candidate Paths Generation

Ignoring all predicates in transitions, a directed graph G can be derived from the EFSM model under testing. To illustrate how our method works, we consider the M_1 shown in Figure 1. First, we generate the shortest TPs which are more likely to be executed to satisfy transition coverage criteria. G is conflict-free since it does not contain any guard. So we can use the classical BFS algorithm with a pruning strategy to generate shortest TPs that meet coverage requirement. However, the traditional state configuration pruning strategy [9] may not generate TPs that we need. U denotes the transition coverage target. Suppose that $U = \{t_1, t_2, t_3, t_4, t_5\}$, for M_1, BFS generates an expanding tree rooted in a given state configuration of M_1 when using the pruning strategy. Figure 3 shows part of BFS tree, which is rooted in the initial state configuration of M_1.

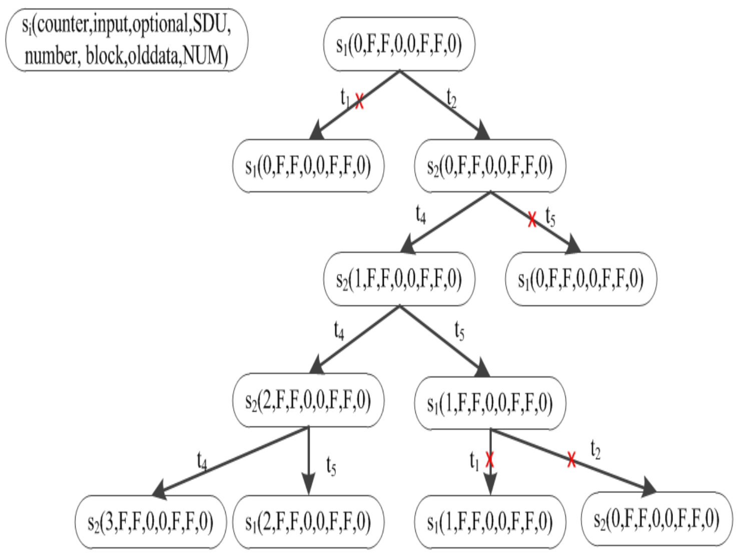


Figure 3. Part of a BFS Tree of the M_1

In Figure 3, t_1 is cut off for its tail state configuration existed in the current TPs, and thus these two TPs are abandoned. So *SC* pruning strategy does not work well in this case. To address this problem, we adopt *TC* pruning strategy instead of *SC* in the process of BFS.

If we can estimate the feasible probability in advance, it will greatly improve the performance of test sequence generation. Our method generates one or more short TPs and then sorts them with the aim to direct the choice of TPs towards those to feasible as CPs. For this reason, a path with more definition-p-use transition pairs has higher probability that it is infeasible. Thus, it is used to estimate the feasibility of a given path, where n is the number of DP in the path. The smaller sum of $v(DP)$ on the path $\sum_{i=1}^{n} v(DP_i)$ states that the path has better feasible probability. We sort shortest TPs by $\sum_{i=1}^{n} v(DP_i)$ in ascending order.

Suppose that U={t$_1$,t$_2$,t$_3$,t$_4$,t$_5$}, we can derive the candidate paths CPs={TP$_1$=t$_1$,t$_2$,t$_3$,t$_2$,t$_4$,t$_5$, TP$_2$=t$_1$,t$_2$,t$_4$,t$_3$,t$_5$,t$_2$, t$_4$, ...}.Given the initial state configuration SC_0 and target transition set U for an EFSM, Algorithm 1 generates candidate paths shown in the Figure 4.

```
Algorithm 1 candidate paths generation ($Gen\_CPs(SC_0, U)$)
1:  input:EFSM model under testing, $SC_0$ and $U$;
2:  output:candidate paths satisfying transition coverage;
3:  initial: $CPs$:=null; int $v(DP)$; $F$:=null; $Found$:=false; $OTs$:=null;
4:  begin
5:  derive graph $G$ without predicates from $M_1$; put $SC_0$ into the queue $F$;
6:  while $F \neq \emptyset \wedge U \neq \emptyset$ do
7:     remove the first node $S_f$ in the queue $F$;
8:     if $Found$==true and $TC.level > Found\_level$ then
9:        sort $CP_i$ by $\sum_{i=1}^{n} v(DP_i)$ in ascending order; return $CPs$;
10:    end if
11:    put all outgoing transitions of $S_f$ into set $OTs$;
12:    while $U \neq \emptyset \wedge OTs \neq \emptyset$ do
13:       if $OTs == \emptyset$ then
14:          $F$.remove($S_f$); break;
15:       end if
16:       get the first transition $t_f$ in $OTs$; $t_f.level$:=$S_f.level$+1;
17:       if the $TC$ of $t_i$ has existed in current $FTP$ then
18:          remove $t_f$ from $OTs$;
19:          if $OTs == \emptyset$ then
20:             $F$.remove($S_f$); break;
21:          else
22:             get first transition $t_f$ in $OTs$;
23:          end if
24:       end if
25:       $CP := CP \circ t_f$;
26:       if $CP$ has covered all the transitions in $U$ then
27:          add $CP$ to the $CPs$; $Found$:=true; $Found\_level := t_f.level$;
28:       end if
29:       remove the first out transition $t_f$ form $OTs$; (tail state of $t_f$).$level$:=$t_f.level$; add the tail $SC$ of $t_f$ to the $F$;
30:    end while
31: end while
```

Figure 4. Algorithm of Candidate Paths Generation

3.3. Test Sequence Generation

Once CPs have been generated, we should check their feasibility. Firstly, we choose the first one in CPs to expand, and traverse each transition t_i in the sequential order in this CP. If t_i is feasible, we continue to traverse the following transitions in the CP_i. We have to adjust the original CP_i to construct new $CP_i^{'}$, otherwise. In the above example, the CP_1= t_1, t_2, t_3, t_2, t_4, t_5, is not a FTP since after executing t_1, t_2, *counter* is 0, less than 4 and t_3 requires *counter* to be equal to 4. Therefore, we should stop expanding the remaining transitions in this CP. And now, the current test sequence is t_1, t_2 and the current state is s_2, which is s_e of t_2, and $U = \{t_3, t_4, t_5\}$. We have to choose another executable outgoing transition of s_2 instead of t_3.

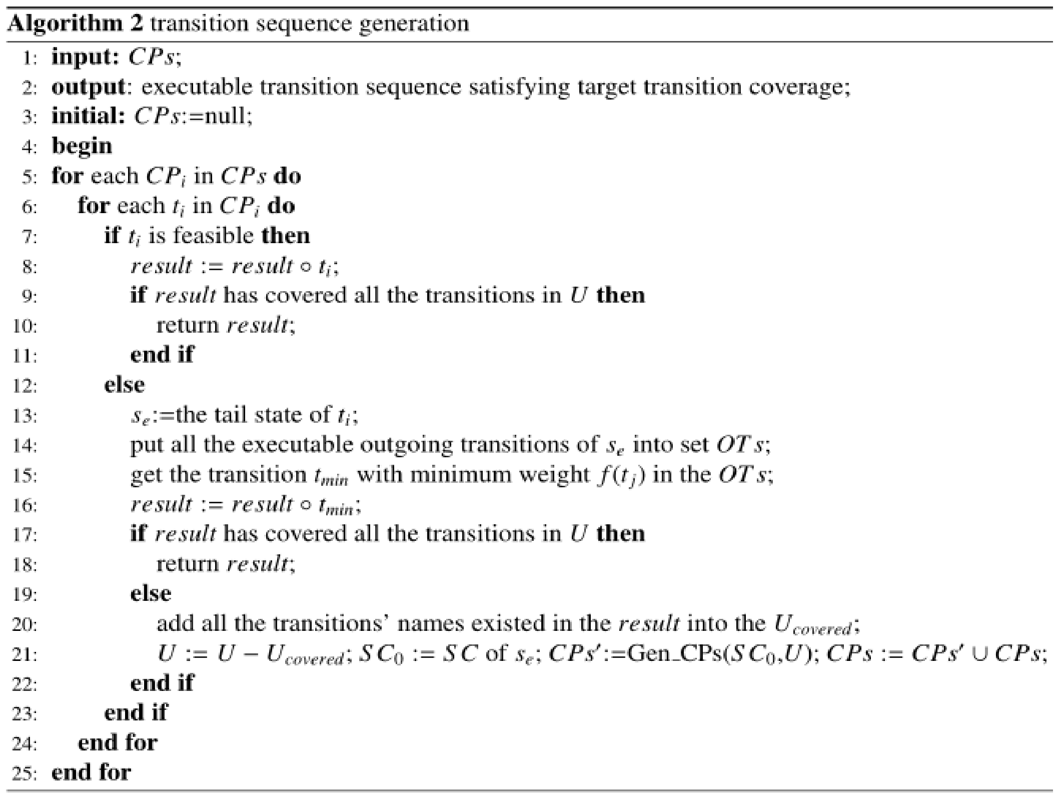
Intend to find the test sequence, which has shorter path length and goodness of feasibility to cover all transitions of EFSM model from U, we present the metric as follows:

$$f(t_j) = n \cdot \sum_{i=1}^{n} \frac{v(PD_i)}{\frac{\alpha}{|TP|}} \tag{1}$$

In the formula (1), n is the number of transitions which are uncovered in the U. $|TP|$ is the length of TP that has traversed. α is a constant which is used to tune the weight of path length in the metric. $v(PD_i)$ represents the value of PD that constructed by t_j and transitions that have not been covered in U. The new algorithm selects the transition with minimum f as next transition to expand.

t_5 is our choice for $f(t_4)>f(t_5)$, thus current test sequence is t_1, t_2, t_5, U={t_3, t_4} and initial SC is s_1 (*counter*, *input*, *optional*, *SDU*, *number*, *block*, *olddata*, *NUM*). $Gen\_CPs(SC_0,U)$ is used to generate new CPs. The above procedure is repeated until all the transitions in the U are covered and the generated test sequence is executable.

Test generation method based on estimation of feasibility is described in the Algorithm 2 which is shown in Figure 5.



```
Algorithm 2 transition sequence generation
input: $CPs$;
output: executable transition sequence satisfying target transition coverage;
initial: $CPs$:=null;
begin
for each $CP_i$ in $CPs$ do
  for each $t_i$ in $CP_i$ do
    if $t_i$ is feasible then
      $result := result \circ t_i$;
      if $result$ has covered all the transitions in $U$ then
        return $result$;
      end if
    else
      $s_e$:=the tail state of $t_i$;
      put all the executable outgoing transitions of $s_e$ into set $OTs$;
      get the transition $t_{min}$ with minimum weight $f(t_j)$ in the $OTs$;
      $result := result \circ t_{min}$;
      if $result$ has covered all the transitions in $U$ then
        return $result$;
      else
        add all the transitions' names existed in the $result$ into the $U_{covered}$;
        $U := U - U_{covered}$; $SC_0 := SC$ of $s_e$; $CPs'$:=Gen_CPs($SC_0$,$U$); $CPs := CPs' \cup CPs$;
      end if
    end if
  end for
end for
```

Figure 5. Algorithm of Transition Sequence Generation

4. Empirical Results and Discussion

In this section, we outline experiments in which the proposed technique was used to generate test sequences from two EFSMs and the results compared with BFS approach.
Two EFSMs were used in the experiments: M_1 [10] and M_2 [11]. In order to evaluate the effectiveness of the proposed approach, we consider two factors: the count of nodes traversed and the length of the test sequences. The set of target transitions to be covered in M_1 and M_2 are shown in the Table 2.
To generate the test sequence includes all the transitions in the U, everytime we choose the first CP in the CPs to expand. The CPs which are used to expand in our approach across each of the EFSMs under study are listed in Table 3.

Table 2. The Set of Target Transitions to be Covered in M_1 and M_2.

NO.	M_1	NO.	M_2
1	U={t_1,t_2,t_3,t_4,t_5}	1	U={t_1,t_2,t_3,t_4,t_5,t_6}
2	U={t_1,t_2,t_4,t_5,t_6,t_7,t_8,t_9}	2	U={t_1,t_2,t_3,t_4,t_5,t_6,t_7}
3	U={t_6,t_7,t_9,t_{11},t_{12},t_{13},t_{15}}	3	U={t_1,t_2,t_3,t_4,t_5,t_6,t_8,t_9,t_{10},t_{11}}
4	U={t_6,t_7,t_9,t_{11},t_{12},t_{14},t_{16},t_{18}}	4	U={t_1,t_2,t_3,t_4,t_5,t_6,t_8,t_9,t_{10},t_{11},t_{12},t_{13}}

Table 3. The CPs that are Expanded to Generate Test Sequences in M_1 and M_2

M_1		M_2	
U	CP	U	CP
1	$CP_1=t_1,t_2,t_3,t_2,t_4,t_5$ $CP_2=t_2,t_4,t_3$ $CP_3=t_4,t_3$	1	$CP_1=t_1,t_2,t_3,t_5,t_4,t_6$
2	$CP_1=t_1,t_2,t_5,t_2,t_4,t_6,t_7,t_9,t_{10},t_8$ $CP_2=t_{10},t_8$ $CP_3=t_8$	2	$CP_1=t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_7$ $CP_2=t_8,t_7$
3	$CP_1=t_6,t_7,t_9,t_{11},t_9,t_{12},t_{13},t_{15}$ $CP_2=t_{11},t_9,t_{13},t_{15}$ $CP_3=t_{15}$ $CP_4=t_{15}$ $CP_5=t_{15}$ $CP_6=t_{15}$	3	$CP_1=t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_8,t_9,t_{10},t_9,t_{11}$ $CP_2=t_9,t_{11}$ $CP_3=t_9,t_{11}$
4	$CP_1=t_6,t_7,t_9,t_{11},t_9,t_{12},t_{14},t_{18},t_{16}$ $CP_2=t_{11},t_9,t_{14},t_{18},t_{16}$ $CP_3=t_{13}$ $CP_4=t_{13}$ $CP_5=t_{13}$	4	$CP_1=t_1,t_2,t_3,t_4,t_8,t_9,t_{10},t_9,t_{11},t_{13},t_{14}$ $CP_2=t_9,t_{11},t_{13},t_{14}$ $CP_3=t_9,t_{11},t_{13},t_{14}$

Table 4 shows the results for the two EFSMs. Compared to the BFS algorithm, the average number of nodes which are searched in the process of test sequence generation in the new algorithm is reduced by 45.5%. Figure 6 clearly indicates the superiority of new technique over the BFS with respect to the number of states explored. Data are also provided in Table 4.

Table 4. Comparison of the Number of Traverse Nodes

M_1	BFS	Our method	M_2	BFS	Our method
1	1204	141	1	18	25
2	5770	5046	2	105	79
3	3536	1470	3	23723	4012
4	6895	3540	4	19351	3373

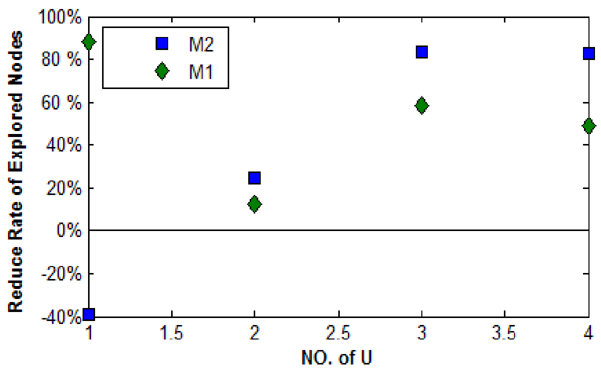


Figure 6. Reduce Rate of Explored Nodes

Our approach performs almost as well as BFS with regard to the length of test sequence which is consistent with our observations from Table 5 and Table 6.

Table 5. Comparison of the Length of Test Sequences Generated

M_1	BFS	Our method
1	$t_1,t_2,t_5,t_2,t_4,t_4,t_4,t_3$	$t_1,t_2,t_5,t_2,t_4,t_4,t_4,t_3$
2	$t_1,t_2,t_5,t_2,t_4,t_6,t_7,t_9,t_{12},t_{11},t_8$	$t_1,t_2,t_5,t_2,t_4,t_6,t_7,t_9,t_{12},t_{11},t_8$
3	$t_6,t_7,t_9,t_{12},t_{11},t_9,t_{13},t_{13},t_{13},t_{13},t_{15}$	$t_6,t_7,t_9,t_{12},t_{11},t_9,t_{13},t_{13},t_{13},t_{13},t_{15}$
4	$t_6,t_7,t_9,t_{12},t_{11},t_9,t_{13},t_{12},t_{13},t_{14},t_{18},t_{16}$	$t_6,t_7,t_9,t_{12},t_{11},t_9,t_{14},t_{18},t_{13},t_{13},t_{13},t_{16}$

Table 6. Comparison of the Length of Test Sequences Generated

M_2	BFS	Our method
1	t_1,t_2,t_3,t_5,t_4,t_6	t_1,t_2,t_3,t_5,t_4,t_6
2	$t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_8,t_7$	$t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_8,t_7$
3	$t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_8,t_9,t_{10},t_9,t_{10},t_9,t_{10},t_9,t_{11}$	$t_1,t_2,t_3,t_5,t_4,t_6,t_4,t_8,t_9,t_{10},t_9,t_{10},t_9,t_{10},t_9,t_{11}$
4	$t_1,t_2,t_3,t_4,t_8,t_9,t_{10},t_9,t_{10},t_9,t_{10},t_9,t_{11},t_{13},t_{14}$	$t_1,t_2,t_3,t_4,t_8,t_9,t_{10},t_9,t_{10},t_9,t_{10},t_9,t_{11},t_{13},t_{14}$

5. Related Work.

Currently, there are many works regarding test generation methods for EFSM models [12-14]. Huang [11] proposed a test sequence generation method by constructing a TEA tree. The main idea of the method is deriving the shortest executable test sequence through expanding the TEA tree, which is rooted in a given state configuration in the breadth-first-search way. However it easily leads to the state explosion problem. Duale [15] introduced a method to generate tests for EFSM models which converts a class of EFSMs into EFSMs in which all TPs are feasible with detection and elimination of conflicts. It is hard to produce test cases that satisfy a test criterion expressed in terms of the original EFSM.

Kalaji [8] presented a novel fitness metric to estimate the feasibility of a path, which is evaluated by being used in a genetic algorithm to guide the search towards TPs that are likely to be FTPs. Lefticaru [16] used independent component-based fitness (ICF) for path data generation from EFSMs. This new approach takes the independent sub-paths into account and uses a global, evolutionary inspired search. However in these methods, the choice of initial candidate test sequences as the parents is usually random, thus it easily leads to blindness search in the subsequent process.

Li [17] applied symbolic execution in generating configuration-oriented executable test sequences from EFSM models. The inputs are given in in symbolic form, thus the values of the context variables are symbols. Expressions over these symbols are exploited to guide the derivation of the test sequences.

Shu [10] studied a test generation method using adaptive and heuristic methods. They analysis intrinsic characteristics of variables and predicates of transitions in a EFSM model and defined a adjacency transition dependence graph based on relationship among transitions. The approach expands a TEA tree with the heuristic guidance to ensure the feasibility of test sequences generated.

6. Conclusion and Prospective

In this paper, we proposed a new executable test sequence generation algorithm for EFSM-based protocol conformance test using the transitions feasibility estimation. Firstly, the new algorithm classifies DPs according to the type of variables they include. Then, it expands CPs which are derived from *G* by the adaptive exploration function to aid the generation of executable test sequence that satisfies a test criterion. The experimental results suggest that our approach can effectively alleviate the state explosion problem usually occurring in the BFS-based TEA test generation methods.

We use BFS with *TC* pruning strategy to generate CPs. Hence, the future research direction is to reduce the number of states explored in the process of CPs generation. For example, we can derive the CPs by using one of FSM-based methods presented in the literature [4-6].

Acknowledgements

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, Public Project of Zhejiang Province under Grant No. 2013C33087.

References

[1] Y. Fu and O. Kone, “Security and robustness by protocol testing”, IEEE Systems Journal, vol.8, no.3, **(2012),** pp.699-707.

[2] T. S. Chow, “Testing software design modeled by finite-state machines”, IEEE Transaction on Software Engineering vol.4, no.3, **(1978)**, pp.178-187.

[3] A. Cavalli, C. Gervy and S. Prokopenko, “New approaches for passive testing using an extended finite state machine specification”, Information and Software Technology, vol.45, no.12, **(2003)**, pp.837-852.

[4] A. V. Aho, A. T. Dahbura, D. Lee and M. U. Uyar, “An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours”, IEEE Transactions on Communications, vol.39, no.11, **(1991)**, pp.1604-1615.

[5] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli and N. Yevtushenko, “FSM-based conformance testing methods: a survey annotated with experimental evaluation”, Information and Software Technology, vol.52, no.12, **(2010)**, pp.1286-1297.

[6] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold and P. McMinn, “An orchestrated survey of methodologies for automated software test case generation’, Journal of Systems and Software, vol.86, no.8, **(2013)**, pp.1978-2001.

[7] R. Yang, Z. Chen, B. Xu, W. E. Wong and J. Zhang, “Improve the effectiveness of test case generation on efsm via automatic path feasibility analysis”, Proceedings of the 13th International Symposium on High-Assurance Systems Engineering, **(2011)**, pp.17-24.

[8] A.S. Kalaji, R.M. Hierons and S. Swift, “An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models”, Information and Software Technology, vol.53, no.12, **(2011)**, pp.1297-1318.

[9] C.M. Huang, M.S. Chiang and M.Y. Jang, “UIO$_E$: a protocol test sequence generation method using the transition executability analysis (TEA)”, Computer Communications, vol.21, no.16, **(1998)**, pp. 1462-1475.

[10] T. Shu, L.G. Liu and W.Q. Xu, “Adaptive Executable Test Sequences Generation from an Extended Finite State Machine’, Journal of Computer Research and Development, vol. 49, no. 6, **(2012)**, pp.1211-1219.

[11] C.M. Huang, M.Y. Jang and M.S. Chiang, “Executable EFSM-based data flow and control flow protocol test sequence generation using reachability analysis”, Journal of the Chinese Institute of Engineers, vol.22, no.5, **(1999)**, pp.593-615.

[12] W.E. Wong, A. Restrepo and B. Choi, “Validation of SDL specifications using EFSM-based test generation”, Information and Software Technology, vol.51, no.11, **(2009)**, pp.1505-1519.

[13] A.S. Kalaji, R.M. Hierons and S. Swift, “Generating feasible transition paths for testing from an extended finite state machine (EFSM)”, Proceedings of the 2009 ICST international conference on Software Testing Verification and Validation, **(2009)**, pp.230-239.

[14] K. Derderian, R.M. Hierons, M. Harman and Q. Guo, “Estimating the feasibility of transition paths in extended finite state machines”, Automated Software Engineering, vol.17, no.1, **(2010)**, pp.33-56.

[15] A.Y. Duale and M.U. Uyar, “A method enabling feasible conformance test sequence generation for EFSM models”, IEEE Transactions on Computers, vol.53, no.5, **(2004)**, pp.614-627.

[16] R. Lefticaru and F. Ipate, “An improved test generation approach from extended finite state machines using genetic algorithms”, Proceedings of the 10th international conference on Software Engineering and Formal Methods,**(2012)**, pp.293-307.

[17] S. Li, J. Wang, X. Wang and Z.C. Qi, “Configuration-oriented symbolic test sequence construction method for EFSM”, Proceedings of the 2th Annual International Computer Software and Applications Conference, **(2005)**, pp.13-18.

Authors

Ting Shu. He received the Ph.D. degree in Computer Science from Zhejiang University in 2010. He is now an associate professor with the School of Information Science and Technology, Zhejiang Sci-Tech University, 310018, Hangzhou, China. His current research interests include software testing and network protocol testing.

Tiantian Ye. She is a Master Degree candidate in Computer Science at Zhejiang Sci-Tech University, She received her B.S. degree in information and computer science from Zhejiang Sci-Tech University in 2013. Her research interests are mainly in software testing and protocol conformance testing.

Xuesong Yin. He received the Ph.D. degree in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2010. He is currently a Professor in the School of Information and Engineering, Zhejiang Radio and TV University, Hangzhou, China. His current research interests include machine learning, data mining, and pattern recognition.

Jinsong Xia. He is a Ph.D. candidate in Control Theory and Control Engineering at School of Mechanical Engineering & Automation and a lecturer at the School of Information Science and Technology, Zhejiang Sci-Tech University,310018, Hangzhou, China. He received his M.S. in computer science from Zhejiang University in 2006. His current research interests are mainly in software testing.