Feasibility Analysis of the EFSM Transition Path Combining Slicing with Theorem Proving*

Gongzheng Lu[1,2] Huaikou Miao[1,3]
[1]School of Computer Engineering and Science,Shanghai University,Shanghai, China.
[2]School of Computer Engineering,Suzhou Vocational University,Suzhou, China.
[3]Shanghai Key Laboratory of Computer Software Testing & Evaluating,Shanghai, China.

Abstract—**It is an important problem to generate test data from EFSM model in model-based testing, but it is time-wasting to generate test data for the infeasible paths, so determining the feasibility of the paths before generating test data for them is necessary. The feasibility of EFSM transition paths is analyzed by combing slicing with theorem proving. It is divided into two phases. In the first phase, the transitions related to the predicate on each transition in the path are got by backward slicing. And in the second phase, the feasibility of the path is determined by theorem proving to prove whether the post-condition of the transitions related to the predicate implying the predicate or not. Experimental result shows that the feasibility of the paths can be decided effectively and the number of theorem proving is reduced greatly, the infeasibility of the path also can be checked quicker by the proposed method.**

Keywords—***EFSM; Feasibility Analysis; Slicing; Theorem Proving***

I. INTRODUCTION

Software testing is an important and traditional software quality assurance technology. Now, test cases are generated by manually which is low efficient, error-prone, non renewable and increases the total cost of software development. So automated testing is a necessary trend, and model-based testing is a main research direction in this area.

In model-based testing, it gets transition paths(transition sequences) from system model firstly, then generates test data (input) which can trigger these transition paths, and gets results (expected output) by executing the generated test data along the transition paths. The system model is usually specified by Finite State Machine (FSM) or Extended Finite State Machine (EFSM). FSM consists of states and transitions. It can only specify the control part of the system model。EFSM extends the FSM with variables. A transition of the EFSM includes predicate (guard condition/precondition) and operation (assignment/postcondition). So it not only can specify the control part but also the data part of the system model. When generating test data from the EFSM model whose transitions including predicates and operations, the transition path may be infeasible which means that there doesn't exist test data can trigger such path. We try to generate test data for the path is time-consuming. So the feasibility of the path should be considered firstly before generating test data for it.

In this paper, the feasibility of EFSM transition paths is analyzed by combing slicing with theorem proving. It is divided into two phases. In the first phase, the transitions related to the predicate on each transition in the path are got by backward slicing. And in the second phase, the feasibility of the path is determined by theorem proving to prove whether the post-condition of the transitions related to the predicate implying the predicate or not. Experimental result shows that the feasibility of the paths can be decided effectively and the number of theorem proving is reduced greatly, the infeasibility of the path also can be checked quicker by the proposed method.

The organization of this paper is as follows: the related works are introduced in section II, the basic concepts used in this paper are in section III, section IV gives the slicing algorithm of the EFSM path, section V describes the feasibility analysis of the EFSM transition paths using our proposed method, section VI shows the experimental results, and the conclusions are given at last.

II. RELATED WORKS

Many works have been done for the feasibility analysis of the path in program in the past. R. Jasper *et al*[1] analyzed the feasibility of the path in Ada program using a theorem prover KITP. The program path was expressed as a first order logic formula, and the feasibility of the path was determined by the theorem prover to prove whether the negation of the formula was a theorem or not. If the negation of the formula was proved to be a theorem, then the path was infeasible, else was feasible. A. Goldberg *et al*[2] also analyzed the feasibility of the path in Ada program using KITP, but the path was represented by a regular expression which was get by symbolic evaluation, backward substitution and simplification techniques, the path feasibility was determined by the theorem prover to check whether the expression was satisfiable or not. J. Zhang *et al*[3] decided the feasibility of the path in C program using backward substitution and constraint solving. Their approach obtained path condition through backward substitution first, and then the path feasibility was determined by a constraint solver to check the satisfiability of the path condition. If the condition was satisfiable, then the path was feasible, else was infeasible. D.F. Yates *et al*[4] generated paths in terms of the assertion which shows that the less predicates the path has the higher probability it's feasible, and they validated the relation between predicate number and path feasibility by experiments. I. Forgács *et al*[5] got feasible program path using principal slicing. This method selected the path which can reach to specified program point and has the approximately minimum affected predicates. They believed that such path was most likely feasible. M.N. Ngo *et al*[6] analyzed the feasibility of program path based on heuristic method. They observed the common properties existing among the infeasible paths, and judged the path feasibility by checking whether the path has the property or not. Recent years, with the model-based testing has been research hotspot, many papers about feasibility of transition path in EFSM model have appeared. A. S. Kalaji *et al* [7] analyzed path feasibility in EFSM model using genetic algorithm, and constructed a feasible metric which was a part of the fitness function of the genetic algorithm. The values of the feasible metric were the penalty values which computed in terms of the types of the predicates and operations using dataflow analysis. But this feasible metric was just an experimental metric. R. Yang *et al*[8] decided the path feasibility combining static with dynamic method. They first analyzed the infeasible paths using the static method similar in [7], and then analyzed dynamically the infeasible paths during test data generation. The number and depth of the searching procedure were limited. The paths that can't determine their feasibility until the maximum number or depth of the searching were considered as infeasible, this may cause incorrect results. J. Cheng *et al*[9] divided variables on the transition path into counter variables, selector variables and paradoxical variables, and considered that the path infeasibility was mainly caused by these variables. But they didn't give the method identifying the transitions related to these variables. T.Y. Wu *et al*[10] expanded the path using evaluation metric, and checked whether this path was linearly independent of the existed paths to determine the path feasibility. It required that the constraints and expressions of the path must be linear.

III. BASIC CONCEPTS

A. Extended Finite State Machine(EFSM)

Extended Finite State Machine is a six-tuple: (S, s_0, V, I, O, T), where S is a finite set of states, s_0 is an initial state, V is a set of context variables, I is a finite set of input messages, O is a finite set of output messages, and T is a finite set of transitions. A transition $t \in T$ is a quintuple: (s_s, i, g, op, s_e), where s_s is the source state of the transition t, $i \in I$ is an input which may related with input parameter, g is a logic expression called guard condition, op is an operation consisted of assignment statements or output statements, s_e is the target state of the transition t. When the state is s_s, the accepted input is i and the guard condition g is satisfied, then the transition t is triggered, the operation op is executed and the state is transferred into state s_e at the same time. The guard condition g and the operation op both can contain input parameters and context variables. Here we consider only the deterministic EFSMs.

B. Slicing

The main reason for which causes the infeasibility of the transition path in EFSM model is the dependency relation among the variables on the transitions. We can get the set of transitions affected the variables using on a transition using slicing.

Slicing is an important reduction method which involves dependency analysis. It can be used to compute the statements that affect the value of the variable in a specified program point. For the variable in the specified program point, the execution of program slicing and the original program have the same result. There are different slicing criteria involving static slicing criterion and dynamic slicing criterion. The static slicing criterion is a tuple:(n,v), where n is the program point, v is the variable needing to be sliced. The static slicing criterion computes the statements which affect the variable v in the statement n without executing the program. And the dynamic slicing criterion is a triples:(x,n^k,v), where x is the program input, n^k is the kth execution of the statement n, and v is the slicing variable. The dynamic slicing criterion requires to provide input and to execute the program. There are two types of slicing method: forward analysis and backward analysis. Forward analysis, given the input, calculates the statements affected by the variable specified in the slicing criterion starting from the specified program point from front to back. Backward analysis computes the statements affecting the variable specified in the slicing criterion from back to front. It needs to record the execution trace of the program according to the dependency diagram.

C. Theorem proving

Automated theorem proving is a method proving the mathematic theorem automatically. In the automated theorem proving, the system behavior and the expected property are specified as logical formula, written as φ and ψ. So in order to check whether the system behavior is consistent with the property or not, we only need to check whether the formula $\varphi \Rightarrow \psi$ is valid or not. Where formula $\varphi \Rightarrow \psi$ means that each system behavior is the behavior of the property.

Theorem proving is divided into two ways: forward proof and backward proof. Forward proof applies inference rules to the known conditions, axioms and proven theorems to generate new theorems until the formula $\varphi \Rightarrow \psi$ holds or derives contraction. The more commonly used way is backward proof which also called Goal-Directed Proof. The proving procedure is: one rule $rule_1$ (or tactics tac_1) given by user is applied to the initial goal g, then the prover executes one step of inference, and generates n sub-goals g_i ($1 \le i \le n$), after this step it waits user to give a rule or strategy of the next step. And the user specifies another rule $rule_2$ (or tactics tac_2) again applying to current sub-goal according to the current state. The prover continues to execute the inference until all of the sub-goals are proven. Now many theorem provers have appeared, such as Isabelle, PVS, Z/EVES etc.

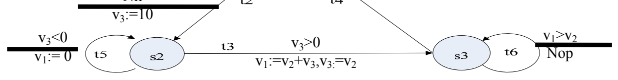
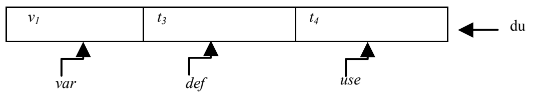


Fig. 1. EFSM *M*

IV. SLICING OF THE EFSM MODEL

Before, slicing was widely used to slice the program. There were not many researches on slicing of the model especially EFSM model. Paper [11] proposed a slicing algorithm for EFSM model, introduced the data dependency and control dependency similar in program slicing into EFSM, and computed slice according to the dependency graph. The dependency graph was constructed from dependency relations among the variables including on the transitions. Here, we only slice the generated transition path, so only the data dependency between the variables is considered.

In order to slice the given path, we need to analyze the data dependency relation in the path, and get all of the definition-use (du) pairs. The du pair is represented as a triples: (*var*, *def*, *use*), where *var* is the variable to which the du pair relates, *def* represents the transition defining the variable, *use* represents the transition using the variable. For example, in figure 1, (t_3, t_4) is a du pair for variable v_1. The triples is represented as follows:



We compute the slice of the path in EFSM using static slicing criterion and backward analysis. The slicing algorithm for the variable v in transition t_j is given in the following:

```
Algorithm 1 slice($t_j$,$v$)
1:Input: Slicing criterion ($t_j$,$v$) and the set of du pairs $DU$
2:Ouput: the slice for $v$
3:Initial $slice$=Ø;
4:while $t_j$ is the initial transition do
5:begin
6:    Let transition $t_i$ be the pre-transition of $t_j$;
7:    if ($t_i$, $t_j$) is the du pair $du_k$ belonging to $DU$ and $du_k.var$=$v$
8:       begin
9:          $slice= t_i \cup slice$;
10:        for all the du pairs $du$ satisftying $du.use= t_i$ and $du.var \in t_i$.rop($v$) do
11:            $slice=slice \cup$ slice($t_i$,$du.var$);
12:       end
13:end
```

In algorithm 1, t_i.rop(v) represents the variables appear in the right side of the assignment expression where v is defined, and v is a variable defined in the transition t_i.

Given a transition t of the transition path, the slice of transition t is formed by combining the set of slices which is get from applying the algorithm 1 to all of the variables used in t. The slice of a path is consisted of the slice of each transition t in the path.

V. FEASIBILITY ANALYSIS OF THE PATHS IN EFSM MODEL

We analyze the feasibility of the paths in EFSM model using theorem proving which check the compatibility between the transitions in the path. Given two transitions t_i and t_j, and t_j is the post-transition of t_i, if post(t_i)$\Rightarrow$pre(t_j), then we say that t_i is compatible with t_j, else they are incompatible. Where post(t_i) represents the post-condition of t_i, pre(t_j) indicates the pre-condition of t_j. We use theorem prover to prove whether the formula post(t_i)$\Rightarrow$pre(t_j) holds or not, namely whether the post-condition of t_i implies the pre-condition of t_j or not. If the formula is proved to be valid, then t_i is compatible with t_j, else t_i is incompatible with t_j.

In EFSM model, the transition is triggered when the corresponding input is accepted and the predicate on the transition is satisfied, and the operation on the transition is executed after the transition is triggered. We ignore the input and output because we don't need to use them in proving. The pre-condition of transition t is ng_t, where ng_t is get by substituting the latest definition of the variables for the variables in the predicate on the transition t. The post-condition of transition t is ($ng_t \wedge nop_t$), where nop_t is get by substituting the latest definition of the variables for the variables in the operation which is executed after the transition is triggered.

A transition path is feasible iff any two adjacent transitions are compatible.

We consider transition path $t_1t_2t_3t_4$. In order to analyze the feasibility of the path, we must check the compatibility of the transition pairs (t_1,t_2), (t_2,t_3) and (t_3, t_4). Firstly the compatibility of (t_1,t_2) is checked, t_1.g is $p_1 \geq 10 \wedge p_1 \leq 20 \wedge p_2 \geq 0 \wedge p_2 \leq 10$, $t_1.op$ is $v_1=p_1 \wedge v_2=p_2$, post(t_1) is $v_1=p_1 \wedge v_2=p_2 \wedge p_1 \geq 10 \wedge p_1 \leq 20 \wedge p_2 \geq 0 \wedge p_2 \leq 10$, pre($t_2$) is *True*. Because the formula post(t_1)$\Rightarrow$pre(t_2) holds, (t_1,t_2) is compatible. Furthermore, we can get (t_2,t_3) is compatible. Similarly, (t_3,t_4) is compatible. Because (t_1,t_2), (t_2,t_3) and (t_3,t_4) are all compatible, transition path $t_1t_2t_3t_4$ is feasible.

If we modify the EFSM M in figure 1 to the model M' in the figure2:

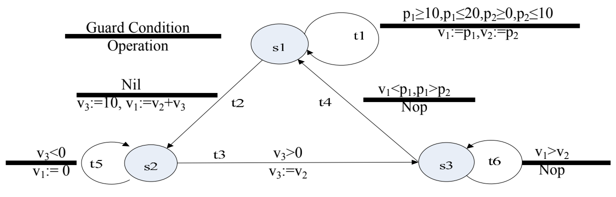


Fig. 2. EFSM M'

We still consider the feasibility of path $t_1t_2t_3t_4$. When we check the compatible of (t_3,t_4), $t_3.g$ is $v_3>0$, $t_3.op$ is $v_3=v_2$. After substituting the latest definition for v_2 and v_3, post(t_3)=$10>0 \wedge v_3=p_2$. $t_4.g$ is $v_1> v_2$. In the replacement procedure, v_1 and v_2 are irrelevant with the transition t_3, but they are replaced by the latest definition in transition t_1 and t_2. It shows that the post-condition of the pre-transition t_3 of t_4 is not relevant with the pre-condition of t_4. Only the post-conditions of the transitions which are depended by the variables defined in the pre-condition of the transition are correlated to the pre-condition. We apply the slicing technique mentioned above to the theorem proving for checking the path feasibility.

Assuming the slice of the predicate g_i in the transition t_i is $\{t_1 \ldots t_{i-1}\}$. The transition path is feasible iff the predicate g_i on each transition t_i of front to back in the path satisfies the formula $\varphi \wedge$post(t_1)$\wedge \ldots \wedge$post(t_{i-1})$\Rightarrow g_i$. Where φ indicates the range of the input parameter, post(t_j)($1 \leq j \leq i$-1) represents the post-condition of t_j. Here, the predicate of t_j is not considered in post(t_j), because the predicates in the path are checked sequentially, and when $\varphi \wedge$post(t_1)$\wedge \ldots \wedge$post(t_{i-1})$\Rightarrow g_i$ is checked, all the predicates before g_i have held.

We consider the feasibility of the transition path $t_1t_2t_3t_4$ in figure 2. For the predicate $v_3>0$ on transition t_3, we get slice $(t_3, v_3)=\{t_2\}$. $\varphi\wedge$post (t_2) is $p_1\geq10\wedge p_1\leq20\wedge p_2\geq0\wedge p_2\leq10\wedge v_3=10\wedge v_1=v_2+v_3$, and $\varphi\wedge$post$(t_2)\Rightarrow v_3>0$ holds. We continue to consider the predicate $v_1>v_2$ on transition t_4. We get slice$(t_4, v_1)\cup$slice$(t_4,v_2)=\{t_1,t_2\}$. $\varphi\wedge$post$(t_1)\wedge$post(t_2) is $p_1\geq10\wedge p_1\leq20\wedge p_2\geq0\wedge p_2\leq10\wedge v_1=p_1\wedge v_2=p_2\wedge v_3=10\wedge v_1=v_2+v_3$, and $\varphi\wedge$post$(t_1)\wedge$ post$(t_2)\Rightarrow v_1> v_2$ holds, so the path $t_1t_2t_3t_4$ is feasible.

After the slicing technique is applied to theorem proving, we don't need to check the compatibility between each two adjacent transitions, and also reduce the number of theorem proving.

VI. EXPERIMENTAL RESULTS

Our experiment uses five EFSMs: lift EFSM, In-Flight safety EFSM, ATM EFSM, Class II transport protocol EFSM and Inres initiator EFSM. Their detailed description is shown in literature [7].

The transition path is generated according to the transition coverage criterion. A test target may be covered by multi-paths, the different path may affect our result, so we generate three different set of paths for each EFSM model using random method. First, the path feasibility is analyzed using theorem proving without slicing. Afterwards, the path feasibility is analyzed using theorem proving with slicing, the slice of the path is computed by the slicing algorithm in section IV.

Table I and II indicate the results of our experiment.

TABLE I. THE STATISTIC RESULT OF FIVE EFSM MODELS

EFSM	Total number of path	feasible paths	infeasible paths	theorem proving (without slicing)	theorem proving (with slicing)	reduction	Reduction ratio
Lift	75	33	42	165	165	0	0%
In Flight safety	96	69	27	268	185	83	30.97%
ATM	90	85	5	401	197	204	50.87%
Class II protocol	63	60	3	145	89	56	38.62%
Inres initiator	48	32	16	179	115	64	35.75%
Total	372	279	93	1158	751	407	35.15%

TABLE II. THE STATISTIC RESULTS OF INFEASIBLE PATHS OF FIVE EFSM MODELS

EFSM	infeasible paths	theorem proving (without slicing)	theorem proving (with slicing)	reductions	Reduction ratio
Lift	42	68	68	0	0%
In Flight safety	27	72	52	20	27.78%
ATM	5	11	11	0	0%
Class II protocol	3	7	7	0	0%
Inres initiator	16	66	36	30	45.45%
Total	93	224	174	50	22.32%

I. CONSLUSIONS

We propose a method for analyzing the feasibility of the transition path in EFSM model in this paper. The proposed method combines slicing and theorem proving. It divides the path feasible analysis into two phases. In the first phase, the transitions related to the predicate on each transition in the path are got by backward slicing. And in the second phase, the feasibility of the path is determined by theorem proving to prove whether the post-condition of the transitions related to the predicate implying the predicate or not. Experimental results show that the feasibility of the paths can be decided effectively and the number of theorem proving is reduced greatly, the infeasibility of the path also can be checked quicker by our method.

It is worth noting that the experimental results are preliminary. The transition path is generated by random method, so there are many infeasible paths in the models. We will integrate our method into the path generation process, and consider the coverage ratio of the path during the generation process.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (NSFC) under grant No.60970007, No.61073050 and No.61170044; and the Shanghai Leading Academic Discipline Project of China under grant No.J50103.

REFERENCES

[1] R. Jasper, M. Brennan, and K. Williamson, et al. Test Data Generation and Feasible Path Analysis[C]. In Proc. of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis(ISSTA94), 1994,pp95-107.

[2] A. Goldberg, T.C. Wang, and D. Zimmerman. Applications of Feasible Path Analysis to Program Testing[C]. In Proc. of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA94), 1994, pp80-94.

[3] J. Zhang, X.X. Wang. A CONSTRAINT SOLVER AND ITS APPLICATION TO PATH FEASIBILITY ANALYSIS[J]. International Journal of Software Engineering & Knowledge Engineering, 2001, 11(2): pp139-156.

[4] D.F. Yate, N. Malevris. Reducing The Effects Of Infeasible Paths In Branch Testing[J]. ACM SIGSOFT Software Engineering Notes, 1989, 14(8): pp48-54.

[5] I. Forgács, A. Bertolino. Feasible Test Path Selection by Principal Slicing[C]. In Proc. of the 6th European Software Engineering Conference held Jointly with the 5th ACM SIGSOFT Symposium on the Foundations Software Engineering (ESEC/FSM 97). 1997: pp378-394.

[6] M.N. Ngo, H.B.K. Tan. Heuristics-based infeasible path detection for dynamic test data generation[J]. Information and Software Technology, 2008, 50: pp641-655.

[7] A.S. Kalaji, R.M. Hierons, S. Swift. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models[J]. Information and Software Technology, 2011, 53: pp1297-1318.

[8] R. Yang, Z.Y. Chen, and B.W. Xu, et al. Improve the Effectiveness of Test Case Generation on EFSM via Automatic Path Feasibility Analysis[C]. In Proc. of the 2011 IEEE 13th International Symposium on High-Assurance Systems Engineering (HASE 2011). IEEE Computer Society, 2011: pp17-24.

[9] J. Cheng, L. Zheng, R.L. Zhao. Infeasible Path Detection for EFSM Models[J]. Journal of Inner Mongolia University (Natural Science Edition), 2011, 42(5): pp498-504.

[10] T.Y. Wu, J. Yan, and J. Zhang. A Path-oriented Approach to Generating Executable Test Sequences for Extended Finite State Machines[C]. In Proc. of the 2012 IEEE 6th International Symposium on Theoretical Aspects of Software Engineering (TASE 2012). IEEE computer society, 2012: pp267-270.

[11] K. Androutsopoulos, N. Gold and M. Harman, et al. A Theoretical and Empirical Study of EFSM Dependence[C]. In Proc. of the 2009 IEEE International Conference on Software Maintenance (ICSM 2009), 2009: pp287-296.