Information Sciences 370–371 (2016) 63–78

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A heuristic transition executability analysis method for generating EFSM-specified protocol test sequences

Ting Shu[a,*], Zuohua Ding[a], Meihwa Chen[b], Jinsong Xia[a]

[a] School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China
[b] Computer Science Department, University at Albany-State University of New York, Albany, NY 12222, USA

ARTICLE INFO

Article history:
Received 3 May 2016
Revised 12 July 2016
Accepted 25 July 2016
Available online 26 July 2016

Keywords:
EFSM
Conformance testing
Executable test sequences
Test generation

ABSTRACT

Automatic executable test sequence generation is a challenging task for protocol test based on the Extended Finite State Machine (EFSM) model. Transition executability analysis (TEA) can guarantee the executability of generated test sequences. However, as a reachability analysis technique, it often suffers from the state explosion problem. To remedy the situation, this paper proposes a heuristic transition executability analysis (HTEA) method for compressing the explored state space during test sequences derivation. In our approach, a transition feasibility guidance matrix is first constructed through extracting the inherent constraint relationship between each adjacent transitions in the EFSM under testing. Then, relying on the built matrix and related runtime feedback, we define an objective function to dynamically evaluate state nodes in the HTEA expanding tree. Finally, the proposed objective function is utilized to heuristically guide the state space traversal in the HTEA tree for executable test sequences generation. Consequently, the executable test generation problem can be reformulated into a multi-objective search problem based on a HTEA tree. Experimental results impressively show the potential of the novel method for avoiding the state explosion during test generation, compared with the classic TEA technique in the breath-first way.

1. Introduction

Undetected minor faults in the protocol implementation could lead to disastrous consequences. To guarantee the communication quality, protocol conformance testing plays an integral role in the protocol development lifecycle. It is an important experimental activity to validate if the protocol implementation under test (IUT) is consistent with its specification or not. In the protocol conformance testing, test sequences that can be described as a series of input/output pairs are usually generated and stimulated to observe the external behavior of the IUT under test. Manual test sequence generation is expensive and error-prone [7]. Thus, automating the generation of test sequences from model-based specifications has become a hot issue for both industry and academic research, in the field of protocol testing.

Finite State Machine (FSM) and Extended Finite State Machine (EFSM) have been widely used to formally describe the protocol specifications under test [25,28]. A protocol specification usually comprises of a control portion and a data portion. The FSM is only suitable for modeling the control portion. Since the EFSM model extends the FSM model by introducing

* Corresponding author.
E-mail addresses: shuting@zstu.edu.cn (T. Shu), zouhuading@hotmail.com (Z. Ding), mchen@albany.edu (M. Chen), js_xia@126.com (J. Xia).

http://dx.doi.org/10.1016/j.ins.2016.07.059
0020-0255/

context variables and parameters, it is enhanced to represent both control and data portions [13,24]. In the past decade, substantial test generation methods based on FSMs have been proposed such as UIO method [14], D method [17] and W method [26]. Unfortunately, when testing from EFSM, these developed approaches are unable to be directly utilized. The main obstacle is that some generated test sequences may be infeasible, due to the existence of variables and associated predicate guards in transitions. That is to say, in terms of certain context variables, condition conflicts between transitions limit the executablility of the generated test sequences. However, automatic feasible test sequence derivation for an EFSM is NP-hard and remains an open research problem [13].

Although transition executability analysis (TEA) technique [18] provides a promising solution to overcome the infeasibility of generated test sequences, it often suffers from the state explosion problem. The reason stems from the fact that the TEA is one of the reachability methods that generate test sequences by exploring a TEA tree's reachable state space. However, the state space of an expanding TEA tree for a specific IUT is usually too huge to be traversed exhaustively. Thus, the classic TEA by means of systematic state space search, such as the breath-first way, may lead to state explosion. To partially alleviate this problem, a pruning strategy for TEA tree is introduced to limit the search space to some extent. But the remaining search space is still large. In addition, in our previous works, we found that it brings about a new potential thread for test generation. That is, corresponding to some specific test coverage criteria, the application of an unsuitable pruning strategy may result in the failure of test generation.

In this paper, we present a heuristic transition executability analysis (HTEA) method for generating feasible test sequences from a class of EFSMs, which are normalized, fully executable and untrapped. These specific constraints of EFSMs are prerequisites for application of TEA-based methods [18]. The purpose of our proposed method is to significantly compress the explored state space for feasible test sequences derivation and to eventually avoid the state explosion problem. In addition to the reduction of explored state space, minimizing the length of a generated test sequence is another objective. It is obvious that exploring the state space in breath-first way tends to find the shortest path satisfying a specific test criterion. Therefore, the proposed method makes a trade-off between the two aims.

In the HTEA approach, we first construct a transition feasibility guidance matrix by extracting the inherent relationship between adjacent transitions from EFSM under test. Then, based on the constructed matrix and the run-time information towards the test coverage criterion, an objective function f_{obj} is defined and computed to heuristically guide the state space traversal for test generation. For a state s_i in the TEA tree, the value of $f_{obj}(s_i)$ represents an approximate estimation of the probability that there exists an optimal feasible path from s_i to the objective transitions in the current coverage set. During test generation, each time a state node with the maximal function value is singled out from the explored nodes as the new root node for continuing to traverse the state space in the TEA tree. As a result, the feasible test generation problem is reformulated into a multi-objective optimal search problem based on a TEA expanding tree.

To evaluate the effectiveness and efficiency of the HTEA, we conduct a detailed case study on four popular protocol EFSM models: INRES[18], Network Monitor [19], SCP [8] and Class II transport protocol [29]. Experimental results indicate that the HTEA significantly outperforms the traditional TEA method based on breath-first search in terms of search state space. The statistical data impressively show our method's potential in avoiding the state explosion problem during test generation.

The remainder of this paper is organized as follows. Section 2 briefly discusses related work. In Section 3, some preliminaries are provided, including an overview of TEA-based methods for test generation. The proposed approach to generating feasible test sequence is presented in Section 4. Section 5 provides experimental results. In Section 6, we draw concluding remarks.

2. Related work

In terms of EFSM-based test generation, many approaches were proposed in the literatures [2,4,5,32,34]. Hierons et al.[16] introduced a two-phase method to overcome the infeasible path problem for a class of EFSM models, on whose structure some restrictions were placed. The method is required to transform the EFSM into a new model called EEFSM without containing any infeasible path. Based on the expanded EFSM, feasible test sequences may then be generated. Uyar and Duale [13,30] attempted to enable the automatic test sequences generation for EFSMs where all operation and guards are linear. Algorithms were represented to detect and eliminate all conflicts among transitions from EFSM model. Therefore, all paths in the resulting EFSM are feasible and FSM-based methods can be directly employed to derive the feasible test sequences. However, the severe assumptions imposed on the considered EFSM limits its general application.

Similarly, some researchers also apply FSM-based testing approaches for testing from EFSM by expanding an EFSM to a FSM [11,23,27]. Unfortunately, model transformation often comes with the state explosion. The main reason is that the size of resultant state space is exponential increasing with the number of variables in EFSM model. Furthermore, such transformations may produce semantic losses, which will complicate test sequences generation [15,21].

Chanson et al. [9] tackled the test sequence executability problem by using constraint satisfaction problem method. When testing from EFSMs containing the influencing self-loops, transition self-loop analysis technique can be utilized to make the generated path feasible. Koh et al. [22] and Bourhfir et al. [6] also produced executable test sequences based on control flow and data flow criterion for an EFSM-specified protocol using the similar technique. Zhang et al. [37] investigated a test generation method and developed a toolkit using the combination of symbolic execution and constraint solving. It was used to deal with the infeasible path problem for testing an SDL specification [31]. But this class of methods is not always applicable for general EFSM models.

Recent work has studied the application of the search-based algorithms for automated test generation from EFSMs. In [12], Derderian et al. described an approach for feasible test generation using a Genetic algorithm (GA). A fitness function is defined to estimate the feasibility of a given path using its number and type of predicate conditions. But the method did not apply the data dependence between transitions to aid the evaluation. Kalaji et al. [12] improved the definition of the fitness function based on the dataflow dependence analysis and proposed a GA-based method for the generation of test sequences. The feasibility metric of a path was further improved to produce test generation from EFSMs with the counter problem in [20]. But GA-based approaches need two steps to found the test sequences. In addition, it does not guarantee the existence of feasible inputs to trigger the generated candidate test sequences. Based on control dependence analysis [3] and Pareto optimization technique [36], Yano et al. [35] presented a multi-objective evolutionary method for executable test generation. Similarly, Yang et al. [33] also used Multi-objective Pareto optimization to deal with the infeasible paths problem. However, these methods have inherent difficulty for test generation of EFSM-specified protocols. The choice of optimal initial population satisfying a specific control and data flow coverage criterion is still a challenging problem.

Transition Executability Analysis (TEA) [18] is a one-stage technique, which can ensure the executability of all derived test sequences from EFSM-based systems. Although the state explosion problem is partially alleviated by overlapping and state limitation strategy [19], relatively few heuristic methods are attempted.

3. Preliminaries

Protocol conformance testing, as a black-box testing, is categorized as: active testing and passive testing. The former relies on exercising preferred test sequences to stimulate the IUT and observe its output behavior. The latter only monitors the input and output of the system without any interaction. This work focuses on executable test sequences generation for EFSM-specified protocols in active testing.

3.1. EFSM model and related definitions

A deterministic EFSM model M can be formally described as a six-tuple $M =< S, s_0, V, I, O, T >$. S is a finite set of states and s_0 is the initial state, $s_0 \in S$. V is a variable set, where $V = V_p \cup V_c$, V_p and V_c represent the set of parameter variables and context variables, respectively. I denotes a set of input events, for $i \in I, i =?ip.in(v_1, v_2, \ldots, v_n)$, $v_i \in V_i$, $n \geq 0$. $?ip.in(v_1, \ldots, v_n)$ shows the input event in from the interaction point ip with a list of input parameters $v_1, \ldots, v_n$. $delay(m)$, as the special input event, denotes a timer, where m is the time. O is a set of output events, for $o \in O$, $o =!ip.out(v_1, v_2, \ldots, v_n)$. $!ip.out(v_1, v_2, \ldots, v_n)$is the output event out from ip with a list of output parameters $v_1, \ldots, v_n$. T denotes a finite set of transitions. For $t_i \in T$, $t_i =< s_i, i, p, a, o, s_j >$ is a transition where s_i and s_j represent t_i's head state and tail state, respectively; t_i is called the incoming transition of s_j and the outgoing transition of s_i; p is a predicate denoting a guard condition for firing t_i; $i \in I$ and $o \in O$ are an input and the associated output; $a \in A$ represents an action operated on variables.

The dynamic behavior of an EFSM can be represented as follows: (1) Receive an input event at state s_i; (2) Evaluate the predicate of each outgoing transition of s_i according to the current values of input parameter variables and context variables; (3) Execute one outgoing transition t_i when its p is satisfied, then the state transfer into s_j. More formally, the state transformation can be described as a state transition function f_T: $S \times V \times I \rightarrow S \times O$.

Generally, an EFSM model is graphically described as a labeled directed graph, where the node denotes state in the form of a circle and a directed edge between a pair of nodes represents a transition. Concretely, Fig. 1 shows an EFSM model for a modified initialor-responder protocol [18], which is taken as an example to make a discussion in this paper. In Fig. 1, the EFSM contains 18 transitions labeled from T1 to T18, where $S = (Disconnected, Waiting, Connected, Sending, Block)$ and $V = (counter, number, NUM, olddata, SDU, input, block, optional)$. For convenience, we label these states from S_1 to S_5.

Some definitions and terms used throughout the paper are given as following:

Definition 1 (State Configuration)**.** A state configuration SC_i of EFSM M is a two-tuple $SC_i =< s_i, \vec{v}_i >$, where $s_i \in S$, $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ is a n-dimensional value vector for variables $v_1, v_2, \ldots, v_n \in V$.

The state configuration is the combination of the current state and variable values. It can uniquely represent the current situation of an EFSM model. For instance, $SC_1 =< S_1, (0, 0, 0, 0, 0, false, false, false) >$ describes the initial configure of M in Fig. 1. At this moment, when receiving an input ?U.CONreq, M will be taken from $SC_1 =< S_1, (0, 0, 0, 0, 0, 0, false, false, false) >$ to $SC_2 =< S_2, (0, 0, 0, 0, 0, 0, false, false, false) >$ by the transition T2. Here, SC_1 and SC_2 are the head and tail state configuration of T2, respectively.

Definition 2 (Adjacent Transition Pair)**.** Let $< t_i, t_j >$ be an adjacent transition pair (ATP), if and only if $s^t(t_i) = s^h(t_j)$, where $s^t(t_i)$ and $s^h(t_j)$ refer to t_i's tail state and t_j's head state, respectively. Denote the set of all adjacent transition pairs as $ATPS$.

Definition 3 (Feasible transition path)**.** Let a transition path $\omega_i = t_1, t_2, \ldots, t_i, \ldots, t_n$ be a feasible transition path (FTP), if and only if $\forall < t_i, t_{i+1} > \in ATP, \bigwedge_1^n t_i.p = true, 1 \leq i \leq n-1$ where $t_i.p$ is the predicate of transition t_i and $n \geq 1$ is the length of the transition path ω_i.

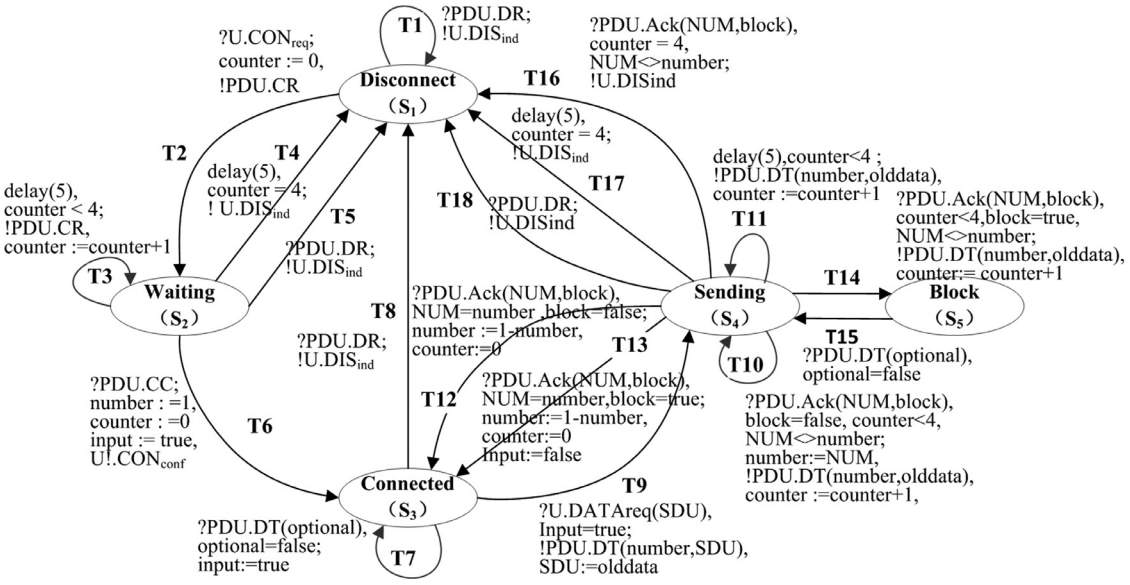

Fig. 1. A modified initialor-responder protocol.

3.2. Protocol conformance testing sequence

Obviously, it is infeasible to conduct an exhaustive test for a system with many states and transitions [24]. Thus, based on control flow (such as all-transition [21], UIOE [18] and trans-CIUS-set [29] etc,.) and data flow (i.e., All-uses [16] and DPO-path [19]), many protocol conformance testing coverage criteria were proposed to validate an implementation. These test criteria intended to find a balance between error detection ability and testing effort. When testing from an EFSM-specified system, test sequences are required to represent all preset test criterion in the form of a series of ordered transitions. Essentially, the main goal of test sequence generation is to search these paths in a generally huge state space.

3.3. Transition executability analysis (TEA)

A feasible test sequence consists of a series of adjacent transition pairs with the potential ability of consecutive execution. In other words, there always exists some test initial data to trigger these ordered transitions contained in the feasible test sequence. However, predicate conflicts between transitions complicate the test generation, when testing from EFSMs. As a solution to the infeasible problem, TEA technique is among most popular reachability analysis methods [18]. It depends on expanding a TEA tree to guarantee the feasibility of derived test sequences.

Definition 4 (TEA tree)**.** For an EFSM M, the TEA tree for generating feasible test sequences can be represented as a three-tuple $T^e =< ND, nd_0, TT >$, where ND is the finite set of nodes, $nd_0 \in ND$ is the root node and TT denotes the finite set of executable transitions. For $nd_i \in ND$, is a reachable state configuration of M. For $t_{ij} \in TT$, represents a transition leading from nd_i to nd_j, $t_{ij}.p = true$. Path(nd_i) represents a feasible path from the root node to nd_i. For $\omega_i = P\text{ath}(nd_i)$, $|\omega_i|$ denotes the length of the transition path ω_i.

In a TEA tree, a node can graphically described as a circle and an executable transition is described as a directed arc starting from its head state configuration to its tail state configuration. To help make the TEA tree definition concrete, a part of the expanding TEA tree for the EFSM in Fig. 1 is illustrated in Fig. 2.

Hence, a feasible test generation method using TEA technique can be formally described as following: (1) choose an initial state configuration as the root node; (2) expand a TEA tree from the root node and check each explored node to compute Path(nd_i); (3) determine if Path(nd_i) is a solution according to the test criterion. Consequently, TEA-based test sequence generation problem can be reformulated as a search problem. Clearly, for a given initial state configuration and EFSM M, the node space of the resultant TEA tree is definite, but it is a huge state searching space. This is the major cause for TEA-based methods easily suffering from the well-known state explosion problem.

3.4. Limitation of state pruning strategy

As mentioned above, the state explosion problem is the main potential barrier to apply TEA technique to derive feasible test sequences. Naturally, it seems intuitive that a state pruning strategy may be a solution for avoiding the state explosion.

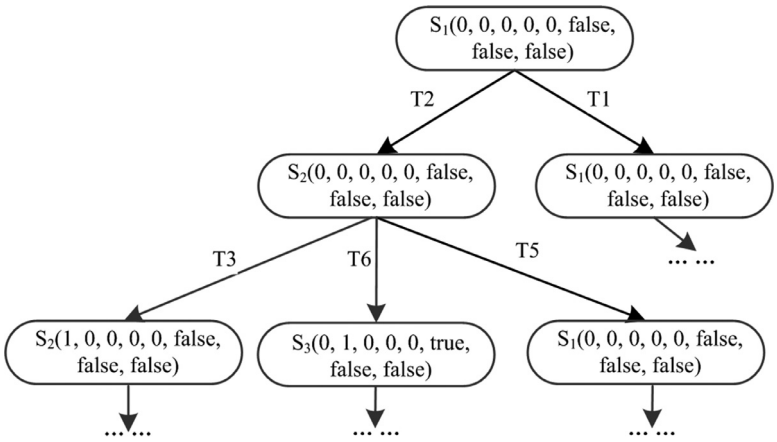

Fig. 2. An example of the TEA tree.

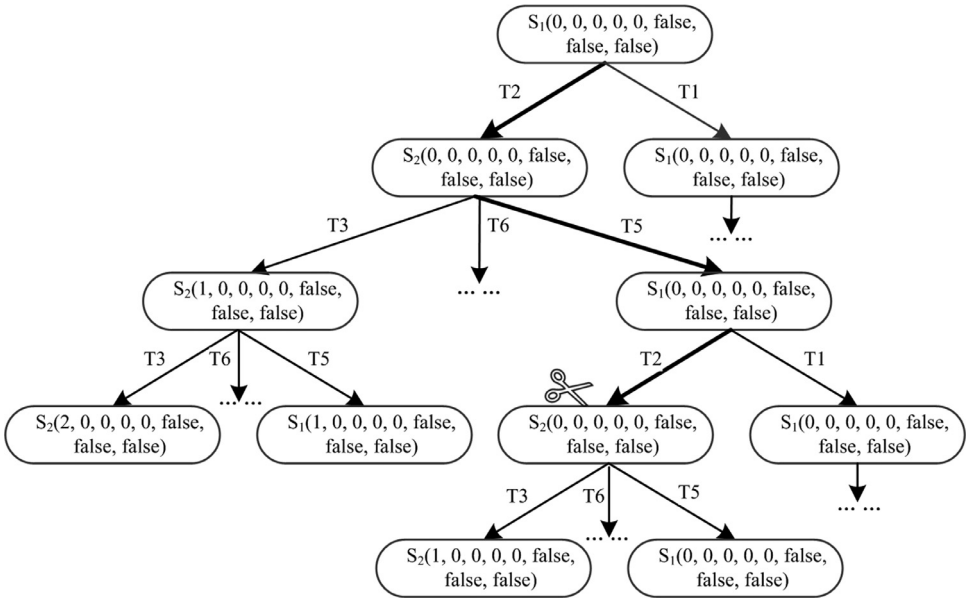

Fig. 3. Partial TEA expanding tree with a pruning strategy.

However, to best of our knowledge, there were comparatively little concerns placed on how to design the pruning strategy with respect to the specified test criterion. In addition, a pruning strategy has a threat to result in the failure of test generation.

In [19], a method is proposed to reduce the number of explored nodes during TEA expanding. It only checks the duplication of the combination of a node and its incoming transition in the current TEA tree, as each next node expanding. The duplicated node will be discarded. More concretely, take the M (in Fig. 1) as an example to illustrate the pruning process. Assume that the goal test coverage transition set is {T2, T3, T4, T5} and root node is $S_1(0, 0, 0, 0, 0, false, false, false)$. For the limited space, Fig. 3 shows the partial expanding TEA tree, in which a node only describes the state name and the values of involved variables. To derive a feasible test sequence, the TEA tree is explored level by level in the Bread-first way. From the Fig. 1, we can see that if a test sequence intended to cover both T4 and T5, it must traversed T2 at least two times. However, the pruning operation will be trigged as the transition T2 is exercised at the second time according to the proposed pruning method. As a result, any potential solution starting from the transition sequence of (T2, T5, T2) is discarded. Eventually, none result path satisfying the test coverage can be generated.

This example indicates that some redundant nodes play an important role during the test generation using the TEA tree. Thus, we proposed a new pruning method to adapt for this case in Section 4.3. Although the modified pruning strategy can limit the number of expanded nodes, the remaining search space is still huge. So a heuristic approach is urged to be developed for the promotion of the utilization of TEA technique in the testing practice.

4. Proposed test generation method

To avoid the state explosion problem, this paper proposed a heuristic transition executability analysis (HTEA) method that is tailored to efficient generation of feasible test sequences. The key idea in our proposed approach is to explore as few as possible nodes for producing a test sequence through heuristic guidance. This is achieved by influencing the order in which nodes are traversed for searching test generation during expanding the TEA tree.

The exploration is dynamically directed by the node metric that estimates the feasibility of a node to be explored next. To this end, an objective function is introduced to compute this metric using the static predication and dynamic feedback during test generation. The static predication depends on a Guidance Matrix (GM) that is constructed to capture the notion of the feasibility estimation of each transition pair in the EFSM. The dynamic part refers to the testing context of the current considered node during TEA tree expanding, such as the number of covered transitions and the searching level. The static and dynamic information contribute to guide test sequence search for a promising direction. Before giving detailed description of the new method, some concepts and definitions should be presented first of all.

4.1. Related definitions and guidance matrix

4.1.1. Feasibility relation classification

A feasible test sequence consists of a series of adjacent transition pairs. In other words, the successive executability of each ATP contained in a FTP contributes to its feasibility. Thus, it is necessary to classify the ATP to facilitate the successive feasibility estimation.

The feasibility relation of an $ATP < t_i, t_j >$ can be classified into three categories:

(1) Conflict relation R^c
There exists a variable v contained both in t_i and the predicate of t_j, such that the value of v will make the condition t_j violation, after traversal of t_i. $< t_i, t_j >$ belongs to conflict relation. Formally, $R(t_i, t_j)$ denotes a feasibility relation between t_i and t_j. let $R(t_i, t_j) \in R^c$, iff $\exists v \in VAR(t_i) \cap VAR(P(t_j))$ and $\text{Exec}(t_i) \rightarrow P(t_j) = false$, where $VAR(t_i)$ and $VAR(P(t_j))$ represents the set of variables contained in t_i and $P(t_j)$ is the predicate condition of t_j, respectively; $\text{Exec}(t_i)$ represents the execution of t_i and $P(t_j)$ is the predicate part of transition t_j. For example, $<$ T2, T4 $> \in R^c$, since the variable *counter* is assigned 0 in T2 and the condition expression of T4 is satisfied when $counter = 4$.

(2) Satisfaction relation R^s
Satisfaction relation includes two types: condition satisfaction and non-condition. For the former, let $R(t_i, t_j) \in R^s$, iff $\exists v \in VAR(t_i) \cap VAR(P(t_j))$ and $Exec(t_i) \rightarrow P(t_j) = \text{true}$. For an instance, $<$ T2, T3 $>$ is a condition satisfaction, which $Exec(\text{T2}) \rightarrow P(\text{T3}) = \text{true}$. For the latter, if t_j has no condition statement, the execution of t_i has no impact on the feasibility of t_j, i.e., t_i and t_j can always be successively traversed, such as $<$ T1, T2 $>$.

Proposition 1. *For an EFSM M and its two adjacent transition pairs* $< t_i, t_j >$ *and* $< t_j, t_k >$, *if* $R(t_i, t_j) \in R^s \wedge R(t_j, t_k) \in R^s$, *then* $R(t_i, t_k) \in R^s$.

Proof. Given an adjacent transition pair $< t_i, t_j >$. Since $R(t_i, t_j) \in R^s$, there exits at least a feasible path t_i, t_j leading from $s^t(t_i)$ to $s^h(t_j)$, according to the definition of satisfaction relation. Similarly, for $< t_j, t_k > \in ATP$, such that $R(t_j, t_k) \in R^s$, there is also a feasible path t_j, t_k. Consequently, M must has a feasible path t_i, t_j, t_k that is the concatenation of t_i, t_j and t_j, t_k, i,e, $R(t_i, t_k) \in R^s$. □

(3) Undecidable relation R^u
Let $R(t_i, t_j)$ be an undecidable relation. It means that their feasibility of successive execution depends on the specified values of variables in EFSM at that moment. That is, the feasibility relation can't be determined in advance through static analysis. In terms of conditional dependence, an undecidable relation can be classified as: conditional independence R^{ci} and conditional correlation R^{cc}. Let $R(t_i, t_j) \in R^{ci}$, when $DUse(t_i) \cap \text{PUse}(t_j) = \emptyset$ and $\text{PUse}(t_j) \neq \emptyset$, where $DUse(t_i)$ and $\text{PUse}(t_i)$ denote the set of variables used in definitions and predicates, respectively. Let $R(t_i, t_j) \in R^{cc}$, iff $DUse(t_i) \cap \text{PUse}(t_j) \neq \emptyset$. R^{cc} is also called definition-predicate (DP) dependence. As an example, $R(\text{T3, T4}) \in R^{cc}$.

4.1.2. Feasibility estimation of DP dependence

DP dependence indicates the potential constraint relationship regarding variables contained in the adjacent transitions. As discussed above, when $R(t_i, t_j) \in R^{cc}$, it is impossible to definitely judge whether t_i and t_j can be successively executed or not. Hence, an approximate predication of the successive execution feasibility of two adjacent transitions may be a solution to this problem. In this paper, we attempt to give a relative order of each adjacent transition pair in terms of their consecutive execution feasibility. To achieve this, the DP relations included in *ATPS* of the considered EFSM need be ranked. Consequently, these related orders are represented as values in (0, 1).

To give the rank, DP dependence in each two adjacent transitions is required to be classified according to the specific combination of definition and predicate. In the following classification, for an $ATP < t_i, t_j >$, we focus on the combination of t_i's definition operations and t_j's predicates. A transitions condition part may have combined guards linked by *AND* and *OR*. Before description of the solution to deal with combination predicate, we first consider the atomic guard.

An atomic guard G can be represented as $G = E_L \Theta E_R$, where E_L and E_R denotes left expression and right expression, respectively; $\Theta \in \{>, <, \geq, \leq, =, \neq\}$ is a logic operator symbol. For the convenience of DP classification, a guard G is converted to the normalization form $G' = v \Theta EP_g$, where $G \equiv G'$, v is a variable appearing both in t_i's definition and t_j's predicate in an adjacent transition pair. EP_g is the remaining logical expression. Clearly, a transition with a guard containing the operator $=$ is more difficult to be fired, compared to other operators. It seems that the operator $\neq$ is most likely to be satisfied. The operator $<$ and $>$ can be considered to be of the same feasibility. Operators of $\leq$ and $\geq$ also have identical probabilities.

The definition use of a variable in a transition is normally separated into two types: I-Use and A-Use. I-Use refers to the definition of parameter variables in the input part of a transition, so a tester has an opportunity to choose value of parameter variables to satisfy its associated predicate. A-Use is a set of context variables included in the action part of transitions. Therefore, here, we concentrate on variables in the A-Use. The form of definition of a variable in a transition is like $v := EP_a$, where v represents a defined variable and EP_a is an assignment expression. It can be reformulated as $v = EP_L \Delta EP_R$, where EP_L and EP_R represent the left expression and right expression, respectively, Δ denotes a calculation operator. When EP_a contains more than one calculation operators, the leftmost one is chosen as the considered Δ. The right remainder of EP_a becomes the EP_R. In a protocol EFSM under testing, calculation operators in the assignment statement of a transition commonly include addition, subtraction and no operator (nop). More formally, three types of calculation operators comprise the set $AO = \{+, -, nop\}$. In this paper, we classify the definition of a variable in terms of the combination of the type of variables contained in EP_a and Δ. As a result, an assignment to variable v can be classified as one of the following types:

(1) AC_{pa}: the value of v depends on parameter variables and Δ is $+$; $Var(EP_a)$ contains at least one parameter variable, $Var(EP_a)$ denotes the set of variables contained in EP_a.
(2) AC_{ps}: the value of v depends on parameter variables and Δ is $-$; $Var(EP_a)$ contains at least one parameter variable.
(3) AC_{pn}: the value of v depends on parameter variables and Δ is nop; $Var(EP_a)$ contains at least one parameter variable.
(4) AC_{va}: the value of v depends on the context variable and Δ is +; $Var(EP_a)$ includes only context variables.
(5) AC_{vs}: the value of v depends on the context variable and Δ is -; $Var(EP_a)$ includes only context variables.
(6) AC_{vn}: the value of v depends on the context variable and Δ is nop; $Var(EP_a)$ includes only context variables.
(7) AC_c: the value of v depends on the constant, i,e., only constant appears in $Var(EP_a)$.

As a result of above analysis, we can see that there are three factors need to be considered when ranking a DP dependence: the variable type included in EP_a, assignment operator and guard operator. On the basis of the classification of the guard and assignment, each DP-related transition pair can be given a ordered weight in (0,1) to show how easy they are sucessively executed. A matrix of DP relation (MDP) that describes the rank of different combination is defined as the following:

$$M^{DP} = \begin{array}{c} \\ AC_{pa} \\ AC_{ps} \\ AC_{pn} \\ AC_{va} \\ AC_{vs} \\ AC_{vn} \\ AC_c \end{array} \begin{array}{c} \begin{array}{cccc} = & < \text{or} \leq & > \text{or} \geq & \neq \end{array} \\ \begin{bmatrix} 23 & 24 & 26 & 27 \\ 23 & 26 & 24 & 27 \\ 24 & 25 & 25 & 27 \\ 18 & 19 & 21 & 22 \\ 18 & 21 & 19 & 22 \\ 18 & 20 & 20 & 22 \\ 8 & 13 & 13 & 16 \end{bmatrix} \end{array} / 28 = \begin{array}{c} \\ AC_{pa} \\ AC_{ps} \\ AC_{pn} \\ AC_{va} \\ AC_{vs} \\ AC_{vn} \\ AC_c \end{array} \begin{array}{c} \begin{array}{cccc} = & < \text{or} \leq & > \text{or} \geq & \neq \end{array} \\ \begin{bmatrix} 0.82 & 0.86 & 0.93 & 0.96 \\ 0.82 & 0.93 & 0.86 & 0.96 \\ 0.86 & 0.89 & 0.89 & 0.96 \\ 0.64 & 0.68 & 0.75 & 0.79 \\ 0.64 & 0.75 & 0.68 & 0.79 \\ 0.64 & 0.71 & 0.71 & 0.79 \\ 0.29 & 0.46 & 0.46 & 0.57 \end{bmatrix} \end{array}. \quad (1)$$

In M^{DP}, all three factors are taken into consideration to quantify the feasibility estimation of a DP transition pair. The row represents a variety of assignment types and the column represents the list of guard operator types. A cell represents the metric that indicates the feasibility rank of a DP relation. The given metric values are chosen based on our previous preliminary experiments and the experimental results illustrate the effectiveness of these values during test generation. Since these metric values are merely used to rank various DP dependences according to their estimated feasibility, other advisable values can also be applied.

A variable can be defined in the action part of a transition more than one times. In such case, the last assignment statement regarding the variable is considered during the rank of the DP relation. Furthermore, multiple different variables definition may occur in t_i, which leads to the existence of multiple DP dependences in t_j's atomic guard. When it happens, the minimum metric among DP relations is used. Eventually, in terms of combined guard, the approach for its evaluation is introduced in Section 4.3.

4.1.3. Transition feasibility estimation

The predicate condition in a transition t_i commonly consists of some condition expressions including variables and operators. For the evaluation of t_i's feasibility, it is necessary to classify the predicates according to the associated variables and operations. First of all, we consider the atomic guard.

A guard that contains variables in V_p is easier triggered than one with variables in V_c or constants. The main cause is that a tester can easily choose value of parameter variables to satisfy a guard condition. Therefore, if the condition in a transition can be determined through parameter variables assignment, we take it as a feasible transition such as T15 in

Fig. 1, during test sequence generation. But context variables in a guard are unable to be directly assigned to excise the transition. Formally, in terms of types of variables, a transition t_i's guard can be categorized as following:

(1) Constant predicate $G^{\emptyset}$: for $g \in G^{\emptyset}, \mathrm{VAR}(g) = \emptyset$, where $\mathrm{VAR}(g)$ is a set of variables contained in the predicate g. For example,T1 and T8 in Fig. 1 have constant predicates.
(2) Parameter predicate G^{p}: for $g \in G^{p}$, $\mathrm{VAR}(g) \cap V_p \neq \emptyset \wedge \mathrm{VAR}(g) \cap V_c = \emptyset$. Concretely, T7 has parameter predicates.
(3) Context predicate G^{c}: for $g \in G^{c}$, $\mathrm{VAR}(g) \cap V_p = \emptyset \wedge \mathrm{VAR}(g) \cap V_c \neq \emptyset$, where it only contains context variables without parameter variables, such as predicates in T3 and T4.
(4) Parameter and context predicate G^{pc}: for $g \in G^{pc}$, $\mathrm{VAR}(g) \cap V_p \neq \emptyset \wedge \mathrm{VAR}(g) \cap V_c \neq \emptyset$, where context variables and parameter variables are both included. A predicate in T10 belongs to G^{pc}.

Based on the above discussion, a 4×4 matrix M^T is introduced to predict the feasibility of the transition, that is called Transition Feasibility Estimation Matrix (TFEM) and given as follows:

$$M^T = \begin{array}{c} \\ \neq \\ \leq \text{ or } \geq \\ < \text{ or } > \\ = \end{array} \begin{array}{c} \begin{array}{cccc} G^{\emptyset} & G^{p} & G^{c} & G^{pc} \end{array} \\ \begin{bmatrix} 1 & 15/16 & 4/16 & 11/16 \\ 1 & 14/16 & 3/16 & 10/16 \\ 1 & 13/16 & 2/16 & 9/16 \\ 1 & 12/16 & 1/16 & 8/16 \end{bmatrix} \end{array} = \begin{array}{c} \\ \neq \\ \leq \text{ or } \geq \\ < \text{ or } > \\ = \end{array} \begin{array}{c} \begin{array}{cccc} G^{\emptyset} & G^{p} & G^{c} & G^{pc} \end{array} \\ \begin{bmatrix} 1 & 0.94 & 0.25 & 0.69 \\ 1 & 0.86 & 0.19 & 0.63 \\ 1 & 0.81 & 0.13 & 0.56 \\ 1 & 0.75 & 0.06 & 0.5 \end{bmatrix} \end{array} \tag{2}$$

In M^T, the row denotes a variety of operators and the column is a list of predicate types. A cell indicates the feasibility rank for a relevant combination of variables and operators in a predicate, which is mapped to a value range from 0 to 1. Bigger is the metric, more probably to be fired is the transition with the resultant predicate. The value of element is not an explicit probability, but an ordered estimation for the feasibility of a transition, with respect to an amalgam of operators and variables in its predicate. Generally, a transition's guard may contain composition conditions linked by **AND** and **OR**. For the former, its estimation value is the multiply of atomic predicates. However, for the latter, we singled out the maximum of two atomic predicates to represent its value.

4.1.4. Guidance matrix

In order to describe and quantize the dependence relation between each ATP in an EFSM, we define a matrix called Adjacent Transitions Feasibility Matrix (ATFM). Formally, the definition of ATFM is given as following.

Definition 5 (Adjacent Transitions Feasibility Matrix (ATFM))**.** Let M^{A} be a $N \times N$ ATFM for an EFSM M, where N represents the number of transitions in M. For $\forall < t_i, t_j > \in ATPS$,

$$M^{\mathrm{A}}_{ij} = \begin{cases} 0, & R(t_i, t_j) \in R^{\mathrm{c}}. \\ 1, & R(t_i, t_j) \in R^{\mathrm{s}}. \\ p_{ij}, & R(t_i, t_j) \in R^{\mathrm{u}} \wedge p_{ij} \in [0, 1]. \end{cases} \tag{3}$$

The value of M^{A}_{ij} describes the estimation of the successive execution probability of an ATP $< t_i, t_j >$. That is, M^{A}_{ij} denotes the feasibility that t_j can be successively triggered after the execution of transition t_i. Clearly, if a dependence relation can be explicitly determined using static analysis, the value 0 and 1 is evaluated to represent the conflict relation R^{c} and satisfaction relation R^{s}, respectively. While it belongs to an undecidable relation R^{u}, a metric p_{ij} is used to represent an estimation of its possibility for successive execution. In this case, p_{ij} is a value ranging from 0 to 1, which is evaluated as following:

$$p_{ij} = \begin{cases} \mathrm{P}(t_j), & R(t_i, t_j) \in R^{ci}. \\ (dp_{ij} + \mathrm{P}(t_j))/2, & R(t_i, t_j) \in R^{cc}. \end{cases} \tag{4}$$

$\mathrm{P}(t_j)$ represents the estimated feasibility of transition t_j, according to the TFEM of considered EFSM. For $R(t_i, t_j) \in R^{ci}$, the value of p_{ij} depends on $\mathrm{P}(t_j)$, since t_i and t_j are independent transitions. Although $\mathrm{P}(t_j)$ can predict the feasibility of a transition t_i, it is not a real probability. Hence, classical conditional probability method is unfit to be applied to compute the p_{ij}, when t_i and t_j have an undecidable relation. Here, for $R(t_i, t_j) \in R^{cc}$, the assignment of p_{ij} considers both the feasibility of t_j and the data dependence between t_i and t_j. Accordingly, $(dp_{ij} + \mathrm{P}(t_j))/2$ is used to quantize the value of p_{ij}, where dp_{ij} is an element of M^{DP} denoting the consecutive execution possibility for t_i and t_j.

According to the definition of ATFM, the greater value of M^{A}_{ij} illustrates the more feasibility for the corresponding ATP $< t_i, t_j >$. For the convenience of the feasibility evaluation for any transition pair in M, M^{A}_{ij} is converted into a new matrix $M^{\bar{\mathrm{A}}}$, which is defined as $M^{\bar{\mathrm{A}}}_{ij} = 1 - M^{\mathrm{A}}_{ij}$. Based on $M^{\bar{\mathrm{A}}}$, the minimum distance between each transition pair in $M^{\bar{\mathrm{A}}}$ can be computed using Floyd's algorithm. Hence, the minimum-weighed adjacency matrix of $M^{\bar{\mathrm{A}}}$ is constructed to heuristically direct the TEA expanding, which is named guidance matrix (GM) defined as following:

Algorithm 1 Guidance matrix construction

Input: EFSM M, M^{T} and M^{DP}
Output: a guidance matrix M^{G}
1: initialize the transition set T;
2: **for** $i \leftarrow 0$ to $|T|$ **do**
3: **for** $j \leftarrow 0$ to $|T|$ **do**
4: **if** $< t_i, t_j > \notin ATP \vee R(t_i, t_j) \in R^{\mathrm{c}}$ **then**
5: $M^{\mathrm{A}}_{ij} \leftarrow 0$;
6: **else**
7: **switch** $R(t_i, t_j)$
8: **case** R^{s}**:**
9: $M^{\mathrm{A}}_{ij} \leftarrow 1$;
10: **case** R^{ci}**:**
11: select a metric $\mathrm{P}(t_j) \in M^{\mathrm{T}}$ according to TFEM definition, $M^{\mathrm{A}}_{ij} \leftarrow \mathrm{P}(t_j)$;
12: **case** R^{cc}**:**
13: select $dp_{ij} \in M^{\mathrm{DP}}$ and $\mathrm{P}(t_j) \in M^{\mathrm{T}}$ according to TFEM and MDP definition, respectively , $M^{\mathrm{A}}_{ij} \leftarrow (dp_{ij} + \mathrm{P}(t_j))/2$;
14: **end switch**
15: **end if**
16: $M^{\bar{\mathrm{A}}}_{ij} \leftarrow 1 - M^{\mathrm{A}}_{ij}$;
17: **end for**
18: **end for**
19: **call** the Floyd-Warshall algorithm to compute the minimum weight matrix $\mathrm{Cmin}(M^{\bar{\mathrm{A}}})$ of $M^{\bar{\mathrm{A}}}$ and corresponding path length matrix L^m;
20: **for each** $M^{\bar{\mathrm{A}}}_{ij} \in M^{\bar{\mathrm{A}}}$ **do**
21: $M^{G}_{ij} = (1 - \mathrm{Cmin}(M^{\bar{\mathrm{A}}}_{ij})/L^m_{ij})/L^m_{ij}$;
22: **end for**
23: **return** M^{G};

Definition 6 (Guidance Matrix (GM))**.** Let M^G be a $N \times N$ guidance matrix for an EFSM M, where N represents the number of transitions in M. For any transition pair $< t_i, t_j >$, $M^G_{ij} = (1 - \mathrm{Cmin}(M^{\bar{\mathrm{A}}}_{ij})/L^m_{ij})/L^m_{ij}$, where $\mathrm{Cmin}(M^{\bar{\mathrm{A}}}_{ij})$ represents the minimum weight and L^m_{ij} is the length of the corresponding shortest path.

Clearly, when M^G_{ij}= 1, it means that there definitely exists a feasible path between t_i and t_j, according to Proposition 1. Otherwise, the value of each cell in M^G indicates the possibility of the existence of a feasible path leading from t_i to t_j. Consequently, if a transition pair $< t_i, t_j >$ has a more higher metric in M^G, it has more opportunity to yield a feasible path linking t_i and t_j. Based on the M^G, a row guidance vector $(1 \times n)$ for the transition t_i can be defined as $\vec{R_i} = [M^G_{i1}, M^G_{i2}, \ldots, M^G_{in}]$. A $n \times 1$column vector $\vec{C} = [x_1, x_2, \ldots, x_i, \ldots, x_n]^{\mathrm{T}}$ represents a coverage transition set in terms of the current testing context, where the ith element x_i is assigned 1, if t_i is a objective transition required to be traversed according to the test goal, otherwise 0. As a result, $\vec{R_i} \cdot \vec{C}$ represents the accumulated feasibility estimation from t_i to each uncovered transition in the coverage transition set.

As described above, the guidance matrix construction can be presented as Algorithm 1. It first ranks each adjacent transition pair in M and produces the resulting matrix $M^{\bar{\mathrm{A}}}$ (from line 2 to line 18). Then, the minimum weight matrix $\mathrm{Cmin}(M^{\bar{\mathrm{A}}})$ and its corresponding path length matrix L^m are calculated using the Floyd–Warshall algorithm. The last process applies the definition of GM to generate the resulting M^G. Therefore, the guidance matrix construction algorithm has complexity $O(|T|^3)$, where $|T|$ is the total number of transitions in M.

4.2. Objective function

As mentioned above, search-based test sequence generation can be formally described as a multi-objective search problem. The main objectives for generating test sequences in EFSM-specified protocols are feasibility, test coverage and test length. Hence, it is necessary to define an objective function to capture the notation of the three goals to probe the optimal solution. In this paper, an objective function is constructed using the guidance matrix representation of EFSM under testing.

It holds guidance factors to direct the TEA tree expanding in taking a decision when choosing the next explored node. The proposed objective function attempts to make a trade-off among the above mentioned aims.

In our proposed method, the objective function is applied to evaluate each node in expanding TEA tree. At each step, the node with maximum function value is taken as the root of sub-tree to be explored. Concretely, during expanding an TEA tree, an explored node may yield multiple executable outgoing transitions. For each transition t_i, we utilize the objective function to compute weights of its tail nodes, representing its fitness as the next traversed node according to our formulated optimization problem. The objective function f_{obj} is formally given as:

$$f_{\text{obj}}(nd_i, t_i, Cov) = \begin{cases} \frac{\vec{R_i}\cdot\vec{C_i}}{n} \times \left(\frac{m+1}{L\times n}\right)^2, & n > 0 \wedge m \geq 0 \\ 1 \quad , & n = 0 \end{cases} \tag{5}$$

nd_i refers to the current node in the considered TEA tree and t_i is nd_i's executable incoming transition. Suppose TC is a set of transitions to be covered according to a specific test criterion. Cov represents the current coverage transition set for nd_i. $\vec{R_i}$ and $\vec{C_i}$ are the guidance vector and coverage transition set vector regarding t_i, respectively. n and m represent the number of uncovered and covered transitions contained in TC by $\text{Path}(nd_i)$, such that $m + n = |TC|$. The initial m is 0. L is the length of $\text{Path}(nd_i)$. Specially, if n equals 0, it indicates that $\text{Path}(nd_i)$ has covered all transitions in TC. That is, $\text{Path}(nd_i)$ is a solution. Thus, the weight of current node is assigned the maximum 1. For simplicity, if $n > 0$, f_{obj} can be transformed into the form $f_{\text{obj}} = F_{\text{avg}} \times ((m+1)/L \times n)^2$, where $F_{\text{avg}} = \vec{R_i}\cdot\vec{C_i}/n$ describes the average feasibility for all transition pairs linking from t_i to each uncovered transitions in Cov. Obviously, the value of F_{avg} is a real in the range [0, 1).

It can be seen that m increases and n decreases, when $t_i \in Cov$ is the first traversed transition. Consequently, f_{obj} is increased. For each node nd_i at the same level L, f_{obj} is to be increased or decreased, depending on the specific value of its F_{avg}. Assume that m and L increased at the same rate, f_{obj} is still increased due to a corresponding reduction in n. However, with the path length L increasing, if no new transition contained in Cov is traversed through the current $\text{Path}(nd_i)$, that is, m and n remain unchanged, f_{obj} is to be decreased exponentially. Thus, f_{obj} holds two guidance factors: encouraging the early generation of a feasible path satisfying specific testing coverage requirements and punishing the increment of the generated paths length.

4.3. Heuristic transition executability analysis (HTEA) method

As described above, in a TEA-based approach, a test sequence can be mapped to a path that covers TC, initialized from a root node through the TEA tree for EFSM under test. Traditional TEA methods intend to generate the shortest test sequence using the Breadth-First-Search (BFS) way. However, it may suffer from the state explosion problem due to the huge state searching space. To address this problem, the proposed approach, named heuristic transition executability analysis (HTEA), introduces the heuristic search to find a near-optimal solution. HTEA method is based on the guidance information from above defined objective function that contributes to tremendously reduce the search state space.

In essence, HTEA is a directed search algorithm motivated by the prevalent optimizing search strategy named Best-First (BF) [1], which each time chooses a node with maximal weight as the next node to be explored. The key idea used in HTEA algorithm is to expand a node that most likely contributes to a solution as early as possible. Since a nodes metric describes the estimation information, direction of the search can be quickly guided to an approximate optimal solution. In terms of an EFSM M, the proposed algorithm that takes an initial state configuration SC_0 and a test coverage set TC, returns a feasible test sequence FTS. The high level description of HTEA algorithm is given as Algorithm 2.

The algorithm can be explained as four steps. The first step is an initialization process to construct a guidance matrix M^G for the subject EFSM M, initialize the root node nd_0, and then put it to the node set ξ_c and ξ_b, respectively. In Step 2, a *while* loop is used to make a heuristic search for a feasible test sequence to cover all transition in TC. ξ_c stores all candidate nodes for next exploration, which have been produced by expanding TEA tree. Each while loop run, HTEA selects a node with a maximal metric from ξ_c as a prioritized target for further state traversal using the function **MaxWeightNode**(ξ_c)(Line 5). If two nodes have the identical weight, **MaxWeightNode**(ξ_c) will return the latest explored one.

Step 3 the executability analysis process is invoked (line 7). For each outgoing transition t_i of the current node nd_i, t_i's guard is checked to determine its feasibility, according to the related variables value in $nd_i.sc$. In terms of each executable transition t_i, it is necessary to judge if the path from the root node to t_i's tail node has covered all the transitions in TC. If so, this path is a FTS and the algorithm will be terminated. Otherwise, step 4 will generate t_i's tail node nd_j and update its associated fields. Especially, it is important to compute the weight of nd_j using the defined objective function $\mathbf{f}_{obj}$. In this way, the direction of TEA exploration can be most likely guided by the feedback information from the dynamic execution context. Then, it utilizes a new pruncing strategy (Line 18) to judge whether the new generated node nd_j can be discarded or not. Our pruncing strategy extends the method in [19] to enable successful test generation through checking covered transitions' count for the duplicate node. If nd_j satisfies the pruncing condition, it will be throw away and the TEA exploration is to be continued. Otherwise, the tail node nd_j will be stored into the node set ξ_c and ξ_b for the preparation of next exploration.

Algorithm 2 HTEA-based test generation

Input: EFSM M, SC_0 and TC
Output: a feasible test sequence FTS

generate guidance matrix M^G by using **Algorithm 1**;
set root node nd_0 : $nd_0.sc \leftarrow SC_0$, $nd_0.level \leftarrow 1$,$nd_0.path \leftarrow null$, $nd_0.weight \leftarrow 1$, $nd_0.\psi \leftarrow \emptyset$, $nd_0.t_{in} \leftarrow null$;
initialize node set: $\xi_c \leftarrow \{nd_0\}$, $\xi_b \leftarrow \{nd_0\}$;
while $\xi_c \neq \emptyset$ **do**
  $nd_i \leftarrow$ **MaxWeightNode**(ξ_c);
  $\xi_c \leftarrow \xi_c - \{nd_i\}$;
  for each executable outgoing transition t_i of $nd_i.sc$ **do**
    $\psi_{temp} \leftarrow nd_i.\psi \cup \{t_i\}$;
    if $t_i \in TC \wedge (TC - \psi_{temp}) = \emptyset$ **then**
      return $FTS \leftarrow nd_i.path \circ t_i$; /*find a FTS*/
    else
      $nd_j.t_{in} \leftarrow t_i$;
      $nd_j.sc \leftarrow$ **TailSC**$(nd_i.sc, t_i)$;
      $nd_j.level \leftarrow nd_i.level + 1$;
      $nd_j.path \leftarrow nd_i.path \circ t_i$;
      $nd_j.\psi \leftarrow \psi_{temp}$;
      $nd_j.weight := \boldsymbol{f}_{obj}(nd_j, t_i, TC - nd_j.\psi)$;
      if $\exists nd_k \in \xi_b \wedge (nd_k.sc = nd_j.sc \wedge nd_k.t_{in} = nd_j.t_{in}) \wedge (|nd_j.\psi| \leq |nd_k.\psi|)$ **then**
        continue; /*node nd_j is prunced, HTEA exploration will be continued */
      else
        $\xi_c \leftarrow \xi_c \cup \{nd_j\}$;
        $\xi_b \leftarrow \xi_b \cup \{nd_j\}$;
      end if
    end if
  end for
end while

5. Experiments

This section describes experiments for test sequence generation from four EFSM models based on the proposed approach. The purpose of experiments is to evaluate the effectiveness and performance of HTEA method regarding the reduction of searching state space, under different testing scales. To achieve this goal, we design two kinds of experiments: E1 and E2. For experiment E1, the test coverage set TC each time just involves one transition. Commonly, a relatively small number of state nodes are required to be explored in this case. In the E2, the number of transitions contained in TC is gradually increased, which leads to the increasing search state space.

Since TEA-based methods can ensure the executability of generated test sequences, test sequence length and the number of explored state nodes become two main factors to measure the effectiveness of the proposed method in our experiments. Thus, the classic TEA (CTEA) approach in BFS way is chosen as the benchmark for showing the difficulty of test generation from these EFSMs. For each TC, we apply the CTEA and HTEA method to yield test sequences under the same test context, respectively. Then, the above two performance indexes are compared and analysed in detail.

Implementation of TEA-based algorithms requires the EFSM model under testing to be expressed in an executable way. That is, it should support for model information extraction, i.e., variables and dependence relations, and systems behavior simulation. To achieve this, a free parsing engine for C#, named GOLD parser [10], is utilized as an interpreter to build the executable model. In the prepare phase for test generation, GOLD parser can be used to perform static semantic analysis of expressions in an EFSM model for constructing the guidance matrix. In the TEA tree expanding phase, given a state configuration and then its outgoing transition's guard expression is parsed and evaluated by GOLD parser to determine its executability. If a transition can be triggered, GOLD parser also helps simulate its dynamic behaviors such as variables assignment and action execution. The experiment prototype is developed in the environment as following: Windows 7 operating system, Microsoft Visual Studio 2010 for C# and GOLD parser builder 5.2.

5.1. Experiment configuration

The four subject EFSM-specified protocols in our experiments are popularly used in other literatures, which are INRES [18], Network Monitor[19], SCP [8], and Class II transport protocol [29]. TEA-based technique can be used to generate both the control-flow and data-flow test sequence. However, for the purpose of the concentration on evaluating the effectiveness

Table 1
Statistical results comparison between CTEA and HTEA in E1.

EFSMs	CTEA method			HTEA method			Ratio1(%)	Ratio2(%)
	# N1	#N2	#L	# N1	#N2	#L		
M1: INRES	181.94	53.44	4.44	39.72	10.02	4.44	78.17	0
M2: Network Monitor	41.06	8.12	7.29	22.35	1.71	7.29	45.56	0
M3: SCP	14.13	0.88	4.5	10.25	0.5	4.5	27.43	0
M4: Class II transport protocol	21.1	0.29	3.48	18.89	0	3.48	10.46	0
Total	64.55	15.68	4.93	22.8	3.06	4.93	40.41	0

of the proposed approach, we design the experiments based on the comparatively simple control flow criterion. In addition, state verification sequence [28,29] is also be omitted, which has no impact on our experimental objectives. Accordingly, a control-flow test sequence to cover a transition comprises of three parts: preamble, tested transition and postamble. A preamble refers to a path initialized from the initial state to the head state of a specified tested transition. A postamble is a path that returns to the initial state from the tail state of a tested transition. Generating postamble can facilitate taking an IUT to the initial state for the next test sequence execution.

In the first experiment E1, we derives a control-flow test case for each considered EFSM. The test case consists of a suite of test sequences, each of which enumerately covers a transition in the EFSM under test. Each test sequence generation thus needs call the generation algorithm twice. The first execution is to find a subsequence linking the preamble and the tested transition. The second run will search another subsequence, i.e., the postamble. The concatenation of two subsequences becomes the resulting test sequence. Since the initial value of variables associated with initial state may influence the experimental results, for each tested transition in each subject EFSM, we repeated test sequences generation 100 times with random initial variables value. The concrete experiment parameters are configured as: (1) INRES EFSM: the initial state is the *Disconnect* and the rang of initial value chosen for each integer variable is [0, 5]; (2) Network Monitor EFSM: the initial state is *Idle* and the rang of initial value chosen for each integer variable is [0, 10]; (3) SCP EFSM: the initial state is s_1 and the rang of initial value chosen for each integer variable is [0, 3]; (4) Class II transport protocol EFSM: the initial state is also s_1 and the rang of initial value chosen for each integer variable is [0, 1000].

The second type experiment E2 is designed to further check the ability of the proposed approach in alleviating state explosion, when *TC* contains multiple transitions. In this case, the postamble generation is also ignored for simplicity. The HTEA algorithm takes the initial state configuration and the specified *TC* as its input and yields a feasible test sequence to cover all transitions in *TC* at least once. For each considered EFSM *M*, we set a test objective set $TCS = \{TC_1, TC_2, TC_3, ..., TC_n\}$, where $TC_i = \{t_1, t_2, ..., t_i\} \in TCS$, $1 \leq i \leq n$ and n refers to the number of transitions in *M*. For example, there are 18 objective *TC*s in the INRES EFSM, such as $\{T1\}, \{T1, T2\}, ..., \{T1, T2, ..., T18\}$. For each *TC*, test generation algorithms are also repeated 100 times with random initialized variables, respectively. The related initial state configuration is similar to E1. In addition, for the limitation of experimental effort, we set 200,000 as the upper bound for the number of explored state nodes. When traversed nodes count achieves the maximum value, the algorithm terminates, which indicates the fact that test generation procedure suffers from state explosion problem.

5.2. Experimental results and analysis

Table 1 shows the statistical results for E1 concerning the four subject EFSM models. #N1 and #N2 represent the average number of explored nodes and pruned nodes for test generation over the specific EFSM, respectively. #L denotes the average length of the resulting test sequence. Ratio1 represents the reduction ratio on average corresponding to #N1, when the proposed method is compared with the classic TEA approach. Meanwhile, ratio2 refers to the average increment ratio of #L. All resulting data presented in Table 1 is the average by running 100 times of repetitions per experiment. Normally, the size of the resultant searching state space for test generation is exponential in the length of generated test sequence using CTEA method, in a specific EFSM under test. However, it can be noted that the average of #L is relatively small value 4.93 in the experiment E1. Thus, it is only required to explore 64.55 state nodes on average to generate a test sequence based on the benchmark method. That is, the procedure of test generation in E1 didn't undergo state explosion. In this case, compared to the classic TEA method, we can be seen that the new method still can compress #N1 to a ratio of 40.41%, without increasing the length of test sequences, as shown in Table 1.

From Table 1, we can see that Ratio1 varied widely between four models under testing. In terms of DP dependence, different features of considered models may be the main cause. Generating EFSM-specified protocol test sequences is generally complicated by the existence of transitions with difficult to satisfy guards. These guards may reference some variables defined or updated in other transitions, such as a counter variable [20]. For example, M1 contains the transition T4 with a guard counter=4 and the value of counter is initially 0. T4 cannot be triggered unless the transition T3 that has an action counter:=counter+1 has already executed four times. In such case, to generate a feasible test sequence covering T4, it is necessary that TEA-based methods expand at least four levels of the TEA tree starting from the tail node of T3. For the HTEA method, partial nodes are required to be heuristically explored. On the contrary, the CTEA method need to traverse

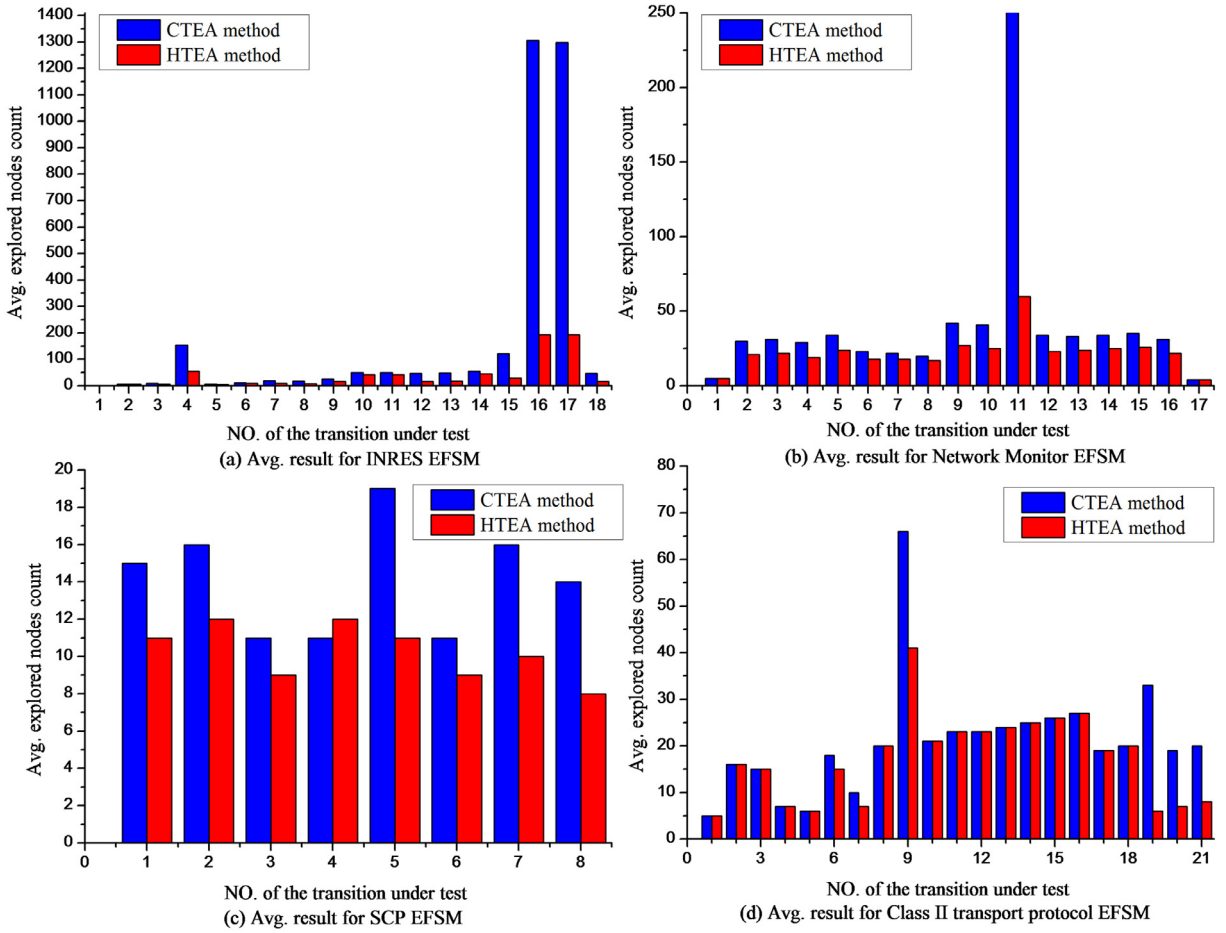

Fig. 4. Explored nodes count for each tested transition in four EFSMs.

nearly all nodes contained in the TEA tree in the breadth-first way, which provides an opportunity to significantly reduce the number of the explored nodes.

Detailed numbers of explored nodes (NEN) for producing test sequences in E1 are described in Fig. 4. The horizontal coordinate is the index of the transition under test in the considered EFSM. The vertical coordinate means the specific NEN. As can be seen in Fig. 4, the NEN varied slowly in the new method for four target EFSMs. However, the CTEA method has a dramatic increase with respect to NEN at some test contexts. Concretely, when testing the transitions T16 and T17 in $M1$ as depicted in Fig. 4(a), the corresponding NEN are 1305 and 1297, respectively, and their resulting test sequence lengths are both 8. But the average NEN and test sequence length of $M1$ are only 181.94 and 4.44 as shown in Table 1. Similarly, Fig. 4(b) shows that the NEN for testing transition T11 is 250 and the length of resultant test sequence is 11.

In the experiment E2, the benefit of the proposed method appears more obviously in terms of compressing the number of explored state nodes. Fig. 5 plots the experimental results of E2. From the Fig. 5, it can be seen that there is a significant trend regarding the reduced ratio of explored state nodes (RRESN) for HTEA method over CTEA approach. The red plot line describes the exponential growth in RRESN, with the increasing number of objective transitions in *TC*. Meanwhile, the blue plot line illustrates that the increased ratio of the length of generated test sequence (IRLGTS) in new method is compared with the CTEA method.

It is worth noting that, in INRES EFSM and Class II EFSM, CTEA method fails to generate all test sequences in respect to the goal *TCs* in E2, due to the state explosion problem. Specially, when deriving the test sequence for *TC*13 in INRES model as shown in Fig. 5(a), the number of traversed nodes in CTEA algorithm achieves the upper bound 200,000. Thus, the algorithm terminates at the searching level 16. However, the length of the test sequence successfully generated in the proposed method for *TC*13 should be 23 and resultant traversed nodes count is only 77. Accordingly, CTEA method is unable to generate test sequences for those *TCs* from *TC*13 to *TC*18. Similarly, the case also occurred in the Class II EFSM. The classic method only produces test sequences for *TCs* from *TC*1 to *TC*4. It demonstrates that the traversed nodes count in CTEA method is also increasing exponentially with the number of the transitions in *TC*. The experimental results in Fig. 5 indicate that the proposed method can effectively produce test sequences for all *TCs* in four subject EFSMs. Moreover, their corresponding average of explored node counts are 176.68, 54.35, 17.37 and 82.47, respectively.

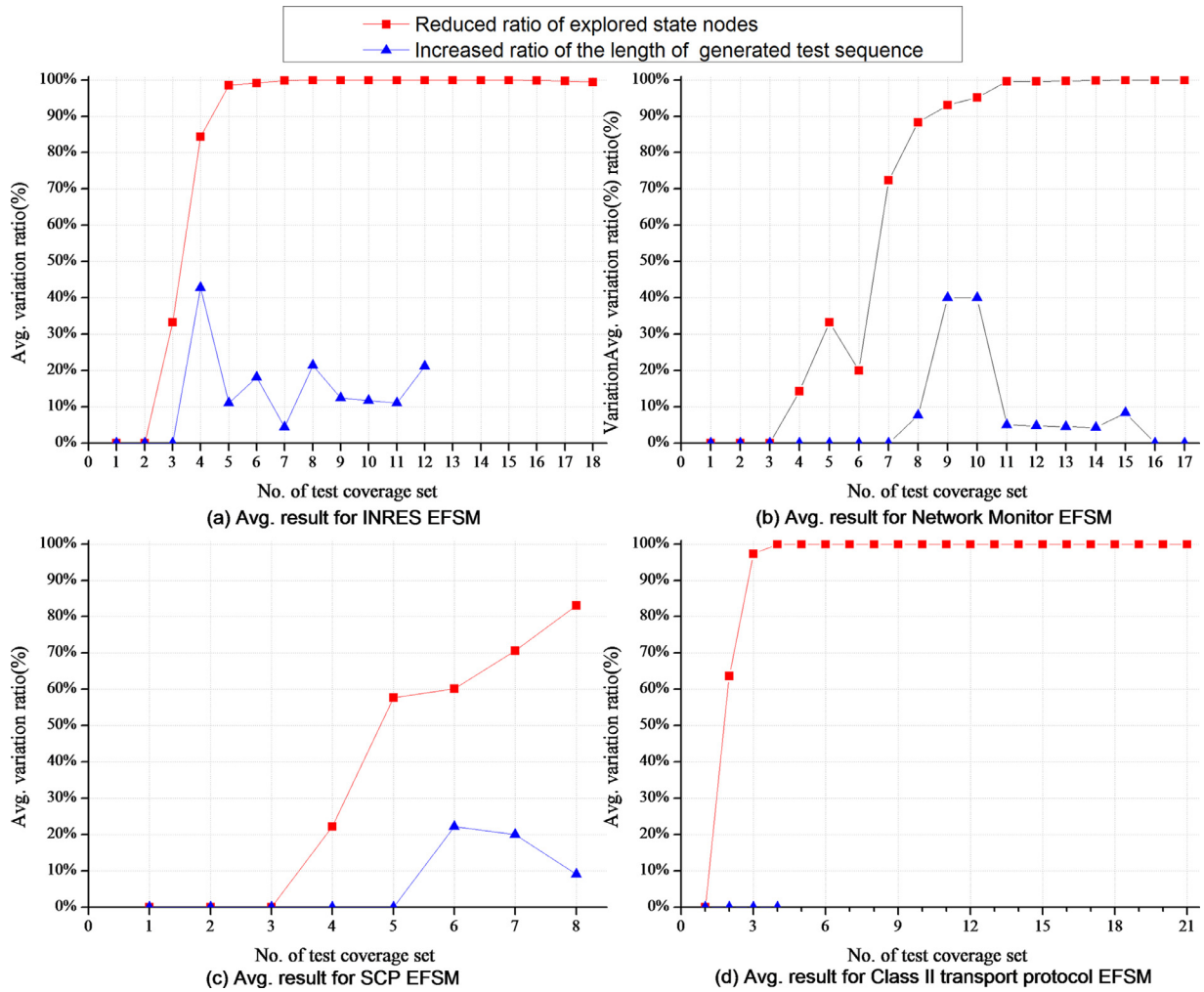

Fig. 5. Avg. experimental results of RRESN and IRLGTS for four subject EFSMs.

To further explore the relationship between the scale of searching state space and the effectiveness of our proposed algorithm, we order all the resulting NEN using CTEA method in E2 as illustrated in Fig. 6. With the increasing NEN of the TEA-based method in the BFS way, the reduced ratio of the new approach can grow exponentially. Specifically, HTEA approach can significantly reduce the number of explored states to 99.9%, when the required states count beyond 104 in the classic method. A conclusion we can draw from the experiments is that the proposed algorithm can effectively derive a feasible test sequence avoiding the state explosion problem and the resulting sequences length is close to the optimal solution in BFS way.

6. Conclusion

Feasible test sequence generation is a challenging and open problem in the field of EFSM-based protocol conformance testing. The TEA technique among reachability analysis methods, can guarantee the executiblity of generated test sequences. However, it often suffers from state explosion problem. To address this problem, we proposed a heuristic executability analysis method for test sequence generation from EFSM-specified systems. Through static analysis of the model under test, the proposed method extracts the feasibility relationship adhere in adjacent transitions of the considered EFSMs and then constructs a guidance matrix. Instead of classic BFS way, we defined an objective function, using the guidance matrix and dynamic testing context, to evaluate each explored state node and determined the next node to be traversed.

In order to check the effectiveness of the proposed approach in terms of alleviating the state explosion problem, we design two type experiments based on four popular objective EFSMs. In the first type experiment, the experimental results show that the new method can on average reduce the number of states traversed in the TEA exploration to approximately half of CTEA method. Meanwhile, the resulting test sequence length is identical to CTEA method. The results of the second

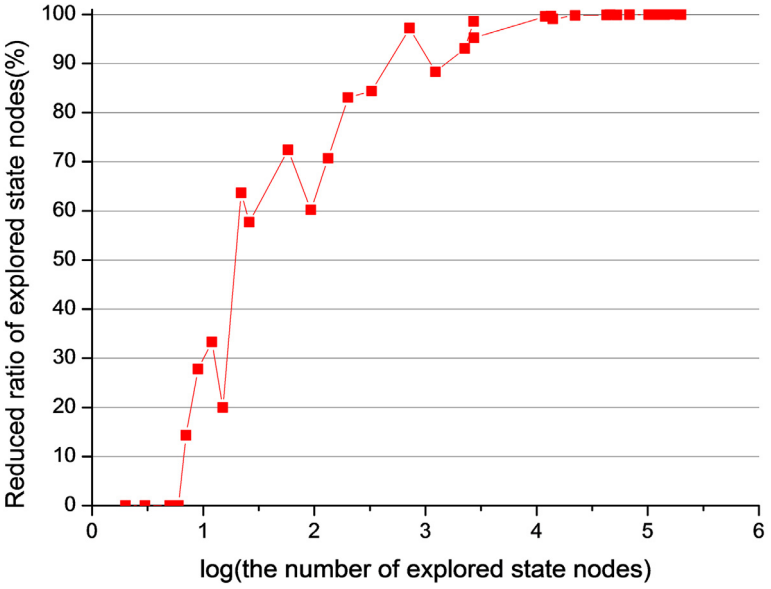

Fig. 6. The reduced ratio of explored state nodes of HTEA method.

type experiment demonstrate that our algorithm substantially outperforms the classic method, as the number of explored state nodes growing exponentially. Moreover, the new method makes a trade-off between searching efforts and test sequence length. The preliminary empirical evaluation shows the practical potential in address the state explosion problem in TEA-based methods.

Our approach presents a new heuristic framework for expanding the TEA tree, which can be applied automatically to TEA-based methods for significantly reducing the explored state nodes count. Future work for this research is to investigate the global optimization of the length of derived test sequences, through considering the combination relations among preambles, tested transitions and postambles.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61101111, 61572441), Science Foundation of Zhejiang Sci-Tech University (ZSTU) under Grant No.1004839-Y.

References

[1] H. Aljazzar, S. Leue, Directed explicit state-space search in the generation of counterexamples for stochastic model checking, Softw. Eng. IEEE Trans. 36 (1) (2010) 37–60.
[2] S. Anand, E.K. Burke, T.Y. Chen, J. Clark, M.B. Cohen, W. Grieskamp, M. Harman, M.J. Harrold, P. McMinn, et al., An orchestrated survey of methodologies for automated software test case generation, J. Syst. Softw. 86 (8) (2013) 1978–2001.
[3] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, L. Tratt, Control dependence for extended finite state machines, in: Fundamental Approaches to Software Engineering, Springer, 2009, pp. 216–230.
[4] N. Asoudeh, Y. Labiche, Multi-objective construction of an entire adequate test suite for an efsm, in: Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on, IEEE, 2014, pp. 288–299.
[5] N. Asoudeh, Y. Labiche, On the effect of counters in guard conditions when state-based multi-objective testing, in: Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on, IEEE, 2015, pp. 105–114.
[6] C. Bourhfir, R. Dssouli, E. Aboulhamid, N. Rico, Automatic executable test case generation for extended finite state machine protocols, in: Testing of Communicating Systems, Springer, 1997, pp. 75–90.
[7] J. Burnim, K. Sen, Heuristics for scalable dynamic test generation, in: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2008, pp. 443–446.
[8] A. Cavalli, C. Gervy, S. Prokopenko, New approaches for passive testing using an extended finite state machine specification, Inf. Softw. Technol. 45 (12) (2003) 837–852.
[9] S.T. Chanson, J. Zhu, A unified approach to protocol test sequence generation, in: INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE, IEEE, 1993, pp. 106–114.
[10] D. Cook, Gold parser builder, http://www.goldparser.org.
[11] A.T. Dahbura, K.K. Sabnani, et al., Formal methods for generating protocol conformance test sequences, Proc. IEEE 78 (8) (1990) 1317–1326.
[12] K. Derderian, R.M. Hierons, M. Harman, Q. Guo, Estimating the feasibility of transition paths in extended finite state machines, Autom. Softw. Eng. 17 (1) (2010) 33–56.
[13] A.Y. Duale, M.Ü. Uyar, A method enabling feasible conformance test sequence generation for efsm models, Comput. IEEE Trans. 53 (5) (2004) 614–627.
[14] Q. Guo, R.M. Hierons, M. Harman, K. Derderian, Constructing multiple unique input/output sequences using metaheuristic optimisation techniques, in: Software, IEE Proceedings-, 152, IET, 2005, pp. 127–140.
[15] R.M. Hierons, M. Harman, Testing conformance of a deterministic implementation against a non-deterministic stream x-machine, Theor. Comput. Sci. 323 (1) (2004) 191–233.

[16] R.M. Hierons, T.-H. Kim, H. Ural, On the testability of sdl specifications, Comput. Netw. 44 (5) (2004) 681–700.
[17] R.M. Hierons, H. Ural, Optimizing the length of checking sequences, Comput. IEEE Trans. 55 (5) (2006) 618–629.
[18] C.-M. Huang, M.-S. Chiang, M.-Y. Jang, Uioe: a protocol test sequence generation method using the transition executability analysis (tea), Comput. Commun. 21 (16) (1998) 1462–1475.
[19] C.-M. Huang, M.-Y. Jang, Y.-C. Lin, Executable efsm-based data flow and control flow protocol test sequence generation using reachability analysis, J. Chinese Ins. Eng. 22 (5) (1999) 593–615.
[20] A.S. Kalaji, R.M. Hierons, S. Swift, Generating feasible transition paths for testing from an extended finite state machine (efsm) with the counter problem, in: Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, IEEE, 2010, pp. 232–235.
[21] A.S. Kalaji, R.M. Hierons, S. Swift, An integrated search-based approach for automatic testing from extended finite state machine (efsm) models, Inf.. Softw. Technol. 53 (12) (2011) 1297–1318.
[22] L.-S. Koh, M.T. Liu, Test path selection based on effective domains, in: Network Protocols, 1994. Proceedings., 1994 International Conference on, IEEE, 1994, pp. 64–71.
[23] D. Lee, M. Yannakakis, Testing finite-state machines: State identification and verification, Comput. IEEE Trans. 43 (3) (1994) 306–320.
[24] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines-a survey, Proc. IEEE 84 (8) (1996) 1090–1123.
[25] S. Maag, C. Grepet, A. Cavalli, A formal validation methodology for manet routing protocols based on nodes self similarity, Comput. Commun. 31 (4) (2008) 827–841.
[26] A. Masood, A. Ghafoor, A. Mathur, Conformance testing of temporal role-based access control systems, Depend. Secure Comput. IEEE Trans. 7 (2) (2010) 144–158.
[27] A. Petrenko, G. Bochmann, M. Yao, On fault coverage of tests for finite state specifications, Comput. Netw. ISDN Syst. 29 (1) (1996) 81–106.
[28] A. Petrenko, S. Boroday, R. Groz, Confirming configurations in efsm testing, Softw. Eng. IEEE Trans. 30 (1) (2004) 29–42.
[29] T. Ramalingom, K. Thulasiraman, A. Das, Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines, Comput. Commun. 26 (14) (2003) 1622–1633.
[30] M.Ü. Uyar, A.Y. Duale, Test generation for efsm models of complex army protocols with inconsistencies, in: MILCOM 2000. 21st Century Military Communications Conference Proceedings, 1, IEEE, 2000, pp. 340–346.
[31] W.E. Wong, A. Restrepo, B. Choi, Validation of sdl specifications using efsm-based test generation, Inf. Softw. Technol. 51 (11) (2009) 1505–1519.
[32] Y. Xia, Y. Jiangyuan, W. Zhiliang, B. Jun, W. Jianping, Modeling and testing of network protocols with parallel state machines, IEICE Trans. Inf. Syst. 98 (12) (2015) 2091–2104.
[33] R. Yang, Z. Chen, B. Xu, Z. Zhang, W. Zhou, A new approach to evaluate path feasibility and coverage ratio of efsm based on multi-objective optimization., in: SEKE, 2012, pp. 470–475.
[34] R. Yang, Z. Chen, Z. Zhang, B. Xu, Efsm-based test case generation: Sequence, data, and oracle, Int. J. Softw. Eng. Knowl. Eng. 25 (04) (2015) 633–667.
[35] T. Yano, E. Martins, F.L. De Sousa, Most: a multi-objective search-based testing from efsm, in: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE, 2011, pp. 164–173.
[36] S. Yoo, M. Harman, Pareto efficient multi-objective test case selection, in: Proceedings of the 2007 international symposium on Software testing and analysis, ACM, 2007, pp. 140–150.
[37] J. Zhang, X. Chen, X. Wang, Path-oriented test data generation using symbolic execution and constraint solving techniques, in: Software Engineering and Formal Methods, 2004. SEFM 2004. Proceedings of the Second International Conference on, IEEE, 2004, pp. 242–250.