International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

EFSM-based Test Case Generation: Sequence, Data, and Oracle

Rui Yang
State Key Laboratory for Novel Software Technology, Nanjing University,
Department of Computer Science and Technology, Nanjing University
Nanjing, 210046, China
ruizi2000@gmail.com

Zhenyu Chen
State Key Laboratory for Novel Software Technology, Nanjing University,
Nanjing, 210046, China
pyzychen@gmail.com

Zhiyi Zhang
State Key Laboratory for Novel Software Technology, Nanjing University,
Nanjing, 210046, China
Xianlingzibiying@gmail.com

Baowen Xu*
State Key Laboratory for Novel Software Technology, Nanjing University,
Department of Computer Science and Technology, Nanjing University
Nanjing, 210046, China
bwxu@nju.edu.cn

Model-based testing has been intensively and extensively studied in the past decades. Extended Finite State Machine (EFSM) is a widely used model of software testing in both academy and industry. This paper provides a survey on EFSM-based test case generation techniques in the last two decades. All techniques in EFSM-based test case generation are mainly classified into three parts: test sequence generation, test data generation, and test oracle construction. The key challenges, such as coverage criterion and feasibility analysis in EFSM-based test case generation are discussed. Finally, we summarize the research work and present several possible research areas in the future.

Keywords: Extended Finite State Machine; coverage criterion; test case generation; test sequence; test data; test oracle.

*Corresponding author.

Acronyms

FSM	Finite State Machine
EFSM	Extended Finite State Machine
SP	State Path
TP	Transition Path
SDL	Specification and Description Language
ESTELLE	Extended State Transition Language
LOTOS	Language of Temporal ordering specification
SUT	System Under Test
UIO	Unique Input/Output (Sequences)
EUIO	Extended UIO (Sequence)
CIUS	Context Independent Unique Sequence
TEA	Transition Executability Analysis
A-use	Assignment-Use
I-use	Input-use
P-use	Predicate use
C-use	Computation use
D-use	Definition-use
IO-df Chains	Input/Output Definition Chains
DFG	Data Flow Graph
CFG	Control Flow Graph
UTS	Unified Test Sequence
CCS	Cyclic Characterizing Sequence
CS	Characterizing Sequence
TDG	Transition Dependence Graph
CSP	Constraint Satisfaction Problem
EDSS	Executable Switching Sequence
EDC-path	Executable DO-path
EC-path	Executable Control Path
EBP-path	Executable Back Path
BFS	Bread First Search
NFEFSM	Normal Form EFSM
EEFSM	Expanded EFSM
PEEFSM	Partially Expanded EFSM
GA	Genetic Algorithm
SS	Scatter Search
NNEOC	Number of Numerical Equal Operators in Conditions
NNEV	Number of Numerical Event Variables
LPEV	Length of Path with Event Variables

1. Introduction

Software testing is an important activity of the software life-cycle. It is estimated that the test cost may occupy about 30% to 50% of software development [1]. Test automation has become a tendency due to its capability of reducing cost and improving effectiveness. Automated test case generation, which has a strong impact on the testing process [2], becomes an important means. One of the common approaches of automated test case generation is to create a model and utilize the model to generate test cases [3]. In general, model-based testing is to derive, execute and evaluate test cases by means of creating models in different abstraction levels. Test cases are generated based on the model to reveal faults and verify whether the implementation conform to its specification.

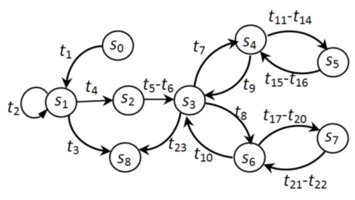
Finite State Machine (FSM) is a behavior model which consists of a finite number of states, transitions between those states which can be described by a finite input/output set, state transition functions and output functions [4]. In practice, many systems usually contain both control parts and data parts, which cannot be represented in FSM. This inspires researchers to design an extended model, called Extended Finite State Machine (EFSM) [4]. EFSM consists of states, predicates and assignments with respect to variables among transitions, such that it can represent both control flow and data flow of complex systems. In the past years, EFSM has been widely used in communication protocols, software development, software testing, sequential circuits, and other areas.

EFSM-based test case generation has been intensively and extensively studied in the past decades. A survey which reviewed the usage of formal specifications to support testing has been proposed [5]. It explored some ways that a formal specification can support testing. However, the focus was not directed on test case generation on EFSM. Dorofeeva et al. [6] reviewed FSM-based conformance testing methods and assessed their complexity, applicability, completeness, fault detection capability, the length and derivation time of their test suites in recent years. To our best knowledge, the newest survey on EFSM-based test generation was in 1996 [7]. In the past years, EFSM-based test case generation has been developed greatly. This motivates us to do a systematic survey of test case generation techniques on EFSM in recent years. This paper focuses on the principal aspects of EFSM-based test case generation: test sequence generation, test data generation, and test oracle creation. The challenges of EFSM-based test case generation, coverage criterion and feasibility analysis, will be discussed. Finally, we discuss the possible tendency and challenges for EFSM-based test case generation in the future.

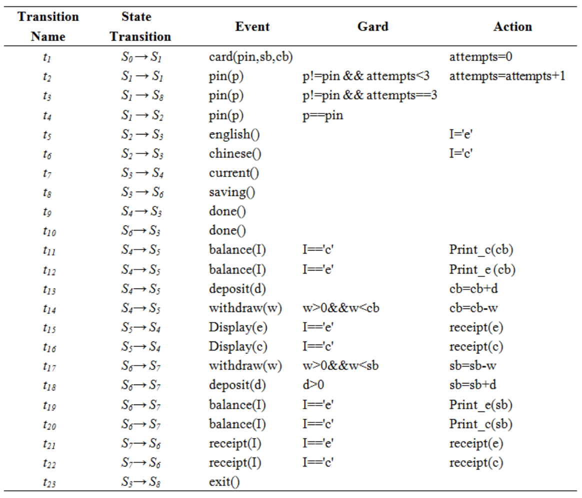
Although there are many existing works on EFSM-based test case generation. There are some challenges to achieve more effective testing. The conflict correlations among some transitions that make some paths infeasible, and the problem that detect infeasible path is undecidable in general [8]. For this reason, FSM-based test generation methods cannot be used directly for EFSM-based testing. In addition, the generation of test data to cover a given feasible sequence (path) and automated test oracle construction are more challenging tasks. In order to clearly summarize the existing work on EFSM-based test generation, we decompose a test case into three parts: test sequence, test data and test oracle. In general, EFSM-based test case generation mainly includes three steps:

(1) Test sequence generation for the specified coverage criterion, based on state identification sequence.
(2) Test data generation for covering the state identification sequence and test sequence.
(3) Test oracle construction for the generated test data.

The rest of this paper is organized as follows. Section 2 introduces some basic concepts for EFSM and the related background on testing. Section 3 summarizes test sequence generation and feasibility analysis. Section 4 summarizes test data generation. Section 5 summarizes the test oracle construction. Section 6 discusses some possible opportunities for future research.



(a)



Transition Name	State Transition	Event	Gard	Action
t_1	$S_0 \rightarrow S_1$	card(pin,sb,cb)		attempts=0
t_2	$S_1 \rightarrow S_1$	pin(p)	p!=pin && attempts<3	attempts=attempts+1
t_3	$S_1 \rightarrow S_8$	pin(p)	p!=pin && attempts==3	
t_4	$S_1 \rightarrow S_2$	pin(p)	p==pin	
t_5	$S_2 \rightarrow S_3$	english()		I='e'
t_6	$S_2 \rightarrow S_3$	chinese()		I='c'
t_7	$S_3 \rightarrow S_4$	current()		
t_8	$S_3 \rightarrow S_6$	saving()		
t_9	$S_4 \rightarrow S_3$	done()		
t_{10}	$S_6 \rightarrow S_3$	done()		
t_{11}	$S_4 \rightarrow S_5$	balance(I)	I=='c'	Print_c(cb)
t_{12}	$S_4 \rightarrow S_5$	balance(I)	I=='e'	Print_e (cb)
t_{13}	$S_4 \rightarrow S_5$	deposit(d)		cb=cb+d
t_{14}	$S_4 \rightarrow S_5$	withdraw(w)	w>0&&w<cb	cb=cb-w
t_{15}	$S_5 \rightarrow S_4$	Display(e)	I=='e'	receipt(e)
t_{16}	$S_5 \rightarrow S_4$	Display(c)	I=='c'	receipt(c)
t_{17}	$S_6 \rightarrow S_7$	withdraw(w)	w>0&&w<sb	sb=sb-w
t_{18}	$S_6 \rightarrow S_7$	deposit(d)	d>0	sb=sb+d
t_{19}	$S_6 \rightarrow S_7$	balance(I)	I=='e'	Print_e(sb)
t_{20}	$S_6 \rightarrow S_7$	balance(I)	I=='c'	Print_c(sb)
t_{21}	$S_7 \rightarrow S_6$	receipt(I)	I=='e'	receipt(e)
t_{22}	$S_7 \rightarrow S_6$	receipt(I)	I=='c'	receipt(c)
t_{23}	$S_3 \rightarrow S_8$	exit()		

(b)

Fig. 1. (a)A simplified EFSM of Automated Teller Machine (b)The detail information of Automated Teller Machine

2. Overview of EFSM

2.1. *Model Definition*

An EFSM model can be represented as a 6-tuple M=(s_0, S, V, I, O, T), where $s_0 \in S$ represents the initial state of the EFSM, S represents a finite set of state, V represents a finite set of the context variables, T represents a finite set of transitions,

I and O represent a set of transition inputs and a set of the outputs respectively. Each transition $tx \in T$ also can be formalized as 6-tuple $tx= (s_i,\ s_j,\ P_{tx},\ A_{tx},\ i_{tx},\ o_{tx})$, where s_i, s_j, $i_{tx} \in I$ and $o_{tx} \in O$ represent the start state of transition *tx*, the end state of transition *tx*, an input parameter and $o_{tx} \in O$ output results, respectively. In addition, P_{tx} represents the predicate conditions with respect to context variables and A_{tx} represents the operations with respect to current variables (variables in the current scope). In some papers P_{tx} and A_{tx} are also called Guards and Actions. For the sake of clarity, EFSM models can be represented as a directed graph $G(V\ ,\ E)$. The elements of V and E are called vertices (represent states of EFSM) and edges (represent transitions of EFSM) (See Figure 1). Initially, the EFSM is at an initial state s_0 associated with the initial variable values. The transition $t0$ will occur if the current values of the variables or input parameter i_{t0} are valid for the predicate condition P_{t0} associated with this transition. In this process, the action A_{t0} associated with this transition is then executed, which can modify the variables or produce some output results o_{t0}, meanwhile, the state of EFSM will be transformed from start state s_0 to the next state. After a series of state transferring, the EFSM will be left in a certain state s_i. Similarly, if the predicate, which may associated with current variables or input parameter i_{ti} of current transition, could be satisfied, the EFSM outputs o_{ti}, changes the current variable values, and moves to the next state s_j. This state sequence constitutes a State Path (SP) whereas this transition sequence constitutes a Transition Path (TP).

Definition 1: A state path (SP) of an EFSM is a sequence of states $s_1\ s_2\ ...\ s_n$, if there exist transitions from state $s_i\ to\ s_{i+1}$, where $i \in \{1, n-1\}$.

Definition 2: A transition path (TP) of an EFSM is a sequence of transition $t_1\ t_2\ ...\ t_n$, where every transition t_i starts from the state that the end state of the previous transition t_{i-1}, where $i \in \{2, n\}$.

In fact, EFSM is an enhanced model that extends from FSM. If all the predicates always true and the variable set is empty, the EFSM degenerate into the FSM. Therefore, FSM can be viewed as a subset of the EFSM. In terms of EFSM, some transitions may associate with complex conditions that difficult to be satisfied, whereas some transitions may associate with no predicate conditions. If associated predicate conditions in a transition path can never be satisfied, the transition path is regarded as infeasible. However, the detection of an infeasible path is generally undecidable [8]. The existence of infeasible paths is a challenge of the automated test case generation for EFSM.

For the complexity of EFSM, in some case more than one transition associate with the same start state and the end state, and these transitions may be event-driven. Therefore, the relationship between states and transitions may be one-to-many. In addition, some EFSMs contain self-loop transition which is a transition that has the same start state and the end state. An EFSM is deterministic if any group of transition has the same input that changes a state, and the guards of more than one transition cannot be satisfied at the same time in this transition group [9]. Conversely, an EFSM is non-deterministic. In addition, some EFSMs do not exist exit state, in contrast, others contain both exit state and initial state.

2.2. *Model Representation*

Before further handle of abstract EFSM, we need to transfer the model into the description or the data structure that can be understood by computers. SDL (Specification and Description Language) [10], ESTELLE (Extended State Transition Language) [11], LOTOS (Language of Temporal ordering specification) [12] and State Transmission Table are the common form to describe EFSM.

SDL is a specification and description language defined by ITU-T in 1976. By means of SDL, a system can be described as a set of interconnected abstract models which are EFSM. Generally, the application of SDL mainly includes process control and real-time systems. A system described by SDL usually consists of the following aspects: system structure, system behavior, inheritance, communication and data operation. The formality of SDL helps to improve the accuracy of system representation, and the semantics of each symbol are precisely defined. Therefore, it is appropriate for generating test cases and determining whether a system complies with the specifications.

ESTELLE is another formal descriptive language for EFSM. It has a strong ability of expression and definition, and can be used to describe both distributing and paralleling system precisely. The grammar and data type of ESTELLE are based on Pascal language, thus it is easy to use. ESTELLE can describe the system behavior as a set of hierarchically-structured modules. A module also can be structured into sub-modules. The state of the module is defined as a configuration set of variable and its corresponding value. The interaction between states is named transition which might include predicate conditions and context variables that are dependent on input parameters.

LOTOS takes advantage of formal descriptive language to ensure disambiguating and convenient to consistency analysis and testing. LOTOS consists of two parts: process algebra and data algebra, where the process algebra is utilized to describe dynamic behaviors of systems, and data algebra is utilized to describe data structures and expressions based on the abstract data type language (ACT ONE). LOTOS can be utilized to describe concurrency, non-deterministic, and synchronous or asynchronous communication systems. It supports various levels of abstraction and provides several specification styles. Since EFSM model is applied more and more in various fields other than in the communication, apart from the aforementioned specification languages, some researchers have been using other methods to describe EFSM models, such as static transfer table, and even program code. With the development of techniques, it is believed that there will be more expressing approaches for EFSM.

2.3. *Coverage Criteria*

In software testing, exhaustive testing for most of the systems is impossible, so it is necessary to select one or several test coverage criteria to obtain corresponding test cases so as to measure test adequacy and decide when to stop testing. There exist many coverage criteria, mainly including code-based coverage (white-box coverage criteria), requirement-based coverage (black-box coverage criteria) and model-based coverage criteria etc. This large set of test coverage criteria can be applied in various types of testing, and different test coverage criteria can be combined to boost the quality of testing.

EFSM contains control flow and data flow, some of their behavior is similar in some ways to a structured program, hence most of structure-based coverage criteria may also be introduced to EFSM-based test case generation. In addition, some of model-based coverage criteria can be used on EFSM-based test case generation naturally. Some available coverage criteria can be utilized for EFSM mainly includes Control-flow-oriented coverage criteria (such as decision coverage, condition coverage, full predicate coverage, etc.), Data-flow-oriented coverage criteria (such as all-defs coverage, all-uses coverage, all-def-use-paths coverage, etc.), and State/Transition-oriented coverage criteria (such as all-states coverage, all-transitions coverage, all-one-loop path coverage, etc.) [13].

In addition, some extensions of the above coverage criteria, as well as some black-box coverage criteria, can also be applied in the EFSM model. These coverage criteria do not appear in this paper due to the limitation of space. The selection of the coverage criteria needs a balance between test effect and test cost. A strong coverage criterion, which can achieve more comprehensive testing, often requires much cost to generate test cases to satisfy it. In contrast, a weak coverage criterion, which requires less cost to generate test cases to satisfy it, often worsens the confidence over the correctness of the software. Therefore, when choosing the coverage criteria, it needs to trade off the costs and effects of testing. Presently, the correlation between coverage criteria and test effects is also a research topic. However, it is not in the scope of this paper.

2.4. *EFSM-based Test Case Generation*

A high-level overview of automated model-based test case generation usually contains two aspects: create a model of the software under test and drive the model to generate test cases. EFSM-based test case generation also follows this process. We further summarize EFSM-based test case generation process into the following major steps of subdivision, as shown in Figure 2. The first step is to model the EFSM from the specification or requirement of the system. The model usually does not reflect the whole of the real system, otherwise, the scale of the model may become uncontrollable. Generally speaking, only the key aspects that to be tested are concerned during the modeling process. The second step is to generate the sequence from the EFSM to achieve testing. For EFSM, the complete sequence may include

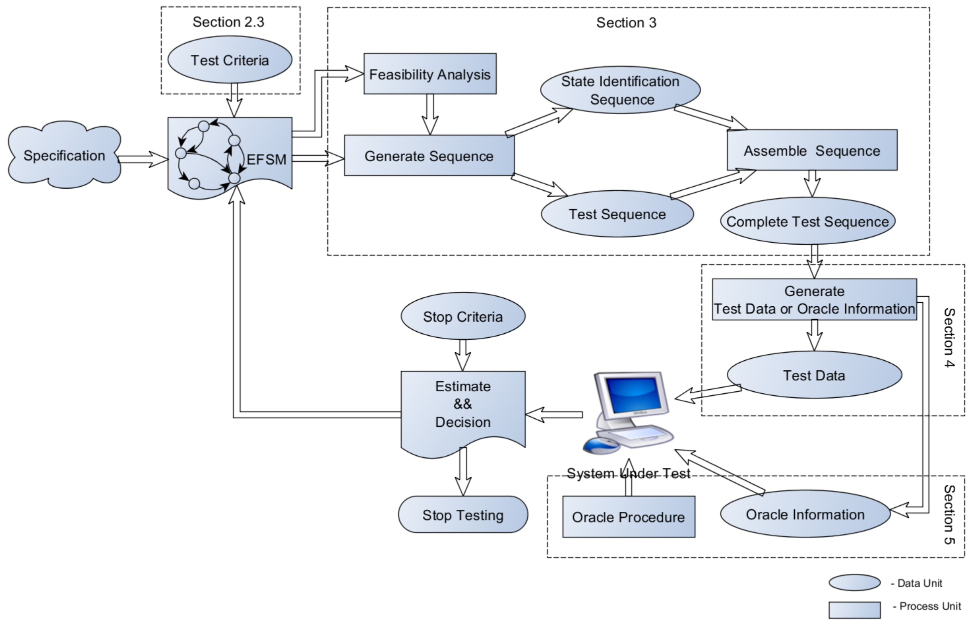


Fig. 2. The Process of EFSM-based Test Case Generation

state identification sequence and test sequence (the detail described in section 3). In order to avoid the infinite number of sequences, at least one coverage criterion should be specified. Due to the features of EFSM, some sequences may be infeasible, hence feasible analysis should be introduced. The third step is to assemble state identification sequence and test sequence into the complete sequence and generate test data to trigger them. Ideally, oracle information should also be generated in this step. The next step is to execute the test cases on the real system with online or offline way (refer to section 5), this step also involves oracle procedure (refer to section 5). The final step is to analyze and estimate the results of the test execution so as to decide to continue or stop the testing by stop criteria. The dashed box indicates that the relevant steps are discussed in the corresponding section of this paper.

Although some research has been proposed some EFSM-based testing techniques, automated test case generation on an EFSM suffers from a number of problems. The first problem that prevents EFSM-based test case generation from becoming a practical technique to aid testing is model acquisition. Despite the fact that there exist a number of automated test case generation techniques for EFSM, most EFSM constructing processes are still manually or semi-manually. The second problem is the existence of infeasible test paths (sequences), that is to say no input data can traverse them. The presence of infeasible paths (sequences) creates difficulties for test case generation. There are some studies on infeasible paths of

Table 1. Comparison and Analysis of EFSM-based test case generation techniques (symbol "-" means not mentioned or not considered)

Literatures	Coverage Criteria	State Identification	Test Sequence Generation and Feasible Analysis	Test Data Generation	Test Oracle
Li [23]	-	EUIO	-	-	-
Huang [36]	def-output-path	-	TEA	-	-
Huang [26, 37]	def-output-path	UIO$_E$	TEA	Random	-
Petrenko [29]	-	CCS	-	-	-
Ural [30]	IO-df-Chain	-	Test Sequence Selection	-	-
Miller [31]	def-observation path	UIO	Converting EFSM to FSM	-	-
Chanson [32]	du-path	CCS	CSP,Transition Loop Analysis	-	-
Chanson [66]	du-path	CCS	CSP,Transition Loop Analysis	Symbolic Execution	-
Ramaligom [24, 25]	trans-CIUS-set, def-use-ob	CIUS	Combinatorial Optimization,Partial Enumeration	-	-
Koh [34]	du-path	UIO	Loop Insertion	-	-
Pang [40]	-	-	Expanding EFSM to FSM	-	-
Hierons [38, 39]	all-uses	-	Expanding EFSM to FSM	-	-
Bourhfir [45]	du-path	UIO	Symbolic evaluation	Random	-
Ural [42],Duale [43]	-	-	Simplex Algorithm, Graph Splitting	-	-
Wong [47, 48]	all-edge	-	CSP	Symbolic Execution	-
Derderian [50]	-	-	Feasible Estimate	-	-
Kalaji [51, 52]	-	-	Genetic Algorithm	-	-
Kalaji [72]	all-transition	-	Genetic Algorithm	Genetic Algorithm	-
Yano [53, 54]	-	-	M-GEO$_{vsl}$	M-GEO$_{vsl}$	Oracle Information
Zhang [70]	-	-	-	Interval Gradient Descent	-
Zhan [68]	-	-	Symbolic Execution	Symbolic Execution	-
Yang [58, 59, 74],Zhang [75]	all-transition	-	Candidate Paths Generation Algorithm,Feasible Evaluation Metric	Scatter search,Runtime information Feedback	Oracle Information
Wu [60]	all-transition	-	Heuristics Search	Symbolic Execution	-

structural program testing [14–18], However, the case of EFSM is more complexity than structural program since a transition in an EFSM may be triggered by three types of enabling conditions: the input event, the current state and a boolean expression involving the context variables [19], and multi-transitions or self-loops may be exist between two states or single state, respectively. This feature also results in a somehow different between EFSM-based and traditional structure-based test case generation. The other problem relates to test oracle which refers to the process of obtaining the expected outputs of the EFSM and comparing them with the actual outputs of the SUT. In fact, these are not problems unique to EFSM-based testing. However, the methods, which attempt to solve these problems according to some associated feature of EFSM, are needed. Though it is impossible to solve all problems completely up to now, lots of previous work has focused on the aforementioned problems, a topic to which we now turn. The aforementioned steps for EFSM-based test case generation are the guidelines for our review, some challenges are also discussed in this process. It should mention that the relationship of the steps is not the respective independence, there have some overlaps between them. We analyzed the number of publications by the used coverage criteria and techniques. To give an overview, the results are listed in advance (see Table 1), the details are presented in the subsequent sections.

3. Sequence Generation

In terms of EFSM-based testing, after the determination of the coverage criteria, the next step to do is finding the sequence (path) set which can identify the current state and the sequence (path) set which can be followed to satisfy the specified coverage criteria. The interrelation between data flow and control flow may cause the infeasible test sequence, hence investigating the potential relationship between actions and predicate conditions is the key issue of detecting infeasible sequence. If a mass of infeasible sequences (paths) can be detected previously, the performance of the test case generation process can be improved greatly by avoiding these infeasible paths. Unfortunately, detection of these infeasible sequences is generally undecidable [8]. According to the test coverage criteria, EFSM-based testing generally includes data flow-based testing, control flow-based testing, and the mixed utilization of above two strategies. In addition, state identification is also a challenge in EFSM testing. State identification methods are used to check whether the system under test (SUT) stays at a certain state after a transition occurred. Generally, the test sequence generation on EFSM model can be split into following two steps:

(1) Generate a state identification sequence for each state of EFSM;
(2) Generate a sequence set and its corresponding data, which satisfy the specified coverage criteria, to check whether the transition is correctly triggered, then concatenate the state verification sequence to the path, so as to obtain a complete test sequence.

Therefore, it can be perceived that the test case generation process and test coverage criteria are closely linked, and test sequence feasibility is also the test sequence had to face. The following section will be organized in accordance with these clues.

3.1. *State Identification Sequence Generation*

In terms of FSM-based testing, UIO(Unique Input/Output) sequence is often utilized to confirm the end state of transition on FSM model. Compared with other state verification sequences, the UIO sequence in practice often obtains shorter test sequences, and nearly all FSMs have UIOs for each state [20, 21].

Definition 3: The UIO sequence of a state s_i which denoted UIO(s_i)=(i_1/o_1)(i_2/o_2) ... (i_n/o_n) is an input/output sequence start from s_i, for all other state t, there is no input/output sequence same as s_i, where n is the length of the sequence.

According to the definition, on the assumption that the FSM model is in a certain state, if the model receives an input sequence of corresponding UIO, the output sequence is different from any other output of other states. Therefore, UIO sequence can be used to identify a certain state of the FSM. The basic method of computing UIO sequence usually can be split into following steps [7]:

(1) For each transition label of the model, compute the list of transitions that

connect with that transition label;

(2) Compute the input/output sequence of each state, the length of input/output sequence is *1*;
(3) If the above state sequences are unique, a UIO sequence for the state is found. Otherwise, compute the sequences of length 2 for the state without UIO sequence;
(4) Continue to compute the input/output sequence of length $i+1$ for state without UIO sequence, until the corresponding UIO sequence is found, where $i>1$.

However, state identification methods of FSM cannot be directly used in EFSM testing due to the existence of conflict between action statements and predicate conditions may lead to UIO sequence infeasible (unexecutable). Chun et al. [22] are the earliest to apply the UIO sequences for confirming the end state of a transition in an EFSM model, but they didn't provide the definition and the computation method of an executable UIO sequence for an EFSM state. Li et al. [23] first explicitly addressed the executability of a state identification sequence of an EFSM. They proposed a control-flow based test case generation method for an EFSM which has only integer variables and parameters. A new type of UIO sequence which named Extended UIO sequence (EUIO) was introduced. In fact, EUIO is an extension of a UIO sequence. However, the problem of finding whether a given UIO sequence has an EUIO sequence is generally undecidable [23]. Ramaligom et al. [24,25] proposed the concept of Context Independent Unique Sequence (CIUS) to resolve the infeasible problem. The CIUS is independent of the context of the specific state to be verified. Therefore, CIUS can be utilized to generate an executable test sequence for state identification by concatenating any executable switching sequence, which makes an EFSM from the initial state to a specific state, and the corresponding CIUSs. The limitations of these methods are: (1)not every EFSM has the CIUS or UIO sequence; (2)the obtained CIUS or EUIO may be a longer executable UIO sequence for a specific state; (3)the generation process is complicated, hence the scope of application is limited to some degree.

In order to overcome the aforementioned problem, Huang et al. [26] proposed a method named UIO_E for control flow protocol test sequence generation on EFSM. In this method, a shortest UIO_E sequence is generated for identifying the reachable state in TEA (Transition Executability Analysis) tree, and switching sequences are generated for concatenating those test events into an executable test sequence during one-pass generation procedure. In general, UIO_E method contains three steps as follows:

(1) Transform each EFSM specification to a normalized EFSM specification by means of the method described in the literature [27]. Normalized EFSM means each transition of EFSM is associated with a predicate condition, and there is no predicate condition in the action statement, that is to say, the Guard for deciding the executability and the associated Action become

explicit;

(2) From the initial state of EFSM, generating a transition executability analysis (TEA) tree that contains the executable transitions and reachable states. By means of the TEA tree, each transition in the EFSM is selected once to be tested by control flow based test criterion. Based on the criterion, the UIO of the tail state is generated from its executable outgoing transitions by TEA;
(3) Concatenate all of the test events to form the final test sequence by using the TEA tree. If an EFSM model is strongly connected, and if each reachable state configuration has at least one feasible transition, then UIO$_E$ method can always generate a feasible test sequence. However, UIO$_E$ method may suffer from the state explosion problem.

Based on Huang's [26] work, Zhou et al. [28] introduced the concept of invertibility which is used to optimize transition executability analysis (TEA) method, so as to shorten the length of test sequence and decrease the required TEA extension space.

Petrenko et al. [29] tried to address the problem of state identification by means of configuration confirming sequence. They try to find a more powerful confirming sequence for a given configuration and an arbitrary set of configurations, that is, a sequence such that the EFSM in the expected configuration produces an output sequence different from that of any other configuration in the given set or at least in a maximal proper subset. This method does not attempt to distinguish a configuration from all other configurations, but only from suspicious configurations (a realistic "black list" of states or configurations). Suspicious configurations set does not need to contain all possible configurations. It may consist of several configuration subsets that defined by fixing certain context variables. Different from all other aforementioned methods, this method not only identifies the end state of transition, but also identifies the configuration of the end state and the context of EFSM. However, The authors do not provide a method that how to produce suspicious sets.

3.2. *Test Sequence Generation*

In terms of EFSM-based testing, traditional FSM-based methods such as UIO sequence, the DS sequence and the W-set cannot be applied due to data flow part implied in the EFSM will affect the system behavior and needs to be tested. Furthermore, the interaction of the control flow and data flow leads to some test sequences infeasible. Dissimilarity to FSM-based testing, data flow based coverage becomes an important coverage criteria for EFSM-based testing. Therefore, we first review some test sequence generation techniques from the viewpoint of data flow coverage criteria in this subsection. Afterwards, we will focus on feasible test sequence generation techniques.

3.2.1. *Data Flow Coverage Based Test Sequence Generation*

For the sake of clarity, some definitions of data flow testing are listed as follows.

Definition 4: A-use (Assignment use) with respect to variable v is when v appears in an assignment expression of transition t; I-use (Input use) with respect to variable v is when v is an input parameter of transition t; P-use (Predicate use) with respect to variable v is when v appears in a predicate expression of transition t; C-use (Computation use) with respect to variable v is when the value of v is referenced in a computation statements of transition t; A-use and I-use are also called D-use (Definition use).

Definition 5: Def-clear-path: A transition path $(t_1\ t_2...t_n)$is a Def-clear path with respect to a variable v if there is no D-use of v in transitions $t_2...t_n$.

Definition 6: Du-Path: A transition path $(t_1\ t_2...t_n)$is a Du-path with respect to a variable v if there is D-use of v in transition t_1 as well as there is P-use or C-use of v in transition t_n, and transition path $(t_2...t_n)$ is a Def-clear path.

In order to utilize EFSM to achieve comprehensive testing, researchers have proposed a variety of data flow coverage criteria to generate test sequences. Ural et al. [30] proposed a method for automated selection of test sequences to test both control and data flow aspects of a protocol. The method is based on the identification and subsequent coverage of every association between each output and all those inputs which influence that output (known as the IO-df-Chains) in the specification. The coverage criteria used in the testing need to cover each IO-df-Chains at least once. However, data flow and control flow are considered separately, and the author did not mention how to generate test sequences which cover the IO-df-Chains and check their executability. In addition, Bourhfir et al. [7] found that this method cannot cover all transitions in some practice.

Miller et al. [31] first converted an EFSM to equivalent FSM with modified inputs and outputs. Consequently, after the conversion, the transition number of FSM increases but the number of states remains unchanged. Then a data flow graph (DFG) is created from FSM, test paths and test sequences are generated from DFG to cover all def-observation paths for testing both control and data flow. Finally, the method combines the control flow graph (CFG) to produce an executable sequence. This method requires that the variables used in the SUT should be accessed by the tester and that in many cases may not be satisfied. Chanson et al. [32] proposed an approach to generate a single test sequence of EFSM called unified test sequence (UTS) which combines both control flow and data flow testing. This method uses data flow analysis techniques proposed in literature [33] to find the data and control dependencies existing among transitions of EFSM to apply all Du-paths coverage. In this method, the CCS (Cyclic Characterizing Sequence) was first generated to test the control part of the EFSM model. CCS for a state is actually the concatenation of the characterizing sequence (CS) of this state, which can be generated from a FSM-based algorithm. Chanson does not define the type of characterizing sequence (CS), and it can be either a UIO, a DS , a W set or their variants. Afterwards, the method analyzes the data portion of the EFSM by means of data flow analysis technique to obtain the TDG (Transition Dependence Graph). The basic idea of TDG is similar to the program dependence graph in code-based testing. Based on TDG, the Def-use pair between transitions of EFSM with respect to a variable and its corresponding Def-clear path can be obtained. Then subsequences which cover all Du-paths and all transitions (and states) are generated based on the previous two methods. Finally, Chanson attempted to find all Du-paths that connect a variable and checked its feasibility by using CSP (Constraint Satisfaction Problem) method, if some paths were unexecutable, then added some loops to make these paths executable, since the loops might change some variables by executing a certain number of loops so as to the constraint might become satisfiable. However, not all unexecutable path can find some loops to make it executable. Moreover, this method verifies the executability after all the paths are generated, many generated paths will be discarded, which affects the test efficiency. Koh et al. [34] introduced the concept of Effective Domain to the EFSM testing. Effective Domain is used to evaluate how effective that a transition with certain variables can be tested in a given path in terms of the range of values. Firstly, this method defines a path set that meets Du-path coverage, then it appends state identification sequences to some transitions in the path set for testing control flow. The method appends state identification sequences to its occurrences that have distinct effective domains due to some transitions can appear in several paths. Ramalingom et al. [35] proposed a unified method for generating executable test cases for both control flow and data flow aspects of an EFSM. The trans-CIUS-set coverage principle is applied in the control flow aspects. In order to provide observability, the "def-use-ob" criterion, which extends from the all-uses data flow coverage criterion, is achieved for testing data flow aspects. Finally, a two-phase breadth-first algorithm is employed for generating a set of executable test sequences to cover the selected criteria. Huang et al. [36] presented an executable test sequence generation method for both data flow and control flow of EFSM. In the data flow testing, the transition paths that contain Def-uses and output uses (all Def-output paths) with respect to a variable are generated and tested. An executable test sequence (ETS) is composed of the following sub-sequences:

(1) The executable switching sequence (EDSS or ECSS): EDSS is generated based on the configuration of the initial state by means of connecting the initial state configuration to the start state configuration of the transition containing an A-use or I-use of a variable;
(2) The executable DO-path (EDC-path) or the executable control path (EC-path): EDC-path is generated based on the associated end state configuration of EDSS by means of concatenating the associated EDSS and DO-path;
(3) The executable back path (EBP-path): EBP-path is generated by means of expanding a TEA tree rooted from the end state of the executable DO-path.

In this method, the definition of the DO-path is similar to the IO-df-Chain [30], the EDSS (or ECSS) is the prefix of the final test sequence and the EBP-path is the suffix of the final test sequence. In fact, this method is a kind of reachability analysis technique for EFSM, the generated test sequences are executable. However, the derived executable test sequences depend on the values of input configuration parameters which need to assign manually. In addition, this kind of technique has a disadvantage of state explosion.

3.2.2. *Feasibility Analysis*

Early EFSM-based testing methods generally do not address the issue of the sequence feasibility. However, directly applying FSM-based methods to EFSM may result in some infeasible sequences, which are due to the variable correlation between the actions and predicate conditions. If there exists the contradiction between action and predicate statement in the test sequence, which means that there is no input parameter can satisfy this test sequence, then this test sequence is considered infeasible.

In order to generate executable test sequences, some expansion methods were proposed. Huang et al. [36, 37] proposed a method based on Transition Executability Analysis (TEA). The TEA can be used to generate an expanding tree which rooted in a given state configuration of EFSM. Each circle and each arc of TEA tree represent a reachable state configuration and an executable transition originated from the corresponding start state configuration, respectively. TEA tree can be expanded via Bread-First-Search (BFS) to generate an executable sequence of transitions by using the TEA technique and giving some variable bound. In addition, this method shortens the test sequence by utilizing the overlap of test subsequence, thus the test sequence of data flow is only considered to overlap a single control flow test sequence of an untested transition. The TEA algorithm can search for the shortest executable switch sequence and test subsequence, however, the final test sequence may be longer since the test sequence concatenated by a number of test subsequence may cover other test subsequences.

Hierons et al. [38, 39] also attempted to bypass the infeasible path problem by expanding the EFSM model. The method first builds a normal form EFSM (NF-EFSM) from an SDL specification of EFSM, then the expansion procedure may be applied on NF-EFSM. Afterwards, this method expands NF-EFSM to form EEFSM (Expanded EFSM) to simplify the test sequence generation, and all the paths in the EEFSM are feasible. Finally, the method selects an appropriate executable path set and the corresponding test data from some PEEFSM (Partially Expanded EFSM, PEEFSM) to achieve the All-uses coverage. Some researchers, such as Ramalingom et al. [35], Q. Pang et al. [40], B. Zhao et al. [41], etc. also utilized the similar method to solve the infeasible problem of EFSM.

Duale et al. [42, 43] considered the conflicting condition between two transitions of an EFSM is determined by formulating a simplified linear programming problem which ignores the cost function. Then the simplex algorithm [44] is utilized to solve linear programming problems to detect the conflict between two transitions.

The elimination of conflicts is based on a graph splitting technique which places two transitions with conflict conditions on two independent sub-graphs so as to avoid the two conflicting transitions are contained in the same path. That is to say, the conflicts of transitions are eliminated from EFSM model, and all paths of the final resulting EFSM graph are feasible. Therefore, the FSM-based test generation methods can be utilized to generate test cases. However, this method only can be utilized for EFSMs in which all operations and guards are linear.

Unfortunately, the size of the FSM obtained by the aforementioned expansion methods may be rather huge, which will add the difficulty of the test case generation. The problem of state explosion will be incurred. Chanson et al. [32] took advantage of the CSP (Constraint Satisfaction Problem) method and transition self-loop analysis to solve test sequence executability problems. This method finds all du-paths with respect to the variable, then checks the executability of each test path. If the path is unexecutable and contains loops, then some loops will be added to attempt to make the path executable. However, not all unexecutable path can find loops to make it executable. Koh et al. [34] utilized the similar method to handle some unexecutable path.

Unlike Chanson's method which confirms the executability after all the paths are generated, Bourhfir et al. [45] verified the executability during path generation which avoids generating paths will be discarded later. Cycle analysis is performed in order to find the shortest cycle to be inserted into a path so as to make it executable. The main steps are listed as follows:

(1) Generate data flow graph G of EFSM.
(2) Select an input value for each input parameter that affect control flow.
(3) Generate executable Du-paths according to data flow graph G, and remove those paths contained in other paths. Add the state identification sequence and postamble (an input sequence to return to the initial state) to each Du-path to form a complete test path.
(4) Check the executability of each test path; if unexecutable, then use the cycle analysis to make it feasible; if still not feasible, then discard it.
(5) For the uncovered transitions, add relevant paths to cover it, the paths are obtained by the aforementioned steps.

The cycle analysis method can only make part of the path feasible. Therefore the application of this method is subject to certain restrictions. Jianguo Wang et al. [46] use a similar approach to address this problem.

Wong et al. [47, 48] proposed a test generation method for SDL-derived EFSM models. The method reformats a set of EFSMs equivalent to the processes in an SDL specification and identifies its "hot spots" (states or transitions in the EFSM which should be prioritized during testing to effectively increase coverage). Then, test sequences are generated which intend to cover selected hot spots. In order to solve the infeasibility problem, a two-step approach is utilized. First, a greedy approach is used to backtrack a test sequence that covers a selected hot spot. Then, the set of constraints on the test sequence is examined by means of a constraint solver [49] to ask the solver whether a given test sequence is feasible.

Since the detection of infeasible paths is generally undecidable [8], in recent years, using the search-based algorithms to detect infeasible paths has become a hot topic. Derderian et al. [50] proposed a fitness function to estimate how easy it is to trigger a path, it also can be used as the basis of a fitness function of search-based algorithms that estimate the feasibility of a transition path by means of the type and number of the predicate operator in the given path. Derderian contends that *"="* is the most difficult type of comparison operator to be satisfied while "$\neq$" is the easiest. Therefore, different predicate operators will have different influence on the feasibility of the transition path. However, the interdependence that may exist between the transitions of the path is not taken into account in this method. Kalaji et al. [51,52] used a genetic algorithm to generate the executable path by analyzing data flow dependencies between the actions and conditions of the transitions of a path. The dependencies among transitions are represented as the penalty values. These values are used in the fitness function of genetic algorithm to generate the executable path. The suggested penalty values for all possible combinations of Guards and Actions between a pair of interdependence transitions are listed in Table 2.

Table 2. The suggested penalty values in article [51]

Guard	Action		
	op^{pv}	op^{vv}	op^{vc}
$g^{pv}(=)$	8	6	24
$g^{pv}(<,>)$	6	12	18
$g^{pv}(\leq,\geq)$	4	8	12
$g^{pv}(\neq)$	2	4	6
$g^{vv}(=)$	20	40	60
$g^{vv}(<,>)$	16	32	48
$g^{vv}(\leq,\geq)$	12	24	36
$g^{vv}(\neq)$	8	16	24
$g^{vc}(=)$	30	60	if c is different 500; else 0
$g^{vc}(<,>)$	24	48	if c is different 0; else 500
$g^{vc}(\leq,\geq)$	18	36	if c is different 500; else 0
$g^{vc}(\neq)$	12	24	if c is different 0; else 500

In the Table 2, Guard is the predicate operation. In general, the Guard (predicate) of a transition has the form $e\ g^{op}\ e'$, where e and e' are expressions, g^{op} is the predicate operator $\{<, >, \neq, =, \leq, \geq\}$, and the type of op is $\{pv, vv, vc\}$ which represents the predicate operation between parameter and variable, variable and variable, and variable and constant, respectively. The assignment represents the assignment operation, where op^{pv} represents the assignment operation between input parameter and variable, op^{vv} represents that between variables, and op^{vc} represents that between constant value and variable. If there exist interdependencies among transitions of a given path of EFSM, then the fitness function can be taken according to the types of Guard and Action to estimate the traversal complexity of this transition path and its feasibility. The fitness function value is computed by using penalty values that are listed in Table 2. In addition, some Guards with certain features, which are likely to make it difficult to satisfy, are also penalized. The computing process performs backward data flow dependence analysis for a variable. The smaller fitness function value of a path means fewer interdependencies among transitions in the path, and it is relatively easier to trigger. Finally, this fitness metric is used to produce feasible paths by GA(Genetic Algorithm). However, all generated paths have the same length that be determined in advance manually.

Yano et al. [53,54] presented a multi-objective evolutionary method for test case generation from an EFSM. First, this method uses an EFSM dependence analysis technique which proposed in literature [55] to obtain the transitions that influence the target transition given by the transition path, and then makes use of the simplified model to generate the test sequence. This test case generation process involves two target functions: (1)the fitness function F_1 takes the evolutionary structural testing approach in literature [56] to direct the search towards the test purpose; (2)the fitness function F_2 intends to minimize the test sequence size to reduce the cost of testing. The population consists of input sequence size, sequence of input events, and parameters of all input events.

The population structure is shown in Figure 3.

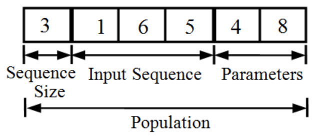


Fig. 3. The structure of population in article [53]

However, the test sequence generated by this method is disorderly, and it does not factor in corresponding relations between path and test sequence, therefore, redundant test sequences may be generated in this process. In addition, this method does not factor in coverage criterion either. In order to cope with the infeasible problem, the method transforms EFSM model into Java code by tool SMC [57], only paths triggered during code execution are considered as candidate solutions.

Yang et al. [58] proposed a method that combines static analysis and dynamic analysis techniques to address the path feasibility problem of EFSM. The authors presented a metric in order to find an appropriate path set to meet the specified coverage criterion, where "appropriate" means that the path set has fewer, longer paths and better feasibility. The presented metric is used to evaluate the path feasibility and the coverage ratio of the EFSM. In addition, semantic analysis technique is used to parse the expressions on transition to make a static abstract model executable, thus the run-time feedback information can be used to guide test data generation. This method first generates a candidate path set with loops (including self-loops) aiming to satisfy the test criterion. Then, some paths with highest probability of feasibility are selected to generate test data. Since static analysis technique may be coarse that cannot predict the infeasible path thoroughly. Therefore, the dynamic analysis technique, as a complementary of static analysis, is used to find the infeasible paths in the process of test data generation. Generally, this method includes the following steps:

(1) Candidate Path Set Generation: Generate a candidate path set with loops or self-loops from the initial state to other states in order to satisfy the test coverage criterion. A constraint condition that just contains loops or self-loops only once is imported to limit the path number.
(2) Feasibility Evaluation: Evaluate the path feasibility and transition coverage of the EFSM by a proposed metric. Then, candidate path set sorted by the metric value in order to facilitate test data generation. The presented metric is listed as follows:
$$f = \begin{cases} \frac{\sum_{i=0}^{k} v(df_i)}{|TP|^d} & If \sum_{i=0}^{k} v(df_i) \neq 0 \\ \\ 0 - |TP| & If \sum_{i=0}^{k} v(df_i) = 0 \end{cases} \tag{1}$$
Where df_i, $v(df_i)$, k , $|TP|$ and d represents the definition-p-use transition pair on a path, the penalty values of definition-p-use transition pair are extended from literature [51], the number of definition-p-use transition pairs in a path, the length of a path and a value that is used to adjust the weight of path length in the metric, respectively.
(3) Test Data Generation Process: By building an executable model, a fitness function is designed by collecting the run-time feedback information and scatter search algorithm is introduced to guide the test data generation. In this step, a dynamic analysis method based on meta-heuristic search is also utilized to find infeasible paths.

In order to improve the precision of the metric, Yang et al. [59] proposed an approach based on Multi-objective Pareto optimization technique to solve the path ordering problem. They designed two fitness functions to obtain the Pareto-optimal solutions of the path sequence, which aims to generate test data more effectively. Two fitness functions f_1 and f_2 are listed as follows:

$$f_1 = \sum_{j=0}^{n-1} \frac{C_1}{\frac{C_2}{\gamma} + (n - j + 1)} \tag{2}$$

$$f_2 = \sum_{j=0}^{n-1} \frac{C_1}{(\sum_{i=0}^{k} v(df_i) + 1) + (n - j + 1)} \tag{3}$$

Where $\sum_{i=0}^{k} v(df_i)$ is same as formula 1, j and γ represent the position of a path in the path set and the count of the distinct transitions in a path, respectively. C_1 and C_2 are constant values to make fitness value smaller, which mean a better solution in the selection process.

In fact, static analysis technique is more effective in detecting infeasible paths that contain simple data types. For complicated data types, dynamic analysis technique is still required. Compared with static analysis, dynamic analysis technology has a higher cost in detecting infeasible paths.

Table 3. Summary of feasible test sequence generation technique

Method Type	Literature	Publish Year	Key Techniques
Expansion method	Miller [31]	1992	Equivalent conversion, control flow and data flow diagram generation
	Ramalingom [35]	1995	BFS algorithm, extending the sequence by single transition step by step
	Huang [36, 37]	1995,1999	Transition Executability Analysis, BFS algorithm
	Pang [40]	1997	Equivalent transformation method, State decomposition transformation
	Duale [42, 43]	2000,2004	Simplified linear programming, simplex algorithm, graph splitting
	Hierons [38, 39]	2002,2004	Normal form EFSM producing, NF-EFSM expansion, state splitting
	Zhao [41]	2007	Modified Transition Executability Analysis
Loop analysis	Chanson [32]	1993	Constraint satisfaction, control flow and data flow analysis, loop analysis
	Koh [34]	1994	Constraint satisfaction, transition test effective evaluate, path reduction
	Bourhfir [45]	1997	DU-path generation, control flow and data flow analysis, cycle analysis
Search-based	Derderian [50]	2009	Feasible estimation, fitness function
	Kalaji [51, 52]	2009,2010	Genetic algorithm, dependence analysis, fitness function
	Yano [53, 54]	2010,2011	Multi-Objective optimization, dependence analysis, fitness function
	Yang [58, 74]	2011,2014	Candidate path generation algorithm, feasible metric, dynamic analysis
	Yang [59]	2012	Multi-Objective optimization, feasible metric
	Wu [60]	2012	Modified DFS algorithm, Evaluation metrics, path set reducing
Others	Wong [47, 48]	2008,2009	Constraint solver,dominator analysis, greedy search
	Lu [61]	2013	Backward slicing, theorem proving

Similarly, Wu et al. [60] adopted a search-based method to address the infeasible problem, they defined the penalty values based on the influential factors that observing in the experiment. The factors included: (1)Transition t or state d appeared times in the selected paths. (2)The number of transitions that started from state d and did not appear in selected paths. (3)The number of states that neighbored with state d and did not appear in selected paths. Based on the penalty values and influence factors, a fitness function was proposed for guiding the search algorithm. Lu et al. [61] utilized the slicing technique to obtain the related predicate on each transition in the path. Afterwards, whether the post-condition of the transitions associated with the predicate implying the predicate or not is utilized to determine the feasibility of the path.

For the sake of the reader, we summarize some principal feasible test sequence generation methods in Table 3. Method Type represents the type of feasible test sequence generation techniques that we classified. Literature and Publish Year represent the research paper and its publish year, respectively. In the same technique type, the paper is listed by the first year (the papers written by the same author listed in one row, therefore, some columns may have two years). Key Techniques show some key techniques that utilized for the feasible test sequence generation in the corresponding research paper. As Table 3 shows, the expansion method is a kind of mainstream technology to generate feasible sequence in the early. However, this kind of method has to face the state explosion problem. Search-based techniques and Constraint satisfaction techniques have presently became popular means in feasible test sequence generation of EFSM.

4. Test Data Generation

After generating the feasible path set which can satisfy the specified coverage criterion, it is still necessary to generate the actual test data to trigger these feasible paths. In EFSM-based testing, the automatic test data generation is still a challenge which is the lack of relevant research and far from mature. Up to now, the methods for automated test data generation for EFSM are mainly including symbolic execution, gradient descent and search-based methods.

4.1. *Symbolic Execution*

Symbolic execution [62–64] is a program analysis technique, which analyzes programs by tracing symbolic rather than concrete values. The actual program execution requires concrete values as the inputs, whereas symbolic execution uses symbols representing values of variables. The execution usually involves operations on complex expressions. The result of a program execution is usually represented as a symbolic expression.

In fact, the basic idea of symbolic execution is not complicated, symbolic execution of a program is represented as a symbolic execution tree that is defined by the execution paths which associate with all possible value assignments to input parameters, and a path condition constraint is constructed to describe the program execution along this path. Each variable appears in the path is an input of the program, when a complete path condition expression is generated, constraint solver will be utilized to solve the path condition expression, and get the result values of the input parameters. Hence, the key problem of symbolic execution is to decide whether a set of constraints can be satisfied. In recent research, for reducing the complexity of the constraints, a popular technique referred to dynamic symbolic execution [65] is proposed which replace some symbolic values with concrete values in symbolic execution process.

Symbolic execution has been widely used in code-based testing, program debugging and program optimization, etc. For EFSM-based testing, Chanson et al. [66] used the backward expansion technique of path-dependent symbolic evaluation to obtain path condition constraint. Backward expansion technique works from the final transition toward the start transition to develop the symbolic expressions. In addition, a heuristic constraint solving procedure is presented to select test data for predicate automatically by solving the constraints for the constrained parameters, and unconstrained ones are assigned values randomly. If the number of iterations exceeds the predetermined bound, the solver considers no solutions.

Xu et al. [67] implemented a tool for obtaining a deterministic EFSM from a program written in a subset of C programs. Then, Zhan et al. [60, 68] utilized the forward expansion technique which works from start state to the end state and constraint satisfaction problem (CSP) to attempt to find appropriate initial values of the variables such that the EFSM may reach a terminal state. Wong et al. [47,48] also used Zhang's constraint solver [49] to obtain test data.

In practice, symbolic execution still has some problems [69]:

(1) The difficulty of handing loops in a path, since the potential number of paths that may need to be examined.
(2) The difficulty of resolving computed storage locations, such as an array or a pointer.
(3) The difficulty of handling of procedure calls.

4.2. *Gradient Descent Algorithm*

Zhang et al. [70] proposed an automated testing data generation method ADS for EFSM. The assumption of the method is the case that all the paths of EFSM used to generate test data are feasible. The method is split into two main phases - interval narrowing and subsection gradient optimal descent algorithm. In the interval narrowing phase, a group of interval narrowing operators is designed to reduce the variables' value scope. If the precondition in the state transition is simple, the test data can be obtained in this phase. Afterwards, subsection gradient optimal descent algorithm is utilized to generate test data from the reduced domain of the variables. In the general, this method includes the following steps:

(1) Initialization: Transform the expressions included in all state transitions into regular expressions;
(2) Interval narrowing: Preconditions in the state transitions are used to reduce the value interval of the input parameter, which may increase the convergence speed of the gradient optimal descent algorithm;
(3) Test data generation: Automatically select the test data by means of the subsection gradient optimal descent algorithm.

    Subsection gradient optimal descent algorithm involves the following steps:

(a) Select the value interval of an unhandled input variable vector, and randomly select an initial point $X^{(0)}$ in this interval; if all the value intervals are handled, then this algorithm fails;
(b) Calculate unit vector $g^{(k)}$ of the negative gradient at the vector $X^{(k)}$($k = 0, 1, \ldots, n$) and its optimal step λ_k;
(c) Let $X^{(k+1)} = X^{(k)} + \lambda\, g^{(k)}$, and ensure that the value of new input variable

after interaction is limited in the value interval, that is:

$$x_i{}^{(k+1)} = \begin{cases} x_i.bottom & x_i{}^{(k+1)} < x_i.bottom \\ \\ x_i.top & x_i{}^{(k+1)} > x_i.top \end{cases} \tag{4}$$

(d) If $X^{(k+1)}$ satisfies all constraint conditions, then output this vector and quit; if $|X^{(k+1)}$- $X^{(k)}|<\eta(\eta$ is an arbitrary smaller positive number), then the gradient is considered to reach the bottom, and there is not much space for tuning, then the process goes to step (a); otherwise, goes to step (b) for further iteration.

In most cases, this method may find a group of solutions, however, in the case of having no solution of input variables, test data need to be selected from interval manually. In addition, this method does not address the infeasible problem.

4.3. *Search-based Test Data Generation*

In recent years, the utilization of meta-heuristic search techniques for the automated test data generation has been attracting more and more attention from researchers. Unfortunately, test data generation is also an undecidable problem [69]. The search-based method has provided an effective way for the automated test data generation. Meta-heuristic search techniques are high-level frameworks based on heuristics, and they aim to seek solutions to some NP-complete or NP-hard problems at a reasonable computational cost, even some problems for which a polynomial time algorithm can solve but is not practical [69]. When generating test data, it needs to transform the test criteria to the fitness functions (or objective functions). Fitness functions are used to compare and evaluate the solutions of the search with the final goal in order to generate test data automatically. Therefore, the design of fitness function is a key problem of the meta-heuristic search algorithm. In addition, the selection of crossover operator and mutation operator will influence the efficiency of the algorithm too. Presently, the popular search-based algorithms include Hill Climbing, Simulated Annealing, Tabu search and Evolutionary Algorithms (such as Genetic Algorithm, Scatter Search Algorithm, Particle Swarm Algorithm), etc.

For search-based technique of FSM-based testing, Lefticaru et al. [71] firstly apply the genetic algorithm to the test data generation on FSM.

The fitness functions of this method are designed as follows:

$$fitness = approach\_level + normalized\_branch\_level \tag{5}$$

$$normalized\_branch\_level = 1 - 1.05^{-(branch\_level)} \tag{6}$$

The *approach_level* followed the McMinn's metric [69] in evolutionary structural test data generation, which are calculated by subtracting one from the number of critical branches lying between the node from which the individual diverged away from the target node, and the target itself. *normalized_branch_level* is derived from the guard predicates which was proposed by Tracey [56] by using the objective functions for relational predicates and logical connectives. This part of fitness function is used to alleviate plateau for offering enough guidance to the search. Normalized means the *branch_level* is mapped into range [0, 1).

For EFSM-based testing, Kalaji et al. [72] proposed fitness function for evaluating transition path of EFSM as follows.

$$Path\_fitness = norm(function\_distance) + transition\_approach\_level \tag{7}$$

where

$$transition\_approach\_level = NumOfCrticalTransAwayFromTarget - 1 \tag{8}$$

$$function\_distance = norm(branch\_distance) + transition\_approach\_level \tag{9}$$

The fitness function for a path is derived in a similar way to the method proposed by Wegener et al. [73]. Given a feasible transition path, the *function_distance* is calculated for each transition with guard by applying the Wegener's method. Then, any transition that has guards is considered as a critical transition and so the value of *function_approach_level* is calculated by subtracting 1 from the number of critical transitions away from the target transition. Similar to formula 6, the *branch_distance* and *function_distance* are also normalized to a value in the range [0,1). The final fitness function *Path_fitness* is the sum of two components: branch distance and approach level.

Yang et al. [58, 74] and J. Zhang et al. [75] built an executable EFSM model by means of semantic analysis of expressions and graph traversal algorithm to generate test data. Through the utilization of execution semantics, the model can be executed like a program, and it still remains the ability of abstract expression. Therefore, the run-time feedback information, which is collected from the test execution of EFSM, can be treated as fitness function and Scatter Search (SS) algorithm [76] is utilized to guide the test data generation. Miller et al. [77] originally applied this technique in code-based testing. Furthermore, the corresponding outputs associated with generated test data are also collected to generate oracle information automatically. Finally, the complete test cases consist of test path, test data and oracles information.

Scatter Search (SS) is a meta-heuristic algorithm framework which contains five main sub-methods. Initially, SS algorithm generates a diverse solution set by using a diversity generation method, and each new solution will be processed by the improvement method and added to the population set. Some of the best solutions in the initial set will be selected to create the reference set. In the next step, solutions in the reference set are grouped into two or more subsets, then, solutions in each subset are combined to produce new individuals [78]. A reference set update method evaluates the new solution to verify whether they can update the reference set by checking the fitness value of solutions. The best solutions will be included in the reference set and the worst solutions will be dropped. This iteration continues until a stop condition is reached or the final solutions are stored in the reference set.

The main methods defined in scatter search algorithm are listed as follows:

(1) Diversification Generation Method: This sub-method is used to generate diverse solutions to fill the Population Set or Reference Set.
(2) Improvement Method: This sub-method is used to improve each new solution that generated from the Diversification Generation method and Solution Combination method by means of a local search algorithm.
(3) Reference Set Update Method: This sub-method manages to build and update the reference set by defining the necessary strategies. The reference set contains high quality solutions with the best fitness value.
(4) Subset Generation Method: This sub-method is used to generate solution subsets from the reference set through a systematic strategy. The subsets are used to create new solutions via solution combination method.
(5) Solution Combination Method: This sub-method combines the solutions in each subset to generate new solutions, which will be processed by the Improvement Method. Afterwards, these improved solutions are used to update the reference set by Reference Set Update Method.

The feedback information that collected to guide the test data generation process in SS algorithm is listed as follows:

$$F_e = \frac{|SP_t|}{|TP|} \times 100 \tag{10}$$

where $|TP|$ and $|SP_t|$ are the length of a path which is specified to generate test data and the length of a sub-path which is covered from the first transition sequentially by the current test data, respectively. If F_e equals 100, that means all transitions of the path are covered, thus the test data are generated successfully.

R. Zhao et al. [79] carried out an empirical study on the efficiency of search based test generation for EFSM models. A close positive correlation between the cost of the test case generation and the number of numerical equal operators in conditions (NNEOC) on a feasible transition path is found. When the NNEOC of a feasible transition path increases, there is a raising relationship between the test generation cost and the number of numerical event variables on a path (NNEV) or length of path with event variables (LPEV), and NNEV increases linearly with the LPEV. In addition, only when NNEOC is considerable, the exponential relationship between test generation cost and NNEV or LPEV appears strong relevance.

Similar to section 3, we summarize some principal test data generation method in Table 4. As Table 4 shows, search-based techniques are increasing introduced into EFSM-based test data generation in recent years. Compared with symbolic

Table 4. Summary of main test data generation methods

Method Type	Literature	Publish Year	Key Techniques
Symbolic Execution	Chanson [66]	1994	Backward substitution, heuristic strategy to solve constraint
	Zhang [68]	2004	Symbolic Execution, Depth-first search, forward tracking
	Wong [47] [48]	2008, 2009	Symbolic Execution, backward tracking
	Wu [60]	2012	Symbolic Execution, depth-first search, forward tracking
Search-based	Zhao [79]	2010	Genetic Algorithm, branch distance, approach level
	Yano [53] [54]	2010, 2011	Multi-Objective, executable model, Dependence Analysis
	Kalaji [72]	2011	Genetic Algorithm, branch distance function and approach level
	Yang [58] [74]	2011, 2014	Scatter Search, run-time information feedback, executable model
	Zhang [75]	2012	Scatter Search, run-time information feedback, executable model
Others	Bourhfir [45]	1997	Random
	Huang [37]	1999	Specify variable bound, random
	Zhang [70]	2003	Gradient descent, interval algebra

execution, search-based test data generation techniques can handle more data type. However, symbolic execution is proved very effective on test data generation [2]. Search-based and symbolic execution are two promising techniques for test case generation, however, both of the two techniques have their advantages and disadvantages. Therefore, there are some new trends in the research of combining the search-based and dynamic symbolic execution technique to make use of the advantages of two techniques in recent years [80–82]. The new hybrid technique may also be introduced into EFSM-based test case generation in the future.

5. Test Oracle

One of the major challenges of software testing is to create test oracle, such as some automated check on the output values to see whether they are correct, or the expected output values of the SUT, this task is more challenging than just generating test input data or test sequences without checking the results. Manually creating test oracle is a time-consuming and difficult task. Moreover, it could easily be wrong that check the behavior of a complex system manually as well as explain the detailed system specification. In addition, with the increasing amount of test results to be determined, accuracy of artificial judgment will be greatly decreased. Therefore, automatic test oracle generation is also one of the important research topics of software testing.

In order to generate complete test cases with oracles, the test case generation system should know about the expected behavior of the SUT precisely (such as the relationship between its inputs and outputs) to predict or check the output results. That is to say, the expected behavior of the SUT must be described clearly. In the broad sense, test oracle is composed of oracle information and oracle procedure [83]. Oracle information represents expected output of the model, however, the oracle information is also called the test oracle in some literatures. Oracle procedure is a process that compares the oracle information with the actual output [84], this process can be applied online (oracle procedure and test execution are done simultaneously) or offline (oracle procedure is done after test execution). In FSM-based testing, the generation of oracle information is still not straightforward even FSM only contains control flow and without complex predicate condition and behavior. For example, when the actual state of the system is not accessible, it is not easy to check that a test sequence of FSM causes the SUT to end up on the correct state. The ideal test system can get the test oracle from an abstract model or system specification automatically, and test prediction and test cases are unrelated. But in the real world, test oracle needs to check any possible outputs at a reasonable cost to determine whether the SUT executed as expected. Therefore, it is impractical to establish test oracle in accordance with all possible cases, and a trade-off between test oracle and test cost is needed. That is to say, there is a choice of test oracle strength, and the weakness or strength selection of the oracle usually relates to the adequacy test criteria. Fujiwara et al. [85] pointed out that the fact that many specifications do not satisfy the assumptions of the FSM model made by most test methods, such as minimality, complete specification, a limited number of states etc.

The above test steps employ the abstract model to generate test cases that applied in SUT, and then determine whether SUT can result in an expected behavior in accordance with corresponding oracle information, this process of which is known as active testing. Contrasted with active testing, passive testing monitors the results of SUT without introducing any special test data. Passive testing is used to determine whether a SUT is faulty by observing the input/output behaviors without interfering with its normal operations [86].

In terms of EFSM-based testing, there is still less study on test oracle. In the active testing, it needs to generate test cases containing oracle information, however, unlike FSM, EFSM model contains not only the predicate conditions, but also the operation statements. Consequently, oracle information for the test data cannot be obtained directly from the EFSM due to expected outputs may be calculated by the original input parameters or context variables. Abstract behavior of EFSM is more similar to a static program, but the abstract EFSM does not have the ability of dynamic execution. In order to generate oracle information automatically, Yang et al. [58] and J. Zhang et al. [75] built an executable EFSM model and collected run-time feedback information to generate test data. By using executable model, the corresponding outputs associated with generated test data are also collected to generate oracle information automatically. Finally, test paths, test data and oracle information are combined into complete test cases. However, the oracle procedure is not considered in this method. Yano et al. [53, 54] transformed EFSM model into Java code by tool SMC [57], it can also obtain oracle information automatically by executing the Java code. However, generating test data from a program are more difficult.

In EFSM-based passive testing, the researchers mainly observe and track the trace of SUT to obtain the test oracle, and the trace usually remains in the background since no operation is applied. Hierons et al. [87] proposed new verdicts that provide more information than the previous verdicts. New verdict functions can return verdicts based on a set of observation that cannot be returned by any single element of observation behavior. They provided a new test verdict and the concept of verdict function, the verdict of an observation set is one of the following: (1)the value correct; (2)the value incorrect; (3)the value uncertain; and (4)the value inconsistent. Cavalli et al. [88] proposed two passive testing methods for Extended Finite State Machine (EFSM) specification. The first method looks for input/output invariants in an EFSM specification and checks whether the trace resulting from the implementation is coherent with these invariants. In order to test the data flow in a strict way, the second method transforms the specification into a set of constraints and controls that the trace respects them. Lee et al. [86] used symbolic logic method to deal with the predicates and used assertions to record the relations among variables. The basic idea behind this method is to refine the valid variable value sets using as much information as possible. Ural et al. [89] proposed an EFSM-based passive fault detection method which provides information about possible starting state and possible trace at the end of passive fault detection, and Interval Refinement and Simplex methods are utilized for performance improvement during passive fault detection. Due to the fact that the test case generation is not required for passive testing, the content for such aspect is not the focus of this paper.

In addition, EFSMs have been also used for other testing areas besides test case generation, such as test case prioritization [90], test case selection [91–93], test reduction [94, 95] and dependence analysis [96–98], etc.

6. Discussion

In this paper, a literature survey of test case generation on EFSM is presented, we review and analyze the research papers in the past two decades, furthermore, the survey focuses on the main aspects of EFSM-based test case generation: model representation, test coverage criterion, test sequence (path) generation methods, executable analysis, test data generation methods, and test oracle construction methods.

EFSM model was early applied in the testing of communication protocols. With the development of model-based testing, EFSM model has been widely applied in many domains of testing. For EFSM-based testing, there have been lots of prominent studies on the state identification sequence generation. However, automated test data and test oracle generation of EFSM model are still far from mature. In addition, current research is relative isolation in the generation of feasible test path and test data, which are connected closely in practice. If the predicate conditions and operation statements contained in a test path are rather complex, the test case generation will be more difficult and costly. For this reason, EFSM-based testing demands exploring more in the relationship among test paths, test data and test oracles, also combining various analytical techniques for better test results.

We propose some possible tendency of techniques and the challenges for future EFSM-based testing as follows.

(1) With the development of software techniques, EFSM model and its vari-

ants will become more popular, and the test model will be increasingly complicated, so the detection of infeasible path in EFSM model will be more difficult. How to combine static and dynamic analysis and optimization techniques to improve the performance of the test case generation by means of avoidance of infeasible paths may be one of the most important areas of future research.

(2) As to the test data generation, the current methods are effective for generating simple type data, but relatively hard to deal with the complex data types. Future studies may focus on the generation of complex test data by making use of meta-heuristic search techniques and dynamic symbolic execution, even hybrid of two techniques.

(3) Automated generation of test oracle may also be one of the followed interest in the research, such studies in EFSM are insufficient. Generally, abstract model can only describe part of the function and behavior of the real system, and only part of the system behaviors can be tested. However, the test cases need to be utilized in real systems to compare oracle information with the actual outputs of the SUT. A precise test oracle that checks every aspect is very expensive, so the techniques of test oracle generation will face the challenges as how to generate the appropriate test oracle to trade off precise and cost.

(4) In practice, the model is often re-modified caused by the evolution of the real system, which calls for generating new test cases by using the information of historical test cases and model evolution. This process is called test augmentation. Model modification may also result in the expiration of previous test cases. Discard of such test cases will not only waste resources, but also reduce the ability of fault detection. Hence, the researchers start to pay attention to test repairing and test augmentation technique in recent years.

(5) Besides, empirical software engineering is the focus of current software engineering. It has become an important approach to explore the actual property and promote research on relevant problems following the scientific method. However, the empirical study in EFSM testing is so limited that it may be an another hotspot of software testing in the future.

Acknowledgements

The work is partially supported by the National Basic Research Program of China (973 Program 2014CB340702), the National Natural Science Foundation of China (No. 61170067 and 61373013).

References

[1] Z. Beizer, Software Testing Techniques, *2nd Ed, Van Nostrand Reinhold Company Limited*, London, 1990, ISBN-13 978-0672327988.

[2] S. Anand, E. K.Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskam, M. Harman, M. J. Harrold, and P. McMinn, An Orchestrated Survey of Methodologies for Automated Software Test Case Generation, *Journal of Systems and Software* 86(2013) 1978–2001.

[3] X. Yuan and A. Memon, Using GUI Run-Time State as Feedback to Generate Test Cases, *in Proc. 29th International Conference on Software Engineering*, IEEE/ACM, Minneapolis, US, 2007, pp.396–405.

[4] D. Lee, M. Yannakakis, Principles and Methods of Testing Finite State MachinesC A survey, *in Proc. of the IEEE* 84(8) (1996) 1090–1123.

[5] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J.Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lttgen, A. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, Using Formal Specifications to Support Testing, *ACM Computing Surveys* 41(2) (2009) 9:1–76.

[6] R. Dorofeeva, K. El-Fakih, S. Maag, R. Cavalli, and N. Yevtushenko. FSM-based Conformance Testing Methods: A Survey Annotated with Experimental Evaluation. *Information and Software Technology* 52 (2010) 1286–1297.

[7] C. Bourhfir, R. Dssouli, and E. M. Aboulhamid, Automatic Test Generation for EFSM-based Systems, *Technical Report IRO 1043*, University of Montreal, 1996.

[8] D. Hedley and M. A. Hennell, The Causes and Effects of Infeasible Paths in Computer Programs, *in:Proceedings of the 8th International Conference on Software engineering*, London, UK, 1985, pp.259–266.

[9] C. Shih, J. Huang and J. Jou, Stimulus Generation for Interface Protocol Verification Using the Non-deterministic Extended Finite State Machine Model, *in Proc. 10th IEEE International High-Level Design Validation and Test Workshop*, Napa, US, 2005, pp.87–93.

[10] Y. Gurevich, P. W. Kutter, M. Odersky, and L.Thiele, Abstract State Machines - Theory and Applications, *in Proc. International Workshop of ASM*, LNCS, 1912 (2000) 1–8.

[11] Estelle. A Formal Description Technique Based on an Extended State Transition Model, *ISO Standard OSI 9074*, 1989.

[12] LOTOS. A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, *ISO Standard OSI 8807*, 1989.

[13] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, 1st Edition, Morgan Kaufmann, San Fransisco, 2006, ISBN-13 978-0123725011.

[14] S. Saingern, C. Lursinsap, and P. Sophatsathit, An Address Mapping Approach for Test Data Generation of Dynamic Linked Structures, *Information and Software Technology* 47(3) (2005) 199–214.

[15] P. Coward, Symbolic execution and testing, *Information and Software Technology* 33(1) (1991) 53–64.

[16] R. Jasper, M. Brennan and K. Williamson, Test Data Generation and Feasible Path Analysis, *in Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis*, Seattle, US, 1994, pp.95–107.

[17] I. Forgcs and A. Bertolino, Feasible Test Path Selection by Principal Slicing, *in Proc. 6th European Software Engineering* 22(6) (1997) 378–394.

[18] D. Gong and X. Yao, Automatic Detection of Infeasible Paths in Software Testing, *IET Software* 4(5) (2010) 361–370.

[19] R. Lai, A Survey of Communication Protocol Testing, *Journal of Systems and Software* 62(1) (2002) 21–46.

[20] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian, Computing Unique Input / Output Sequences Using Genetic Algorithms, *in Proc. 3rd International Workshop on*

Formal Approaches to Testing of Software, LNCS, Montreal, Canada, 2004, pp.169–184.

[21] K. Derderian, R. Hierons, M. Harman, and Q. Guo, Automated Unique Input - Output Sequence Generation for Conformance Testing of FSMs, *The Computer Journal* 49(3) (2006) 331–344.

[22] W. Chun and D. Amer, Test Case Generation for Protocols Specified in Estelle, *in Proc. 3rd International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols: Formal Description Techniques, North-Holland Publishing*, 1990, pp.191–206.

[23] X. Li, T.Higashino, M,Higuchi, and K. Taniguchi, Automatic Generation of Extended UIO Sequences for Communication Protocols in an EFSM Model, *in Proc. 7th International workshop on Protocol test systems*, London, UK, 1994, pp.225–240.

[24] T. Ramalingom, K. Thulasiraman, and A. Das, Context Independent Unique Sequences Generation for Protocol Testing, *in Proc. 15th Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, US, 1996, 1141–1148.

[25] T. Ramalingom, K. Thulasiraman, and A. Das, Context Independent Unique State Identification Sequences for Testing Communication Protocols Modelled as Extended Finite State Machines, *Computer Communications* 26(14) (2003) 1622–1633.

[26] 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* 21(16) (1998) 1462–1475.

[27] B. Sarikaya, G. V. Bochmann, and E. Cerny, A Test Design Methodology for Protocol Testing, *IEEE Transactions on Software Engineering* SE-13(5) (1987) 518–531.

[28] X. Zhou, Y. Qu, and B. Zhao, Using Invertibility to Reduce the Length of Test Sequences in EFSM model(in chinese), J*ournal of China Institute of Communications* 21(11) (2000) 48–55.

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

[30] H. Ural and B. Yang, A Test Sequence Selection Method for Protocol Testing, *IEEE Transactions on Communication* 39(4) (1991) 514–523.

[31] R. E. Miller and S. Paul, Generating Conformance Test Sequences for Combined Control and Data Flow of Communication Protocols, *in Proc. Protocol Specifications, Testing and Verification (PSTV' 92)*, IFIP, 1992, pp.13–27.

[32] S. T. Chanson and J. Zhu, A Unified Approach to Protocol Test Sequence Generation, *in Proc. IEEE Conference on Computer Communications(INFOCOM '93)*, San Francisco, US, 1993, pp.106–114.

[33] E. Weyuker and S. Rapps, Selecting Software Test Data Using Data Flow Information, *IEEE Transactions on Software Engineering* (1985) 367–375.

[34] L. Koh and M. Liu, Test Path Selection based on Effective Domains, *in Proc. International Conference on Network Protocols(ICNP'94)*, Boston, US, 1994, pp.64–71.

[35] T. Ramalingom, A. Das and K. Thulasiraman, A Unified Test Case Generation Method for the EFSM Model Using Context Independent Unique Sequences, *in Proc. International Workshop on Protocol Test Systems (IWPTS'95)*, US, 1995, pp.289–305.

[36] C. M. Huang, Y. C. Lin, and M. Y. Jang, An Executable Protocol Test Sequence Generation Method for EFSM-Specified Protocols, *IFIP Transactions C: Communication Systems - Protocol Test Systems* (1995) 20–35.

[37] C. M. Huang, M. Y. Jang, and Y. C. Lin, Executable EFSM-based Data Flow and Control Flow Protocol Test Sequence Generation Using Reachability Analysis,

Journal of the Chinese Institute of Engineers 22(5) (1999) 593–615.

[38] R. M. Hierons, T. H. Kim, and H. Ural, Expanding an Extended Finite State Machine to Aid Testability, *in Proc. 26th Annual International Computer Software and Applications*, Oxford, UK, 2002, pp.334–339.

[39] R. M. Hierons, T. H. Kim, and H. Ural, On the Testability of SDL Specifications, *Computer Networks* 44(5) (2004) 681–700.

[40] Q. Pang, S. Cheng, and Y. Jin, Equivalent Transformation of EFSM and Protocol Conformance Testing(in chinese), *Journal of China Insititue of Communications* 18(4) (1997) 37–42.

[41] B. H. Zhao, B. Chen and Y. Qu, A Test Sequence Generation Algorithm Based on Improved Transition Executability Analysis(in chinese), *Journal of University of Science and technology of China* 37(9) (2007) 1096–1100.

[42] M. U. Uyar and A. Y. Duale, Test Generation for EFSM Models of Complex Army Protocols with Inconsistencies, *in Proc. 21st Century Military Communications*, Los Angeles, US, 2000, pp.340–346.

[43] A. Y. Duale and M. U. Uyar, A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models, *IEEE Transactions on Computers* 53(5) (2004) 614-627.

[44] D.Solow, Linear Programming: An Introduction to Finite Improvement Algorithms, *North-Holland: Elsevier Science Ltd*, New York, 1984, ISBN-13 978-0444009128.

[45] C. Bourhfir, R. Dssouli, E. Aboulhamid, N. Rico, and P. A. Aisenstadt, Automatic Executable Test Case Generation for Extended Finite State Machine Protocols, *in Proc. the 10th International IFIP Workshop on Testing of Communicating Systems*, Jeju Island, Korea, 1997, pp.75–90.

[46] J. Wang and J. Wu, An Extended Finite State Machine Based Generation Method of Test Suite(in chinese), *Journal of Software* 12(8) (2001) 1197–1204.

[47] W. E. Wong, A. Restrepo, and Y. Qi, An EFSM-based Test Generation for Validation of SDL Specifications, *in Proc. 3rd International workshop on Automation of Software Test*, Leipzig, Germany, 2008, pp.25–32.

[48] W. E. Wong, A. Restrepo and B.Choi, Validation of SDL Specifications Using EFSM-based Test Generation, *Information and Software Technology* 51(11) (2009) 1505–1519.

[49] J. Zhang and X. Wang, A Constraint Solver and Its Application to Path Feasibility Analysis, *International Journal of Software Engineering and Knowledge Engineering* 11(2) (2001) 139–156.

[50] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo, Estimating the Feasibility of Transition Paths in Extended Finite State Machines, *Automated Software Engineering* 17(1) (2009) 33–56.

[51] A. S. Kalaji, R. M. Hierons, and S. Swift, Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM), *in Proc. International Conference on Software Testing Verification and Validation*, Denver, US, 2009, pp.230–239.

[52] A. S. Kalaji, R. M. Hierons, and S. Swift, Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM) with the Counter Problem. *in Proc. 3rd International Conference on Software Testing, Verification, and Validation Workshops*, Paris, France, 2010, pp.232–235.

[53] T. Yano, E. Martins, and F. L. de Sousa. Generating Feasible Test Paths from an Executable Model Using a Multi-objective Approach. *in Proc. 3rd International Conference on Software Testing, Verification, and Validation Workshops*, Paris, France, 2010, pp.236–239.

[54] T. Yano, E. Martins, and F. L. de Sousa, MOST: A Multi-objective Search-Based Testing from EFSM, *in Proc. 4th International Conference on Software Testing, Verification and Validation Workshops*, IEEE Computer Society, Berlin, Germany, 2011, pp.164–173.

[55] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, and L. Tratt, Control Dependence for Extended Finite State Machines, *in Proc. 12th International Conference on Fundamental Approaches to Software Engineering*, 2009, pp.216–230.

[56] N. Tracey, J. Clark, K. Mander, and J. McDermid, An Automated Framework for Structural Test-data Generation, *in Proc. 13th IEEE International Conference on Automated Software Engineering*, Hawaii, US, 1998, pp.285–288.

[57] "SMC" [Online]. Available: http://smc.sourceforge.net

[58] R. Yang, Z. Y. Chen B. W. Xu, W. E. Wong, and J.Zhang, Improve the Effectiveness of Test Case Generation on EFSM via Automatic Path Feasibility Analysis, *in Proc. The 13th IEEE International High Assurance Systems Engineering Symposium*, Boca Raton, US, 2011, pp.17–24.

[59] R. Yang, Z. Y. Chen B. W. Xu, and Z. Y. Zhang, A New Approach to Evaluate Path Feasibility and Coverage Ratio of EFSM Based on Multi-objective Optimization, *in Proc. the 24th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, US, 2012, pp.470–475.

[60] T. Wu, J. Yan, and J. Zhang, A Path-oriented Approach to Generating Executable Test Sequences for Extended Finite State Machines, *in Proc. 6th International Symposium on Theoretical Aspects of Software Engineering*, Beijing, China, 2012, pp.267–270.

[61] G. Z. Lu and H. K. Miao, Feasibility Analysis of the EFSM Transition Path Combining Slicing with Theorem Proving, *in Proc.7th International Symposium on Theoretical Aspects of Software Engineering*, Birmingham, UK, 2013, pp.153–156.

[62] R. S. Boyer, B. Elspas, and K. N. Levitt, SELECT-a Formal System for Testing and Debugging Programs by Symbolic Execution, *in Proc. the international conference on Reliable software*, Los Angeles, US, 1975, pp.234-245.

[63] L. A. Clarke, A System to Generate Test Data and Symbolically Execute Programs, *IEEE Transactions on Software Engineering* 2(3) (1976) 215–222.

[64] J. C. King, Symbolic Execution and Program Testing, Communications of the ACM 19(7) (1976) 385–394.

[65] P. Godefroid, N. Klarlund, K. Sen, DART: Directed Automated Random Testing, *in Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, Chicago, US, 2005, pp.213–223.

[66] S. T. Chanson and J. Zhu, Automatic Protocol Test Suite Derivation, *in Proc. 13th IEEE Networking for Global Communications(INFOCOM '94)*, Toronto, Canada, 1994, pp.792–799.

[67] X. Chen and J. Zhang, EFSM generation for C programswith Functions, *in Proc. 6th International Conference for Young Computer Scientist*, Hangzhou, China, 2001, pp.90–94.

[68] J. Zhang, C. Xu, and X. Wang, Path-Oriented Test Data Generation Using Symbolic Execution and Constraint Solving Techniques, *in Proc. 2nd International Conference on Software Engineering and Formal Methods*, Beijing China, 2004, pp.242-250.

[69] P. McMinn, Search-based Software Test Data Generation: A Survey, *Software Testing, Verification and Reliability* 14(2) (2004) 105–156.

[70] Y. Zhang, L. Qian, and Y. Wang, Automatic Testing Data Generation in the Testing Based on EFSM(in chinese), *Chinese Journal of Computers* 26(10) (2003) 1295–1303.

[71] R. Lefticaru and F. Ipate, Automatic State-Based Test Generation Using Genetic Al-

gorithms, *in Proc. 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, 2007, pp.188–195.

[72] 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* 53(12)(2011) 1297–1318

[73] J. Wegener, A. Baresel, and H. Sthamer, Evolutionary Test Environment for Automatic Structural Testing, *Information and Software Technology* 43(14) (2001) 841–854.

[74] R. Yang, Z. Y. Chen, Z. Y. Zhang, Z. C. Liu, and B. W. XU, A New Approach of Automated Test Case Generation on Extended Finite State Machine, *Science China-F* 44(5) (2014) 588–609.

[75] J. Zhang, R. Yang, Z. Y. Chen, Z. H. Zhao, and B. W. Xu, Automated EFSM-based Test Case Generation with Scatter Search, *in Proc. International Conference on Software Engineering Workshop on Automated Software Test*, Zurich, Switzerland, 2012, pp.76–82.

[76] F. Glover, A Template For Scatter Search And Path Relinking, *in Proc. Selected Papers from the Third European Conference on Artificial Evolution*, London, UK, 1998, pp.3–54.

[77] W. Miller and D. L. Spooner, Automatic Generation of Floating-Point Test Data, *IEEE Transactions on Software Engineering* SE-2(3) (1976) 223-226.

[78] A. J. Nebro. F. Luna, E.Alba, B. Dorronsoro, J. J. Durillo, and A. Beham, AbYSS: Adapting Scatter Search to Multi-objective Optimization, *IEEE Transactions on Evolutionary Computation* 12(4) (2008) 439–457.

[79] R. Zhao, M. Harman, and Z. Li, Empirical Study on the Efficiency of Search Based Test Generation for EFSM Models, *in Proc. 3rd International Conference on Software Testing, Verification, and Validation Workshops*, Paris, France, 2010, pp.222–231.

[80] K. Lakhotia, P. McMinn, and M. Harman, An Empirical Investigation into Branch Coverage for C Programs Using CUTE and AUSTIN, *Journal of Systems and Software* 83(12) (2010) 2379–2391.

[81] M. Souza, M. Borges, M. Amorim, and C. S. Pasareanu, CORAL: Solving Complex Constraints for Symbolic Pathfinder, *in Proc. 3rd International Symposium of NASA Formal Methods*, LNCS 6617 (2011,) 359–374.

[82] M. Harman, Y. Jia, and B. Langdon, Strong Higher Order Mutation-based Test Data Generation, *in Proc. 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Szeged, Hungary, 2011, pp.212–222.

[83] A. Memon, I. Banerjee, and A. Nagarajan, What Test Oracle Should I Use For Effective GUI Testing, *in Proc. 18th IEEE International Conference on Automated Software Engineering*, Montreal, Canada, 2003, pp.164–173.

[84] D. J. Richardson, S. L. Aha, O.O'Malley, Specification-based Test for Reactive Systems, *in Proc. 14th international conference on Software Engineering*, Melbourne, Australia, 1992, pp.105–118.

[85] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, Testing Selection Based on Finite State Models, *IEEE Transactions on Software Engineering* 17(6) (1991) 591–603.

[86] D. Lee, D. Chen, R. Hao, R. E. Miller, J. Wu, and X. Yin, A Formal Approach for Passive Testing of Protocol Data Portions, *in Proc. 10th IEEE International Conference on Network Protocols*, Paris, France 2002, pp.122–131.

[87] R. M. Hierons, Verdict Functions in Testing With a Fault Domain or Test Hypotheses, *ACM Transactions on Software Engineering and Methodology* 18(4) (2009) 1–19.

[88] A. Cavalli, C. Gervy, and S. Prokopenko, New approaches for Passive Testing Using An Extended Finite State Machine Specification, *Information and Software Technology* 45(12) (2003) 837–852.

[89] H. Ural and Z. Xu, An EFSM-Based Passive Fault Detection, *Testing of Software and Communicating Systems* 4581 (2007) 335–350.

[90] B. Korel, G. Koutsogiannakis, and L. Tahat, Model-based Test Prioritization Heuristic Methods and Their Evaluation, *in Proc. 3rd international workshop on Advances in model-based testing*, London, UK, 2007, pp.34–43.

[91] L. Mariani, S. Papagiannakis, and M. Pezze, Compatibility and Regression Testing of COTS-Component-Based Software, *in Proc. 29th International Conference on Software Engineering*, Minneapolis, US, 2007, pp.85–95.

[92] B. Guo, M. Subramaniam, and H. F. Guo, An Approach to Regression Test Selection of Adaptive EFSM Tests, *in Proc. 15th International Symposium on Theoretical Aspects of Software Engineering*, Xi'an, China, 2011, pp.217–220.

[93] M. Subramaniam, B. Guo, and Z. Pap, Using Change Impact Analysis to Select Tests for Extended Finite State Machines Software Engineering and Formal Methods, *in Proc. 7th IEEE International Conference on Software Engineering and Formal Methods*, Hanoi, Vietnam, 2009, pp.93–102.

[94] S. Selvakumar, M. R. C. Dinesh, and C. Dhineshkumar, Extended Finite State Machine Model-Based Regression Test Suite Reduction Using Dynamic Interaction Patterns, *in Proc. International Conference on Recent Trends in Business Administration and Information Processing*, Thiruvananthapuram, India, 2010, pp.475–481.

[95] N. Ngamsaowaros and P. Sophatsathit, A Novel Framework for Test Domain Reduction using Extended Finite State Machine, *in Proc. International Conference on Software Engineering Advances*, Cap Esterel, France, 2007, 25–31.

[96] H. Ural and H. Yenigun, On Capturing Effects of Modifications as Data Dependencies, *in Proc. 36th IEEE International Conference on Computer Software and Applications Conference*, Swissotel Grand EfesIzmir, Turkey, 2012, pp.350–351.

[97] K. Androutsopoulos, N. Gold, M. Harman, Z. Li, and L. Tratt, A Theoretical and Empirical Study of EFSM Dependence, *in Proc. IEEE International Conference on Software Maintenance*, Edmonton, Canada, 2009, pp.287–296.

[98] K. Androutsopoulos, D. Clark, M. Harman, R. M. Hierons, Z. Li, and L. Tratt, Amorphous Slicing of Extended Finite State Machines. *IEEE Transaction on Software Engineering* 39(7) (2013) 892–909.