uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yanping Chen
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Computer Science)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Specification-based Regression Test Suite Generation and Reduction
TITRE DE LA THÈSE / TITLE OF THESIS

Hasan Ural
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Guy-Vincent Jourdan

Bogdan Korel (Illionois Institute of Technology

Yvan Labiche

Robert Probert

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Specification-Based Regression Test Suite Generation and Reduction

by

Yanping Chen

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies for partial fulfillment of the requirement for the degree of

Ph.D. in Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa

Ottawa, Ontario, Canada

December 2009

@ 2009, Yanping Chen, Ottawa, Canada

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-61375-7
Our file Notre référence
ISBN: 978-0-494-61375-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

During maintenance of evolving software systems, regression testing is crucial for confirming that the unchanged parts of the system have not been adversely affected by the modifications on the system specification and implementation. It is time- and resource-consuming, especially for large software systems.

Much attention has been paid to the regression testing area in recent years. Most regression testing techniques are code-based. There exists limited research on requirement-based regression testing techniques. In our research, we are interested in model-based regression testing techniques. Model-based testing involves, among other activities, test generation, execution, and evaluation using software models [Binder 00]. Some such models are given in formal description languages, e.g., Extended Finite State Machine (EFSM) models.

In this thesis, we present two regression testing techniques, a regression test suite (RTS) generation method (RTSG method), and a regression test suite reduction method (RTSR method). Both methods are based on EFSM *dependence analysis*. The RTSG method generates regression test cases to construct a regression test suite instead of selecting test cases from an existing test suite. The RTSR method is an extension to an existing requirement-based regression test suite reduction approach presented in [Korel+02] and [Xie 05].

In this thesis, based on [Korel+02] and [Xie 05], twelve types of dependencies are identified related to three types of elementary modifications (EMs), i.e., adding, deleting, and changing transitions in an EFSM model representing an SUT. These dependencies capture the effects of the model on the EMs, the effects of the EMs on the model, and the side-effects of the EMs. The proposed RTSG method constructs an RTS by covering all occurrences of these dependencies caused in a given EFSM model by a given set of EMs.

Given an EFSM representing the requirements of a system under test (SUT) and a set of EMs on the EFSM, interaction patterns are identified related to each type of EMs, i.e., adding, deleting, and changing transitions in the EFSM. These interaction patterns capture the effects of the model on the EMs, the effects of the EMs on the model, and the

side-effects of the EMs. The proposed RTSR method reduces the size of a given RTS by examining interaction patterns covered by each test case in the given RTS.

In this thesis, we also propose algorithms to obtain these dependencies; to generate potential interactions with respect to an EM; to generate a regression test suite; and to reduce a given regression test suite.

To show the effectiveness of the proposed RTSG and RTSR methods, we conducted case studies on seven EFSM models including an EFSM model representing an IBM product feature. Three studies were done on each EFSM model.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor Dr. Hasan Ural for his invaluable guidance, continuous support, and frequent encouragement, especially for giving me feedback at amazing speed.

I would like to thank everyone from the Advanced Software Engineering Research and Training Group (ASERT), especially Dr. Robert L. Probert, and Tuong Nguyen, for their useful suggestions and help. Moreover, I would like to thank Dr. Tahat for providing experimental materials.

I would also like to thank my husband, my son, and my parents for their love, support, and encouragement during the whole period that went into pursuing my graduate studies.

Finally, I would like to acknowledge financial support from IBM Canada, the Ontario Centres of Excellence, the Natural Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarships Program, the Ontario Graduate Scholarship Program (OGS), and the University of Ottawa.

List of Abbreviations and Terms

adaptive maintenance: Section 2.4, page 27

Activation CD = activation control dependence: Section 3.1.3, page 52; Section 3.2.3, page 61

Activation DD = activation data dependence: Section 3.1.3, page 51; Section 3.3.3, page 69

Activation GCD = activation ghost control dependence: Section 3.1.3, page 53; Section 3.2.3, page 63

Activation GDD = activation ghost data dependence: Section 3.2.3, page 62, Section 3.3.3, page 69

Affecting CD = affecting control dependence: Section 3.1.1, page 49

Affecting DD = affecting data dependence: Section 3.1.1, page 49; Section 3.3.1, page 67

Affecting GCD = affecting ghost control dependence: Section 3.2.1, page 57

Affecting GDD = affecting ghost data dependence: Section 3.2.1, page 57; Section 3.3.1, page 67

Affected CD = affected control dependence: Section 3.1.2, page 49

Affected DD = affected data dependence: Section 3.1.2, page 49; Section 3.3.2, page 68

Affected GCD = affected ghost control dependence: Section 3.2.2, page 59

Affected GDD = affected ghost data dependence: Section 3.2.2, page 59; Section 3.3.2, page 68

AP = activating path: Section 4.1.1, page 75

applicable: Section 4.1.1, page 77

CD = control dependence: Section 2.7.2, page 32

CFG = control flow graph: Section 2.3, page 21

complete test case: Section 4.1, page 74

contributing modification: Section 3.1.3, page 53

corrective maintenance: Section 2.4, page 27

corrective regression testing: Section 2.4, page 27

CRTA = controlled regression testing assumption: Section 2.5, page 28

DD = data dependence: Section 2.7.1, page 31

def = definition of v: Section 2.7.1, page 31

definition-clear path: Section 2.7.1, page 31

delta: Chapter 2, page 9

du-pair = definition–use pair: Section 2.7.1, page 31

EFG = event-flow graph: Section 2.2.2, page 19

EFSM = Extended Finite State Machine: The abbreviation is defined in Section 1.2, page 7. The term is defined in Section 2.6, page 28.

EM = elementary modification: Section 1.2, page 8

error-prone state: Section 6.2.4, page 114

error-prone transition: Section 6.2.4, page 114

function: Section 2.8.1, page 34

FVT = Function Verify Test: Section 6.2, page 104

increment: Section 1.1.2, page 2

IP = interaction pattern: the abbreviation is defined in Section 1.2, page 8. The term is defined in Section 5.1, page 87

IT = integration tree: Section 2.2.2, page 19

major change: Section 3.3, page 65

minor change: Section 3.3, page 65

model executability: Section 2.6, page 29

MOLS = Mutually Orthogonal Latin Squares: Section 2.2.2, page 18

MSC = Message Sequence Chart: Section 1.2, page 7

NDPM = new dependence per modification: Section 3.1.1, page 49

path: Section 2.7.1, page 31

pattern: Section 5.1, page 87

perfective maintenance: Section 2.4, page 27

post-dominate: Section 2.7.2, page 32

preamble: Section 4.1.2, page 77

postamble: Section 4.1.2, page 77

progressive maintenance: Section 2.4, page 27

progressive regression testing: Section 2.4, page 27

regression testing: Section 1.1.3, page 3

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Introduction

Regression testing is an essential part of an effective testing process for ensuring software quality. It is well known to be an important software maintenance activity performed with the aim of gaining confidence in modified software [Rothermel+00]. During software maintenance, both the specification and implementation of the software system are modified to fix defects, change functionality, or fulfill new requirements. *Regression testing* is the process of validating modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by the modifications [Harrold+01].

In this chapter, we define terms that are used in this thesis. Also, we briefly address problems in regression testing and motivation of our research. Finally, we give the organization of this thesis.

1.1.1 Software Maintenance and Regression Testing

Software maintenance is the most costly phase of the software life cycle, and has been estimated to comprise between 50 and 80 percent of the total software development cost [Leung+91]. One software regression horror story was the network failure of AT&T on September 17th, 1991, which disrupted long-distance calls and air traffic control communications in the New York metropolitan area. Three airports had to be shut down for five hours. It cost AT&T millions of dollars to recover the network service and its company's reputation. Shortly before that disaster, in January 1990, AT&T's long-distance telephone network also had a nine-hour breakdown caused by a code patch, which was not regression tested.

Most software systems evolve to respond to changed requirements based on changing environments, changing needs, new concepts and new technologies. Software usually grows in the number of functions, components and interfaces. Existing systems may be expanded for uses beyond the scope of their original design. Thus, modification of software is inevitable [Leung+91].

Regression testing is one of the necessary maintenance tasks. According to [Harrold+01], it is used to help detect any adverse impact of modifications. This adverse impact of a change is often called the "ripple effect", and is known to be a serious cause of software regression (worsening) as the result of a change [Probert 05]. Fortunately, it is widely accepted that efficient and effective regression testing can reduce the frequency and cost of software maintenance.

1.1.2 Software Development Process and Regression Testing

With widespread usage of object-oriented programming techniques, more and more projects follow an evolutionary process model, also called an incremental model [McGregor+01].

Under this process model, a system is developed as a sequence of increments. An increment is a deliverable that provides some of the functionality required for the system, including models, documentation and code. For each increment, the development process feeds new and/ or revised designs and implementations into the testing process. The testing process feeds identified failures back into the development process. The development process and the testing process form a continual feedback loop as shown in Figure 1-1 [Chen 02].

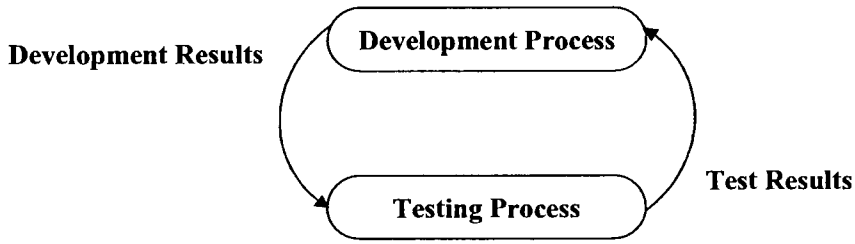


Figure 1-1: The feedback loop of development and testing processes

In the incremental model, successive increments add and sometimes revise system functionality while keeping most of the functionality of previous increments. Regression testing is typically done between increments to ensure that changes do not adversely affect correctly working parts of the previous versions.

Reuse has been proposed as a general solution to chronic software development problems: namely, lengthy development time and high cost, unacceptably frequent failures, low maintainability, and low adaptability [Binder 00]. A major precondition of reuse is to ensure that the reused components match the new requirement and do not conflict with new components. An example of disasters caused by incorrect reusing of components was the explosion of Ariane 5 on June 4, 1996. The rocket was on its first voyage, after a decade of development costing $7 billion. Estimates of the total cost of the destroyed rocket and its cargo vary from a low of $350 million in a European Space Agency press release to a high of $2.5 billion reported in Florida Today Space Online. The main reason for the explosion was that a time sequence based on the requirements of Ariane 4 was reused, but this did not match the requirements of Ariane 5 [SIAM 96]. Effective regression testing should have caught this problem before the disastrous launch.

With object-oriented techniques, skillful design and hard work, classes and components can be produced that may be used in many applications. New projects are built by re-using components from legacy systems or third parties. The obvious contribution of regression testing to reusability is to assess and assure the quality of re-used components, and to gain confidence in their integration.

1.1.3 Regression Testing Techniques

In [Leung+91], Leung and White defined *regression testing* as a testing process, which is applied after a program is modified. It involves testing the modified program with some test cases in order to re-establish our confidence that the program will perform according to the (possibly) modified requirements. In this thesis, we will use the definition in [IEEE 610], which is: *regression testing is defined as selective retesting of a system, subsystem, or component to verify that modifications have not caused unintended effects and that the system still complies with its specified requirements.*

A *regression test case* is a test case that is expected to pass when run on a modified version of the system under test (SUT). A *regression test suite* (RTS) is composed of regression test cases. A *regression fault* is a fault induced by an executable configuration of the modified SUT. It is revealed by a test case that the modified version of the SUT does not pass.

In [Leung+91], the authors defined two regression test strategies: the *retest all* strategy which reuses all tests, and the *selective* strategy which uses a subset of the existing test cases in regression testing. These definitions are widely used in the literature. But in [Mayrhauser+99], different definitions are given for *retest all* and *selective regression testing* strategies. A *retest all* strategy tests the system all over again, since it assumes that changes could have introduced errors anywhere in the code. A *selective regression testing* strategy assumes that not all parts of the software could have been affected by the modifications. With the selective regression testing strategy, we retest only the parts of the system affected by modifications. Retesting an entire system that has received few changes is very expensive for large systems. By not running a modified system on test cases that test unaffected parts of the SUT, substantial effort may be saved. In this thesis, we define *retest all* and *selective regression testing* strategies as in Mayrhauser et al.

Retest all regression testing is a simple and easy strategy with the least risk of missing a regression fault. But, if, with respect to time and cost a full regression test run is prohibitive, selective regression testing strategy must be chosen.

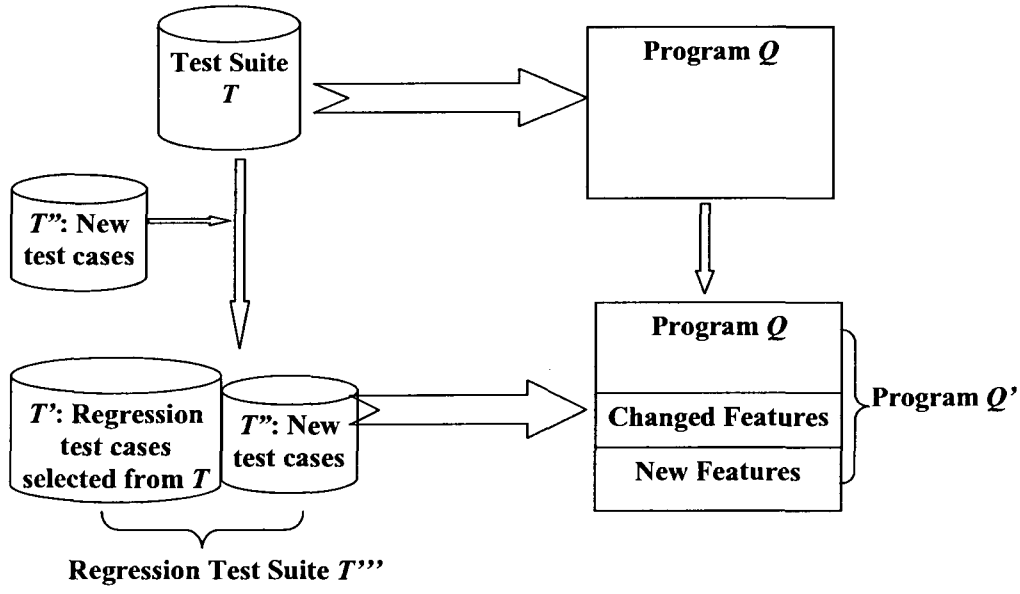


Figure 1-2. Regression testing techniques

Regression testing techniques basically consist of the following steps [Rothermel+96], which are illustrated in Figure 1-2.

1. Identify the modifications that were made to Q to yield Q'.
2. Select $T' \subseteq T$, a set of test cases to execute on Q', based on these modifications.
3. Test Q' with T', to establish the correctness of Q' with respect to T'.
4. If necessary, create T'', a set of new functional or structural test cases for Q'.
5. Test Q' with T'', to establish the correctness of Q' with respect to T'', and gather test information for T''.
6. Create T''', the regression test suite for Q', by combining T' and T''.

Thus, for either *retest all* or *selective* regression testing strategies, the tester may need to develop new tests to exercise new features of the software or to cover parts of the software that are no longer covered by existing tests.

1.2 Statement of the Research Problem and Motivation

Many researchers are investigating regression testing techniques (RTT). Their research spans a wide variety of topics. For example, Brown and Hoffman [Brown+90] worked on test environments and automation of the regression testing process. Harrold, Gupta, and Soffa [Harrold+93], and Wong et al. [Wong+95] addressed test suite management. Rothermel and Harrold [Rothermel+96] presented a framework to evaluate regression test selection techniques.

In recent years, much attention has been put into the regression testing area. Most techniques are code-based (white box) techniques, that is, they select or create tests based on information about the differences between the original code and the modified version [Binkley 97, Harrold+01, Kung+95, Rothermel+00, Xie+05]. Some techniques are *specification-based* (black box) methods, that is, they select or create tests based on information obtained from program specifications [Briand+02, Chen+02, Muccini+05, Stocks+96, Vieira+00].

We are interested in specification-based RTTs, because, in our opinion, there are several shortcomings with code-based techniques. First, code-based RTTs are not effective for detecting specification-level errors; thus, they cannot be used to validate an implementation (check if the implementation satisfies the specification). It has been

shown that specification-based techniques and code-based regression techniques complement each other in the kind of faults they detect [Juristo+06].

Secondly, code-based RTTs require testers to access and understand the code to some degree [Chen+02]. This requirement causes many practical problems. Testers have to spend time reading the code written by others and trying to understand how it works. This is very time-consuming, costly, and error-prone.

Finally, code-based RTTs are language-dependent. In some software systems, more than one programming language may be used, e.g., an Internet application may use Java, JSP and HTML. More than one code-based regression technique will then be necessary for regression testing. This makes the situation more complex.

We also notice that, during software maintenance, system models are frequently modified to reflect changes in system requirements. When a system model is changed, model based testing techniques can be applied to the modified model to partially test the SUT with respect to selected requirements. But such an RTS may be very large even for relatively small systems [Tahat+01]. In addition, model based testing techniques usually attempt to satisfy some coverage criterion for constructing a test suite [Bourhfir+97, Huang+95, Sarikaya+86, Ural+91]. Fortunately, an RTS may not need to target the same coverage as an original test suite. This is mainly for two reasons. The first reason is that, only part of the SUT is tested by the RTS to verify that modifications have not caused unintended effects in the SUT and that the SUT complies with the changes in the requirements. The second reason is that, frequent regression testing during software maintenance benefits from minimizing the size of the RTS. Thus, we are interested in a specification-based regression test suite reduction technique (RTSR method) that can efficiently reduce the size of a given RTS.

No matter which regression test selection strategy is used, testers may only select regression test cases from existing test suites, which are designed before modifications. This is not always practical since there are several problems with this kind of selection strategy. First, the modified version of the SUT may include new or changed features. Thus, specifications of the SUT have changed, and new test cases for the changed specifications are needed. Secondly, the modifications might have involved fixing a bug; but the developer may have fixed only the symptoms of the bug and not the cause of the

bug. Rerunning only the test cases that revealed the bug will not reveal such a problem. In many cases, additional new test cases are needed. Third, regression test selection techniques also need to maintain the test suite that is used to retest the program. The overhead involved in storing and updating test suites could be very high.

To provide new test cases for regression testing, and to save the overhead of maintaining and updating the test suite, another approach is to generate test cases for only the changed parts of the system during regression testing. Some researchers in this area have proposed regression test generation techniques which can be categorized into *white box* methods [Korel+98, Gupta+92] and *black box* methods [Tahat+01, Xu+04] as well.

Some recent work has focused on black box regression test generation. For instance, there are commercial tools for generating tests from a state machine specification, such as Autolink in Telelogic TAU [Telelogic]. TAU derives test sequences from the combination of an Extended Finite State Machine (EFSM) model in the Specification Description Language (SDL), and a test purpose specification in the Message Sequence Chart (MSC) notation. This technique can be used to generate comprehensive regression test suites, or RTSs. However, when one uses this technique to generate RTSs, not all MSCs available for the SUT will be used. This decreases the coverage of the functionality for testing. Furthermore, the generated test sequences are guided by MSCs, which in most cases are themselves designed manually by testers, or are drawn by the tool in "guided" state space search. In either case, the quality of generated regression test cases relies on the quality of the MSCs, while the quality of the MSCs relies on the testers' technical skills. Also, the more human intervention is involved, the less efficient the method may be. Thus, we are interested in a specification-based regression test suite generation technique (RTSG method) that can address new or modified features as well as unchanged specifications of an SUT. In addition, we need an RTS created by the RTSG method to be able to reveal improperly fixed bugs (the appearance of the bug, not the bug itself was fixed).

We need both the RTSR and the RTSG methods to be objective. Testers' technical skills should not be the greatest determinant of the effectiveness of regression testing, and the methods must be efficient. Fortunately, these two methods are well suited for automation.

The main motivating factors behind this research are:

- The absence of effective specification-based RTTs to deal with both changed (new or modified) and unchanged specifications of an SUT
- The absence of an objective requirement-based RTT
- The absence of an efficient requirement-based RTT which is scalable

This thesis presents two specification-based RTTs: the RTSG method to generate an RTS, and the RTSR method to reduce the size of a given RTS. Given a set of modifications to requirements of an SUT, we map these modifications onto elementary modifications (EMs) on the EFSM model of the SUT. Then, for each EM, we identify dependencies reflecting the effects of the EM on the EFSM model, the effects of the EFSM model on the EM, and the side-effects (indirect effects, or outcomes that are not the results originally intended. These may be foreseen or unforeseen) of the EM. Based on these dependencies, we propose the RTSG method for constructing an RTS: for each dependence introduced by EMs, we construct a test case. These dependencies then are used to generate interaction patterns (IPs) that exhibit all these dependencies related to the EMs. Based on the IPs, we propose the RTSR method to reduce a given RTS which is not generated by our approaches.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we describe background information on regression testing, the EFSM model, EFSM dependencies presented in Static Dependence Graph (SDG) form, and basic terminology used throughout this thesis. Chapter 3 discusses dependence analysis based on the EFSM model and SDG in regression testing. Based on our discussions in Chapter 3, Chapter 4 addresses our RTS generation (RTSG) method, and Chapter 5 addresses our RTS reduction (RTSR) method. In Chapter 6, we present and assess the results of our case studies on the use of the proposed approaches. Chapter 7 concludes this thesis.

2. Background: Regression Testing and the EFSM Model

Regression testing is not a simple extension of testing. Myers observed that modifying an existing program was a more error-prone process than writing a new program in terms of errors per statement written [Myers 79]. Jones reported that high-technology companies like IBM had "a bad fix (error) injection rate which ranges from less than 2% to more than 20% with a modern average in the 1990s running to about 7%" [Jones 97]. Note that this observation does not apply directly to any one company, such as IBM, but reflects rates at a number of similar companies.

Regression testing is used to confirm that fixed bugs have been truly fixed, that new bugs have not been introduced (ripple effect) in the process, and that features that were shown to correctly function are intact [Nguyen 01]. Regression testing is effective for revealing regression faults caused by the delta (set of differences between two consecutive versions) side effects, delta/baseline incompatibilities, and undesirable feature interactions between a baseline and a delta. It is also effective for revealing bugs caused by bad fixes in corrective maintenance. Hence, regression testing can contribute to corrective maintenance, adaptive maintenance, and perfective maintenance.

2.1Discussion of Regression Test Strategies

Regression testing has to address the following fundamental problems [Kung+95]:

- How can all components affected by changes in some components be identified?
- What strategy should be used to retest these affected components?
- What are the coverage criteria for retesting these components?
- How should regression test suite be formed?

To address the above issues, the following activities are essential for any regression testing strategy [Chen 02].

A pragmatic regression testing strategy must:

1. Identify the affected components and select a regression test coverage criteria.
2. Form regression test suite to test the affected components.
3. Execute the modified system on the regression test suite.

4. Gather and evaluate test results, including evaluating the behavior of the modified system and reporting the coverage of the regression test suite.
5. Revise the test plan as required for the next regression testing session.

Activities 1 and 2 are the key actions of any regression testing strategy.

In our research, we propose two RTTs, namely, the RTSG method and the RTSR method. Instead of selecting regression test cases from the original test suite, and adding new test cases to form RTSs, the RTSG method generates RTSs based on modifications directly. Therefore, the RTSG method in our research consists of the following steps (compared to Rothemel et al.'s steps presented in Section 1.1.3):

1. Identify the modifications that were made to Q to yield Q'.
2. Generate T', a set of tests to execute on Q', based on these modifications.
3. Test P' with T', to establish the correctness of Q' with respect to T'.

On the other hand, the RTSR method in our research is a test suite reduction technique. It consists of the following steps (compared to Rothermel et al.'s steps presented in Section 1.1.3):

1. Identify the modifications that were made to Q to yield Q'.
2. Select test cases from the given test suite to form T', a set of tests to execute on Q', based on these modifications.
3. Test Q' with T', to establish the correctness of Q' with respect to T'.

Each RTT has advantages and disadvantages. One technique may be better than other in a specific situation, but not in every situation. Hence, we need a way to evaluate and compare existing RTTs.

Rothermel et al. presented a widely used framework to evaluate regression selection testing techniques in [Rothermel+96]. The framework consisted of four categories that were defined as follows.

- *Inclusiveness*: the extent to which an RTT chooses tests that will cause the modified program to produce different outputs than the original program, and thereby expose faults caused by modification.
- *Precision*: the ability of an RTT to avoid choosing tests that will not cause the modified program to produce different outputs than the original program.
- *Efficiency*: the computational cost, and thus, practicality of an RTT.

- *Generality*: the ability of an RTT to handle realistic and diverse language constructs, arbitrarily complex code modifications, and realistic testing applications.

In our research, we are interested in both regression test selection techniques and regression test generation techniques. The first two categories of Rothermel's framework, namely, inclusiveness and precision, are not applicable to regression test generation techniques. Therefore, Rothermel's framework cannot be used to evaluate regression test generation techniques. Accordingly, we define a new framework that also consists of four categories to evaluate an RTT: *effectiveness*, *efficiency*, *generality*, and *scalability*.

- *Effectiveness*: the ability of an RTS constructed by an RTT to detect regression faults
- *Efficiency*: the cost of creating an RTS, including the resource and time required by the RTT. The higher the cost of an RTT, the less efficient is the RTT
- *Generality*: the ability of an RTT to function in a wide and practical range of situations. It indicates the range of applications for the RTT
- *Scalability*: the ability of an RTT to function on a large size SUT

2.2 Existing Regression Test Generation Approaches

Existing regression test generation techniques can be classified as *code-based* and *requirement/specification-based* approaches. We will give a critical review of regression test generation techniques in this section.

2.2.1 Code-based Regression Test Generation Approaches

Code-based regression test generation techniques gather information from source code of an SUT to guide regression test generation process. Surprisingly, there has not been much work on code-based regression test generation methods. Most researchers in code-based regression testing area work on regression test selection approaches. Only few of them consider generating regression test cases instead of selecting test cases from the original test suite.

In [Gupta+92], a code-based regression test approach using slicing technique was addressed. This approach is based on control flow diagram. It can be used for either

regression test generation or regression test selection. The authors tried to identify all def-use associations that were affected by modifications in regression analysis.

Given a control flow diagram for modified programs, for a modification at a program point p, the regression analysis has the following steps:

1) Use backward program slice algorithm at p for variable v to identify all statements in the program that might affect the value of v at p.
2) Use forward program slice algorithm at p for variable v to identify all statements in the program that might be affected by the value of v at p.
3) Using the results of 1) and 2), together with given test criteria, either selects or generates regression test cases.

This approach belongs to *selective regression testing* strategies. Applying our evaluation framework presented in Section 2.1, we found this approach had some shortcomings.

- *Effectiveness*: this approach only considered data-flow analysis in regression analysis. Therefore, the RTS generated may not be powerful to reveal defects in the control-flow of the program
- *Efficiency*: although [Gupta+92] didn't mention automation of this approach, in our opinion, this approach can be automated. Also, since it is not required to maintain an original test suite to generate regression test cases, there is no overhead of maintaining and updating the original test suite. Therefore, this is an efficient approach
- *Generality*: this approach is based on control-flow diagram. Therefore no matter which programming language is used to implement the SUT, algorithms of this approach is applicable.
- *Scalability*: this approach can be extended to inter-procedural regression testing, but as a code-based approach, it may not suitable for larger scale testing.

Korel et al. presented a white box regression test generation approach based on input-output analysis in [Korel+98]. The purpose of this approach was to generate test cases such that each test case revealed a fault for unchanged functions. The authors assumed the specifications of the functions were unchanged and each output parameter of a

module under test corresponded to a module function. There are three steps in Korel's approach:

1) Identify all pairs of equivalent output parameters for an original module M_{old} and its modified version M_{new}.
2) Use a pre-processor to automatically generate special code (a driver) for both modules, M_{old} and M_{new}. A Report_Error statement is included in the driver for each pair of equivalent output parameters. An example of the generated driver is as follows:

   1 Generate input $x_1, x_2, \ldots, x_k$
   2 Call M_{old} (in $x_1, x_2, \ldots, x_k$, out $y_1, y_2, \ldots, y_m$)
   3 Call M_{new} (in $t_1, t_2, \ldots, t_j$, out $z_1, z_2, \ldots, z_n$)
   4 if $y_i \neq z_j$, $1 \leq i \leq m$, $1 \leq j \leq n$, then Report_Error;
   .....
3) Use an existing automated white box test data generation method, TESTGEN, to find program inputs to execute the *Report_Error* statements in the driver (line 4 to line x in the above example). Since the specifications of the functions are unchanged, outputs of M_{new} should match outputs of M_{old}. If TESTGEN finds one program input that can execute any of the *Report_Error* statements, which means outputs of M_{new} do not match outputs of M_{old}, one test case that can reveal a fault is generated.

The target of this approach is finding inputs to reveal regression faults. The approach does not use modification information in regression test case generation. Thus, it is a *retest all regression test* strategy. This approach is evaluated as follows:

- *Effectiveness*: this approach only works for unchanged functions. It cannot be used to test new or changed functions
- *Efficiency*: the first step of this approach requires manual work, while the rest two steps are automated. Since the manual work involved is not very complicated and time-consuming, the efficiency of this approach is not hampered
- *Generality*: TESTGEN is a test data generation system for Pascal programs. Therefore, it only works for systems implemented in Pascal

- *Scalability*: step 1 and 2 of this approach can be applied to large systems. In step 3, TESTGEN searches test data on code level. Hence, as the size of SUT gets bigger, the required search space increases rapidly. Therefore, this approach may not be used for large scale systems.

2.2.2 Requirement/ Specification-based Regression Test Generation Approaches

Requirement or specification-based regression test generation techniques consider requirements or specifications of an SUT as resources to generate test cases. They do not take into consideration the implementation of the SUT.

Tahat et al. proposed a requirement-based test generation approach that had been extended to regression testing in [Tahat+01]. This extension attempts to generate regression test cases for individual changed requirements expressed in SDL.

Tahat's approach consists of four major steps:

1) Describe individual requirements in SDL fragments with requirements labelled, and then combine the SDL fragments into a complete SDL system model for the SUT.
2) Convert the SDL system model to an EFSM system model. Requirements are mapped to transitions and states in the EFSM model. Requirement labelling of the SDL transitions is preserved in the EFSM transitions.
3) Mark EFSM sub-models that represent changed requirements.
4) Use a black box test generator to generate test cases for marked EFSM sub-models.

As a *selective regression testing* strategy, this approach has the following pros and cons:

- *Effectiveness*: this approach works for changed requirements only. But only changed requirements themselves were considered. However, a changed requirement may have side-effects on other requirements. Besides, fixes to a bug may have effects on other parts of the SUT, too. Both cases should be tested in regression testing

- *Efficiency*: this approach can be fully automated. Therefore, the cost to create an *RTS* is very low, and the approach can be efficient
- *Generality*: this approach can be applied to any system which can be represented by an EFSM.
- *Scalability*: as a requirement-based regression test technique, this approach can be applied to large scale applications.

[Xu+04] proposed a specification-based test generation approach using model checking. This approach generated regression test cases by comparing two versions of the specifications of an SUT. To apply the approach, system changes have to be reflected in the software specifications as in the source code. Also, for each *hold property* (Every property is actually a testing path/ branch of the system. A *hold property* is a property that doesn't change in the modified specifications), the branch is considered to be correct, and for each non-hold property, the branch is considered to have been changed (has regression faults).

Four major steps of this approach are as follows:

1) Derive properties from the original version of the specification using a *Translator*
2) Testers choose test coverage criteria and extra properties, such as new properties introduced by the modifications.
3) The two versions of the specification are fed to a *Comparator* to produce the regression test specifications based on selected test coverage criteria in step 2).
4) Feed the results from the *Comparator*, together with the properties derived in step 1) to a *Test Generator*. The *Test Generator* will choose the shortest counterexample for each non-hold property and translating it into the corresponding test case.

Accordingly, we classify this approach as a *selective regression testing* strategy. This approach has the following advantages and disadvantages:

- *Effectiveness*: in our opinion, there are four reasons that the generated *RTS* may not be effective in detecting regression faults. First, this approach assumes that system evolution is reflected in both the source code and the specification document. But in real life, specifications may not be up-to-date. Secondly, some hold-properties should not hold in the modified version of the SUT, and some

non-hold properties could become hold properties, since the properties have changed. Third, sometime the counterexample (test case) cannot be generated because there is no such execution path as a property specified. Last, testers' technical skills affect the quality of the generated RTS

- *Efficiency*: this approach allows testers to direct the process, such as loading versions of specifications, choosing test coverage criteria, defining extra properties, and so on. This gives flexibility to the approach, but deceases the efficiency of the approach
- *Generality*: this approach supports different specification languages. Therefore, it can be used widely
- *Scalability*: as a specification-based technique, this approach can be applied to large scale systems.

Mayrhauser et al. published three papers about their work. [Mayrhauser+99] is the latest of the three papers. It addresses a specification-based test generation approach based on application domain analysis, which extracted common information about a problem domain and specified the operations of the domain. A domain model of an SUT describes how inputs could be supplied to the SUT, and contains objects, commands, script components, and constraints and rules between the components. The authors aim to identify regression sub-domains, then generate and select regression test cases based on the sub-domains. They assume that specification changes can all be reflected as changes to an element of the domain model. Under the domain model, test generation has three stages: Scripting, Command Template Generation, and Parameter Value Selection.

This approach uses an automated test generation tool named Sleuth to identify regression sub-domains for regression testing, and then generate test cases based on regression sub-domains. Depending on different regression test strategies, two kinds of sub-domains, namely, minimal regression sub-domains and extended regression sub-domains can be built. A sub-domain contains X (a set of parameters), C (a set of commands), SC (a set of script component names) and SR (a set of script rules). The major steps to identify these two kinds of sub-domains are as follows:

Minimal regression sub-domain:

1) Define set X that contains all parameters which are not unique to the deleted commands, and all parameters of the new commands.
2) Define set C that contains all commands that use parameters in X.
3) Define set *SC* that contains all script component names that contain commands in C.
4) Define set *SR* that contains all script rules that use commands in C.

Extended regression sub-domain:

1) Define set X that contains all parameters which are not unique to the deleted commands, and all parameters of the new commands.
2) Define set C that contains all commands that use parameters in X.
3) Define set *Factor* that contains all other parameters that are used by commands in C, but not in X.
4) $C = C \cup$ all commands that use parameters in *Factor*.
5) Define set *SC* that contains all script component names that contain commands in C.
6) Define set *SR* that contains all script rules that use commands in C.

As a *selective regression testing* strategy, this approach has the following advantages and disadvantages:

- *Effectiveness*: not all specification changes can be reflected as changes to a command, parameter, or rule, such as non-functional specification changes. Besides, extended regression sub-domain includes commands that share parameters with a changed command. But these commands are not necessarily affected by the changes, since the shared parameters are not changed. Also, some other affected commands may be missed, simply because they don't share parameters with a changed command. Therefore, in our opinion, this approach is not effective
- *Efficiency*: this approach is fully automated. Hence, it may be classified as efficient
- *Generality*: this approach can be applied to any system

- *Scalability*: as a requirement-based regression test generation technique, this approach can be applied to large scale applications.

[White 96] presents a GUI test generation approach based on pair-wise interaction test and Mutually Orthogonal Latin Squares (MOLS). The basic idea of this approach is generating minimum number of tests that cover pair-wise interactions. The author states that testing pair-wise interactions can detect almost all GUI defects. He assumes that transitions to the next screen depend only upon the current screen.

There are three steps in this method.

1) Form MOLS, which is a set of latin squares of the same order.
2) Generate test cases based on values generated from the MOLS
3) In case of a modification, modify the MOLS accordingly (add or delete a column), then generate test cases. Also, use "don't care" entries to replace some unchanged factors.

In this approach, test cases are generated from MOLS for interaction pairs between factor values. Since "don't care" entries are used in the MOLS, not all interaction pairs between factor values are considered in test case generation. Therefore, this approach is a *selective regression testing* strategy.

- *Effectiveness*: some generated test cases may not traverse the modified portion. The author defines the number of factors, k, such that $k - 2 \leq n - 1$, where n is the order of the latin squares used. $|F_2| = n$ (the number of selections of factor which has the second highest number of selections in all factors under test). In case of $k - 2 > n - 1$, the number of MOLS, n-1, is not sufficient to generate tests for all the factors under test, then for the rest k-(n-1) factors, random value will be generated. If the modified factor is among these k-(n-1) factors, generated regression test cases may not be effective to detect regression faults. Besides, because a complete set of MOLS do not exist for certain orders, n could not be equal to 6 or 10, or any other order value for which n-1 MOLS do not exist ($n \neq 6, 10, 14, 21, 22, \ldots$)
- *Efficiency*: this approach can be automated and thus it is efficient
- *Generality*: this approach can be used for pair-wise testing only, such as interaction testing, or configuration testing
- *Scalability*: this approach can be applied to large scale systems

[Memon+05] gives a GUI smoke-test generation approach based on event-flow graph (EFG) and integration tree (IT). The authors attempt to automate structural GUI analysis, smoke-test generation, test-oracle generation, test execution and coverage evaluation. The key assumptions of their work are: first, events within a component do not interleave with events in other components without the components being explicitly invoked or terminated, secondly, the specification of GUI doesn't change.

There are five steps in this approach:

1) *GUI* ripper obtains a formal model of the SUT's GUI by reverse engineering.
2) The tester verifies the model and makes necessary changes
3) EFG and IT are generated
4) The tester specifies the number of test cases of different length
5) A test generator and a test oracle generator generate smoke-test cases based on the results of 3) and 4).

This is a smoke-test generation approach which does not use modification information to guide test case generation. Also, this is a *retest-all regression test* strategy rather than a *selective regression testing* strategy.

- *Effectiveness*: this approach relies on testers' manual work to control the number of generated smoke-test cases, and to examine the unsuccessful test cases. Therefore, quality of the generated RTS is questionable.
- *Efficiency*: there is a lot of manual work involved, especially in step two. This, reduces the efficiency of the approach
- *Generality*: this approach works for GUI testing only
- *Scalability*: since this approach uses reverse engineering, and a reverse engineering tool usually requires a large amount of search spaces and resources, there is significant limitation on the complexity of the GUI under test.

[Fine+04] proposes a probability guided random regression test generation approach. The author uses probability of each test specification covering each coverage task to guide specification selection and test case generation. It requires that the probability of each test specification covering each coverage task be obtained from past executions of tests, or a Coverage Directed Generation Engine.

Three steps of Fine's approach are as follows:

1) Gather probability information
2) Determine specification that should be included in the RTS to make the expected coverage probability maximum
3) Generate test cases for the selected specification until expected coverage is achieved.

As a random testing approach, this approach belongs to *retest-all regression test* strategy, and it has major shortcomings.

- *Effectiveness*: randomly generated RTSs do not always cover modified and affected portion of the SUT, if it doesn't achieve 100% coverage. Moreover, probability from past execution is not always reflecting the behaviour of the generated regression test suite. Therefore, quality of the generated RTS is questionable
- *Efficiency*: this approach needs a lot of manual work. Hence, it cannot be classified as an efficient approach
- *Generality*: there is no limitation on using this approach
- *Scalability*: this approach can be applied to large scale systems.

2.3 Existing Model-based Regression Test Selection Approaches

There are two distinct approaches for model-based regression testing, i.e., "regression testing of model artefacts" and "regression testing of software using model artifacts". In the first approach, the model design itself is tested, and in the second approach the model is used to form test suites that are executed on the code. In this section, we will give a critical review of model-base test selection techniques that form RTS to test implementation of the model.

Briand et al. addressed a RTT using use case diagrams, sequence diagrams, and class diagrams in [Briand+02]. They proposed a formal mapping between design changes and a classification of regression test cases, i.e., three categories: reusable, retestable, and obsolete.

Given UML models that represent the SUT, the regression analysis in Briand et al.'s approach has three steps:

1). Compare class diagrams, use case diagrams, sequence diagrams to identify design changes.

2). Realize consistency check

3) Classify test cases into three categories. Select retestable test case as regression test cases.

This approach belongs to *selective regression testing* strategies. Applying our evaluation framework presented in Section 2.1, we find that:

- *Effectiveness*: only direct effects of modifications to the SUT are considered in this approach, i.e., only test cases traversing changed design are considered as retestable test cases. Therefore, test cases that test indirect effects of the modification may be missed.
- *Efficiency*: as the authors said, this approach could be automated. It is an efficient approach.
- *Generality*: this is a UML-based approach. Therefore, it is appropriate for any SUT that are modeled using UML.
- *Scalability*: UML can be used to model large scale. Thus, this approach can be applied to large scale systems.

[Chen+02] proposed a regression test selection strategy using risk analysis. System behaviors are described in a UML activity diagram, and a traceability matrix between the activity diagram and test cases is obtained. The activity diagram was treated like a control flow graph (CFG), and baseline and delta versions of the CFG were compared to identify the changes. Test sequences containing changed activity diagram elements were selected using the traceability matrix.

There are three steps in Chen et al.'s approach:

1). Obtain a traceability matrix to track the coverage of test sequences and activity diagram elements (edges)

2). Identify activity diagram elements that are affected by changes (affected entities).

3). Select test cases that traverse the affected entities as regression test cases.

This approach uses modification information for selecting regression test cases. Hence, it is a *selective regression testing* strategy. It is evaluated as follows:

- *Effectiveness*: Chen et al.'s approach can significantly reduce the size of a given test suite. But it only considers testing direct effects of modifications to the SUT, i.e.,

test cases traversing the affected entities in the activity diagrams are selected. Thus, test cases to test indirect effects of the modification may be missed.

- *Efficiency*: if information to obtain the traceability matrix in step 1) is kept electronically, this approach can be totally automated. It is an efficient approach.
- *Generality*: as a UML-based approach, this approach is appropriate for state-based systems.
- *Scalability*: UML activity diagrams can be used to model large systems. Thus, this approach can be applied to large scale systems.

In [Farooq+07], the authors presented a UML-based selective regression testing strategy that uses state machines and class diagrams for change identification. They assumed that the state machine is flattered, and attributes of the context class can be distinguished from local variables and parameters. They provided change definitions for UML state machines and class diagrams, and use the definitions to classify test cases and form RTS.

Farooq et al.'s regression testing process contains the following steps:

1). Identification of changes in the delta version of the SUT

2). Analysis of elements that are indirectly affected due to some dependencies.

3). Inspection of the effects of both changed elements and indirectly affected elements on the base line test suite.

4). Discarding the test cases that are no longer valid and selecting a subset of the baseline test suite that needs to be exercised for regression testing.

This approach is a *selective regression testing* strategy, and has advantages and disadvantages.

- *Effectiveness*: unlike the previous two approaches, Farooq et al.'s approach deals with both direct effects and indirect effects of modifications to the SUT. A transition is considered as a modified transition if its associated transitions are modified. RTSs formed by this approach can be used to test both direct effects and indirect effects of the modifications. But, in our opinion, there might be other transitions that are affected by the modifications.
- *Efficiency*: this approach can be automated. Thus, it can be an efficient approach.
- *Generality*: this approach is appropriate for systems that can be modeled in UML.

- *Scalability*: since UML is used widely in modeling large scale system, this approach can be applied to large scale systems.

Korel et al. presented a model-based RTS reduction approach in [Korel+02]. The authors used dependence analysis of EFSM model to reduce the size of a given RTS. This approach identified the differences in the original and modified EFSM model as a set of elementary modifications (EMs) on EFSM transitions: elementary addition of a transition and elementary deletion of a transition. It assumed that interactions between EFSM transitions are represented as EFSM dependencies between transitions, and changes of a transition were expressed by a pair of elementary addition and an elementary deletion.

Three major steps of this approach are as follows:

1) Identify the difference between the original EFSM and modified EFSM model as a set of EMs: elementary addition of a transition and elementary deletion of a transition.
2). During traversal of each test case, compute three interaction patterns (IPs) for each EM.
3). A test is included in the reduced test suite if at least one of its IPs does not exist for any of the test in the reduced test suite.

This approach is a *selective regression testing* strategy. But it is not complete, and has shortcomings.

- *Effectiveness*: Korel et al.'s approach considers testing both direct effects and indirect effects of modifications to the SUT. This approach can significantly reduce the size of a given test suite. But, although Korel et al. have defined some data and control dependencies arising from each EM informally, some dependence types were not identified.

  In this thesis, we adopt the notion introduced in Korel et al.'s work, utilize some of their definitions for control and data dependencies, revise some of their definitions for control and data dependencies, and define new dependencies that have not been identified by them. We also extend the use of dependence analysis to RTS generation.

- *Efficiency*: this approach can be automated. It is an efficient approach even it has some shortcoming, i.e., this approach considers only two types of EM, i.e., addition

or deletion of a transition. Change of a transition is expressed by a pair of deletion of existing transition and addition of a replacing transition. However, unless the starting or terminating state of the transition needs to be changed, such a representation becomes costly.

- *Generality*: this approach is appropriate for state-based systems that can be modeled using formal description languages, such as EFSM, SDL, and ESTELLE.
- *Scalability*: this approach can be applied to large scale systems.

Naslavsky et al. published their work in [Naslavsky+07]. Their approach used model transformation techniques to create an infrastructure composed of UML model, model-based control flow graph model, traceability model and test generation hierarchy model. The traceability model was used to identify regression test cases from the test generation hierarchy model. The original UML model and the modified UML model were prerequisites of this approach. The original test suite had to been designed using Naslavsky et al.'s test case generation approach.

Steps of Naslavsky et al.'s approach are as follows:

1)The original UML model and the modified UML model are compared to get a UML model describing the changes.

2)Use the model describing the changes and the traceability model to identify a subset of elements in the model-based control flow graph model impacted by the changes.

3)Create a new version of the model-based control flow graph model.

4)Using the traceability model to identify entities in the test generation hierarchy model impacted by the modifications to the model-based control flow graph model. Test cases under impacted paths are selected as regression tests.

As a *selective regression testing* strategy, this approach has the following pros and cons:

- *Effectiveness*: only direct effects of modifications to the SUT are considered in this approach, i.e., only test cases traversing the modified entities are selected. Therefore, test cases to test indirect effects of the modifications may be missed.
- *Efficiency*: this approach can be automated. It is an efficient approach.

- *Generality*: this approach is appropriate for systems that can be modelled using UML. But the original test suite has to be created under the same infrastructure. This limits the use of this approach.
- *Scalability*: this approach can be applied to large scale systems.

Wu and Offutt [Wu+03] presented a UML based RTT for component based software using collaboration diagrams, class diagrams and state machines. They provided a UML-based mechanism to represent different types of modifications. Based on the UML-based representation, they discussed regression testing strategies for different maintenance activities.

This approach is a *selective regression testing* strategy. But it focuses on regression testing strategy and not on change impact analysis.

- *Effectiveness*: both direct effects and indirect effects of modifications to the SUT are considered in this approach. But the authors didn't provide detailed regression analysis techniques for each type of modifications. Under proper regression analysis, RTSs formed based on this approach can be used to test both direct effects and indirect effects of the modifications.
- *Efficiency*: Wu et al. did not provide detailed change impact analysis. They only give examples and suggested retesting strategies. More work is needed before the approach can be implemented by tools.
- *Generality*: this approach is appropriate for component-based systems.
- *Scalability*: as a UML-based technique, this approach can be applied to large scale systems.

We summary the above discussions of related work in Table 2-1. Effectiveness, Efficiency, Generality, and Scalability are measured as High, Medium, or Low.

Table 2-1.Summary of related work

Type	RTT	Effectiveness	Efficiency	Generality	Scalability
Code-based regression test generation	[Gupta+92]	Medium	High	High	Medium
	[Korel+98]	Low	Medium	Low	Medium
Requirement-based regression test generation	[Tahat+01]	Low	High	High	High
	[Xu+04]	Low	Medium	High	High
	[Mayr-hauser+99]	Low	High	High	High
	[White 96]	Medium	High	High	High
	[Memon+05]	Low	Low	Low	Low
	[Fine+04]	Medium	Low	High	High
Model-based regression test selection	[Briand +02]	Medium	High	High	High
	[Chen+02]	Medium	High	Medium	High
	[Farooq +07]	High	High	High	High
	[Korel+02]	High	High	High	High
	[Naslavsky +07]	Medium	High	Medium	High
	[Wu+03]	High	N/A	High	High

2.4 Classification of Modifications and Types of Regression Testing

During software development and maintenance, many modifications may occur when the software system is corrected, updated, or fine-tuned. According to the maintenance activities performed on the software system, White classified the modifications into three categories [White 91]:

- Corrective maintenance: Involves correcting software failures in order to keep the software working properly. This type of maintenance is usually referred to as "fixes".
- Adaptive maintenance: Involves adapting the system in response to new user requirements or change in the operational environments.
- Perfective maintenance: Involves any enhancement to the system. The objective may be to provide additional functionality, tune system performance, or improve system maintainability by restructuring the software architecture.

In corrective maintenance, in most situations the specification is not likely to be changed and no new modules are likely to be introduced. But sometimes, fixing identified code errors leads to specification changes. Many modifications occurring during the development phase are similar to corrective maintenance type modifications. In adaptive maintenance and perfective maintenance, changes occur in user requirements and / or the system specifications. New functionality is often introduced. Modifications occurring in the development phase to implement new or modified specifications are similar to these two types of modifications. Leung and White referred to both adaptive maintenance and perfective maintenance as progressive maintenance [White 91].

Based on the classification of modifications addressed above, we can classify regression testing into two categories:

- Corrective regression testing: To test corrective maintenance, that is, test a fix to existing specifications.
- Progressive regression testing: To test progressive maintenance, that is, test a modification to reflect a modified specification.

In this thesis, our discussions about regression testing will be based on this classification.

2.5 Controlled Regression Testing Assumption (CRTA)

In regression testing, we try to determine if a modified system has the same behaviour as the unaffected portion of the previous version of the system by running all or some of the same tests. To be safe without being too inefficient and too conservative, our approaches

are based on an assumption called the controlled regression testing assumption (CRTA). In [Rothermel+96] CRTA was described as follows:

When Q' is tested with T, we hold all factors that might influence the output of Q', except for the code in Q', constant with respect to their states when we tested Q with T.

Ideally, regression tests should be run while CRTA holds. In case that CRTA does not hold, the behaviours of an SUT may be affected by the test environment. When CRTA holds, we can identify whether the behaviours of a system have been changed after modification. In our research, we assume that CRTA holds.

This can be quite difficult in a complex, multi-platform test environment. Testers may have to identify and document uncontrollable factors and the test cases that are potentially affected by them in practice.

2.6 EFSM Model

System requirements can be modeled using formal description languages, such as EFSM or SDL. In this thesis, we concentrate on the EFSM system models.

An EFSM is formally defined as an 9-tuple $<S, Start, Exit, I, O, P_I, P_O, L, T>$ where

- S is a nonempty finite set of states
- *Start* is the *start* state and *Exit* is the *exit* state
- I is a nonempty finite set of input interactions
- O is a nonempty finite set of output interactions, including null output
- P_I is a nonempty finite set of input interaction parameters
- P_O is a nonempty finite set of output interaction parameters
- L is the nonempty finite set of local variables
- T is a nonempty finite set of transitions

The set of all variables defined and used in an EFSM is denoted $V = P_I \cup P_O \cup L$.

Each transition $t \in T$ is a 6-tuple $<s_s, s_t, i, c, o, a>$ where

- $s_s, s_t \in S$ are the *starting* and *terminating* states of t, respectively
- $i \in I$ is the input interaction (possibly with parameters from P_I) of t
- c is the *enabling condition* of t which is a Boolean expression defined over $P_I \cup L$
- $o \in O$ is the output interaction (possibly with parameters from P_O) of t

- a is a sequence of actions of t expressed as functions f: $V \rightarrow V$

In a given EFSM model, it is assumed that every state is reachable from *Start* and *Exit* is reachable from every state. Model executability (assuming that enough and sufficiently precise detail is provided) may be desirable for automatic synthesis of source code, test suite prioritization, etc. But in our research, we do not need the executability of EFSM models. We are interested in modifications on transitions instead of states, since transitions represent active elements of the EFSM model. For each transition, we assume there is at most one EM in the given set of EMs. Multiple modifications on the same transition are thus assumed to be combined into one EM.

In this thesis, we use an EFSM to model requirements of the SUT. Each requirement r of the SUT can be mapped onto the EFSM model. In our discussion, usually, a requirement r is specified as a sequence of transitions, and a function f is mapped to (a part of) a transition of the EFSM model. Any changes to requirements or functions will be mapped to changes to transitions in the EFSM model. Hence, once an EFSM model is created to represent requirements of a given SUT, testing changed requirements or functions is equivalent to testing changed transitions in the EFSM model. We will discuss mapping modifications on requirements/ functions onto the EFSM model later (in Section 2.8).

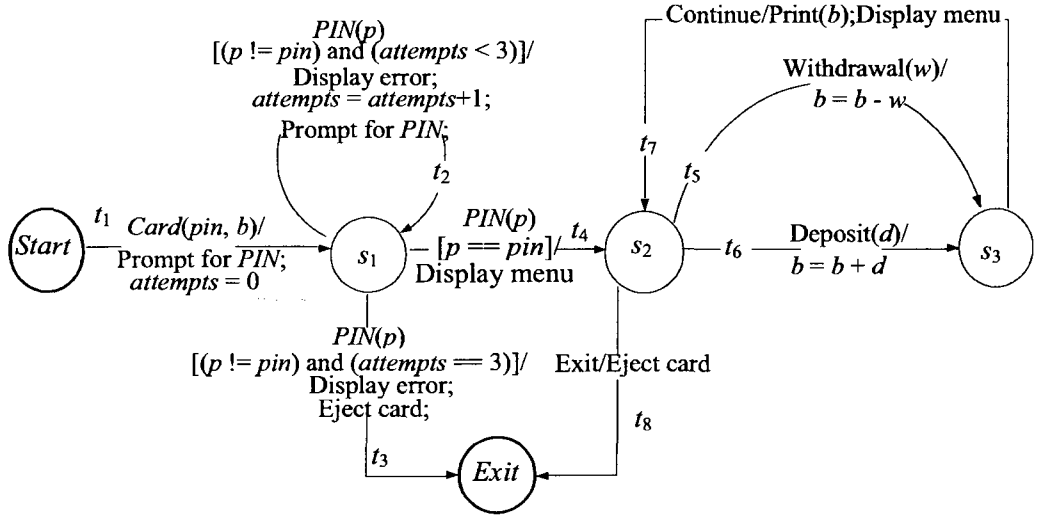


Figure 2-1. EFSM model of a simplified ATM system

An EFSM is represented as a directed graph where states are represented as nodes and transitions as directed edges between states. Figure 2-1 shows an EFSM model of a simplified ATM system. This example EFSM is introduced by [Korel+02], and is useful to illustrate differences between their approach and ours. We use this example EFSM as a running example in this thesis.

In our discussion, we represent each transition in an EFSM with a transition ID t_i and a 6-tuple $<s_s, s_t, i, c, o, a>$. For example, the set of transitions in the example EFSM is given in Table 2-2.

Table 2-2. Transition set T for the example EFSM

Tran.#	s_s	s_t	i	c	o	a
t_1	*Start*	s_1	*Card(pin, b)*		Prompt for *PIN*	*attempts* = 0
t_2	s_1	s_1	*PIN(p)*	(*p* != *pin*) and (*attempts*< 3)	Display error; Prompt for *PIN*	*attempts* =*attempts*+1
t_3	s_1	*Exit*	*PIN(p)*	(*p* != *pin*) and (*attempts*==3)	Display error; Eject card	
t_4	s_1	s_2	*PIN(p)*	*p* == *pin*	Display menu	
t_5	s_2	s_3	Withdrawal(*w*)			*b* = *b* - *w*
t_6	s_2	s_3	Deposit(*d*)			*b* = *b* + *d*
t_7	s_3	s_2	Continue		Print(*b*); Display menu	
t_8	s_2	*Exit*	Exit		Eject card	

2.7 EFSM Dependencies

In the EFSM model, data dependence and control dependence may exist between transitions. Dependence analysis focuses on interactions between transitions in the model.

2.7.1 Data Dependence

Data dependence captures the notion that one transition defines a value for a variable and the same or some other transition may potentially use this value [Korel+02]. Before defining data dependence, we first adopt the following terminology.

A definition (def) of $v \in V$ is an occurrence of v in a transition by which v takes a value (e.g., an occurrence of v on the left hand side of an action or in an input interaction). A use of $v \in V$ is an occurrence of v in a transition which directly affects the computation being performed (e.g., an occurrence of v on the right hand side of an action), or allows one to see the result of some earlier definitions (e.g., an occurrence of v in an output interaction), or directly affects the control flow in the EFSM (e.g., an occurrence of v in the enabling condition of a transition).

A *path* $(t_1\ t_2\ \ldots\ t_{m-1}\ t_m)$ is a sequence of (not necessarily distinct) adjacent transitions. A path $(t_1\ t_2\ \ldots\ t_{m-1}\ t_m)$ is said to be from the starting state of t_1 to the terminating state of t_m. A path $(t_1\ t_2\ \ldots\ t_{m-1}\ t_m)$ is a definition-clear path from t_1 to t_m with respect to (w.r.t.) $v \in V$ if either $m = 2$ or $m > 2$ and v is not defined at $t_2\ \ldots\ t_{m-1}$. A pair (def of v at t_1 and use of v in t_m) is a definition-use pair (du-pair in short) w.r.t. v if def of v at t_1 is the last definition of v at t_1, use of v at t_m is a use of v in t_m (before v is (possibly) defined at t_m), and there is a definition-clear path from t_1 to t_m w.r.t. v [Hong+03].

Let t_i and t_k be (not necessarily distinct) transitions, and $v \in V$ in an EFSM. There is a data dependence (DD) from t_i to t_k w.r.t. v, denoted DD(t_i, t_k, v), iff v is defined in t_i as the last definition of v in t_i, v is used in t_k (before v is (possibly) defined in t_k), and there is a definition-clear path from t_i to t_k w.r.t. v. In other words, a DD(t_i, t_k, v) is a du-pair (def of v in t_i, use of v in t_k) w.r.t. v.

Figure 2-2 shows a graphical representation of a DD(t_i, t_k, v).

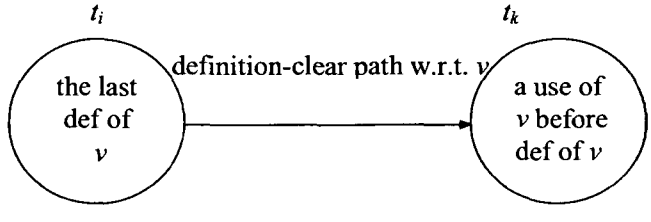


Figure 2-2. Data dependence DD(t_i, t_k, v)

In the example EFSM (Figure 2-1), t_1 defines b, t_5 uses b, and along t_1 (t_2) t_4 t_5, b is not redefined. Thus, there exists a DD from t_1 to t_5 w.r.t. b. Also, t_5 defines b, t_5 uses b, and along t_5 t_7 t_5, b is not redefined. Therefore, a DD exists from t_5 to t_5 w.r.t. b.

2.7.2 Control Dependence

Control dependence captures the notion that one transition may "influence" the traversal of another transition. Control dependence between transitions is defined in terms of the concept of post-dominance [Korel+02].

Let s_1 and s_2 be two distinct states, and t be an outgoing transition from s_1 in an EFSM (Figure 2-3):

1. s_2 post-dominates s_1 iff s_2 is on every path from s_1 to the Exit.
2. s_2 post-dominates t iff s_2 is on every path from s_1 to the Exit through t.

Note that if s_2 post-dominates t, then s_2 post-dominates the terminating state of t.

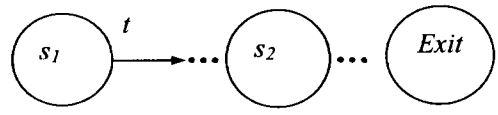


Figure 2-3. Post-dominance

Let t_i and t_k be two outgoing transitions from s_1 and s_2, respectively (see Figure 2-4). There is a control dependence (CD) from t_i to t_k, denoted CD(t_i, t_k), iff s_2 does not post-dominate s_1 and s_2 post-dominates t_i.

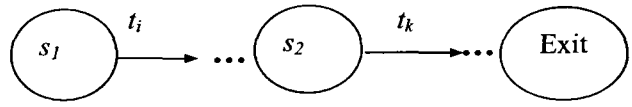


Figure 2-4. Control dependence CD(t_i, t_k)

In our example EFSM, s_2 does not post-dominate s_1 but s_2 post-dominates t_4. Therefore, there is a CD from t_4 to t_5.

2.7.3 Static Dependence Graph (SDG)

DDs and CDs in an EFSM are graphically represented in a Static Dependence Graph (SDG). In SDG, nodes represent EFSM transitions and directed edges represent EFSM data and control dependencies. Let C and D be the set of all control and data dependencies in an EFSM, respectively. That is, $C = \{(t_i, t_k) \mid (t_i, t_k)$ is a CD from t_i to $t_k\}$ and $D = \{(t_i, t_k, v) \mid (t_i, t_k, v)$ is a DD from t_i to t_k w.r.t. $v\}$. The SDG of a given EFSM is constructed as a directed graph $G[E, N]$ as follows:

Let $t_i, t_k \in T$ and $v \in V$ of the EFSM.

$E \leftarrow \varnothing$; $N \leftarrow \{n_i \mid n_i$ for each $t_i \in T\}$

For each $(t_i, t_k) \in C$, $E \leftarrow E \cup \{$a dashed edge from t_i to $t_k\}$.

For each $(t_i, t_k, v) \in D$, $E \leftarrow E \cup \{$a solid edge from t_i to $t_k\}$.

Figure 2-5 shows SDG of the example EFSM of Figure 2-1.

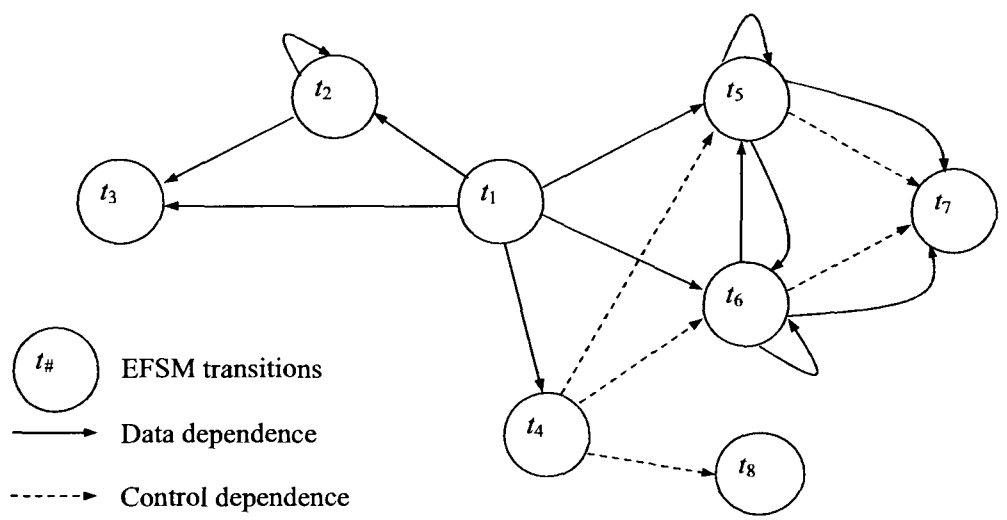


Figure 2-5. SDG for the example EFSM

2.8 Modifications on Requirements in an EFSM

In this thesis, as was done by Korel et al., we consider both the original and the modified EFSM models. The set of modifications are either obtained before the original EFSM

model is modified to construct the modified EFSM model, or by comparing the original EFSM model with the modified EFSM model.

2.8.1 Requirements and Functions

In our discussion, the specification of an SUT is defined by a set of requirements R. We consider each requirement $r \in R$ as a collection of functions drawn from a set F of functions where each function $f \in F$ can contribute to several requirements. In this context, functions show how the requirements can be satisfied rather than how to implement the system. That is, functions are at the specification/ design level, not at the implementation level. When we use EFSM as modelling language, usually, a function is specified as part of a transition or one complete transition. Since each individual requirement is satisfied by a sequence of functions, requirements are associated with EFSM through transitions. Figure 2-6 shows the relation between requirements and functions.

An individual requirement $r_i \in R$ consists of a requirement ID, and a sequence of functions $F(r_i)$ that work together to satisfy r_i. Since a requirement is satisfied by at least one function, $F(r_i)$ cannot be empty. An individual function $f_j \in F$ consists of a function ID, a description of the partial system behaviour, and a set of requirements $R(f_j)$ that the function contributes to. We take (part of) a transition in the EFSM as the function f_j's description. It is a 5-tuple $<t_i, i, c, o, a>$, [1] where t_i is the transition id in the EFSM model that satisfies f_j. When a function is not used by any requirement, $R(f_j)$ is empty.

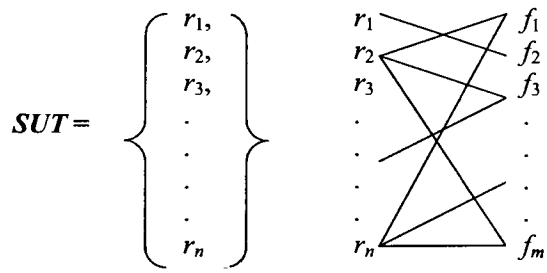


Figure 2-6. Requirements and functions

[1] The definition of the other four elements of the 5-tuple is the same as the 6-tuple of transition. Please refer to Section 2.6 for details.

To summarize, R and F have the following structures, which are shown in Table 2-3:

$R = \{r_1, r_2, r_3, ..., r_n\}$

$F = \{f_1, f_2, f_3, ..., f_m\}$

$F(r_i)$: the sequence of functions that work together to satisfy requirement r_i.

$R(f_j)$: the set of requirements that function f_j contributes to.

Table 2-3. Set of requirements R and set of functions F

R:

Req. ID	Func. Sequence
r_1	$F(r_1)$
r_2	$F(r_2)$
...	...
r_n	$F(r_n)$

F:

Func. ID	Description	Req. Set
f_1	$<t_i, i, c, o, a>$	$R(f_1)$
f_2	...	$R(f_2)$
...	...	...
f_m	...	$R(f_m)$

In our research, we are working on selective regression testing strategies, which try to re-test affected parts of an SUT only, not the whole SUT. Adapting regression testing definition by IEEE, together with our definition of requirements, functions and transitions, we give our definition of requirement-based regression testing as follows:

Let R be a set of requirements of a system under test Q. Let M be a set of requirements modifications which resulted a set R' of modified requirements and a modified system Q'. Requirement-based regression testing is selective retesting of Q' to verify that modifications have not caused unintended effects and that Q' still complies with R'.

The selective regression retesting only tests a subset of Q'. To ensure the quality of Q' by testing a subset of Q' only, this subset of Q' must contain both changed portion of Q and unchanged portion of Q which may be affected by the modifications.

2.8.2 Modification on Requirements and Functions

We denote a set of modifications on R and F as M. In set M, there can be six different types of modifications: add a new function, delete a function, or change a function, and

add a new requirement, delete a requirement, or change a requirement. A modification of a function f_j will affect requirements in $R(f_j)$, and a modification of a requirement r_i may affect functions in $F(r_i)$. It may be argued that any change c_k can be expressed as a deletion d_j and an addition a_i. However, there exist minor changes that would be expensive in expressing as deletions and additions. Such minor changes are represented by c_k in M.

2.8.2.1 Modification set *M*

Since a requirement is satisfied by a sequence of functions, we will deal with modifications on functions first. Therefore, M is sorted in the way that all elements on functions are ahead of elements on requirements.

In M each type of modifications is given as follows:

- Add a new function: add(f_j, desc, $R(f_j)$),
  - f_j is a unique function id to identify the new function
  - desc is (part of) a transition in the EFSM as the function's description, which is
    - if f_j is a complete transition in the EFSM, the description is a transition id and a 6-tuple $<s_s, s_t, i, c, o, a>$
    - if f_j is part of a transition in the EFSM, the description is a transition id and a 4-tuple $<i, c, o, a>$[2]
  - $R(f_j)$ is the set of all requirements that the new function f_j will contribute to (Note that f_j might contribute to some new requirements not currently in R)
- Delete an existing function: delete(f_j) where $f_j \in F$
- Change an existing function: change (f_j, desc) where $f_j \in F$, and desc is the updated version of the existing description of $f_j \in F$.
- Add a new requirement: add(r_i, $F(r_i)$),
  - r_i is a unique requirement id to identify the new requirement
  - $F(r_i)$ is a sequence of functions that will contribute to r_i (Note that r_i might be contributed to by some new functions not currently in F)
- Delete an existing requirement: delete(r_i) where $r_i \in R$

[2] The definition of each element of the 4-tuple is the same as the 6-tuple.

- Change an existing requirement: change(r_i, $F(r_i)$) where $r_i \in R$, and $F(r_i)$ is the new sequence of functions that will contribute to r_i (Note that r_i might be contributed to by some new functions not currently in F)

A new or changed requirement might be contributed to by some new functions not currently in F. But these new functions have to be already identified when preparing M. That means there have to be elements for these functions in M. Also, for the functions that need to be changed due to the new or changed requirements, there have to be elements for these functions in M as well. In M, all these elements appear before elements for the new or changed requirements.

2.8.2.2 Preparing *M*

Usually modifications on Q are given in textual format. Since a requirement is satisfied by a sequence of functions, when we parse the texts, we will deal with modifications on functions first. Any modification on requirements will be kept and will be handled after all function modifications have been processed. Thus, the textual modification will be parsed twice. In the first round, only function modifications are put into M. In the second round, requirement modifications are added into M.

Initially, M is empty. During the parsing, we deal with each of the six types of modifications as follows:

- Add a new function: If there is an a_f for this function in M, stop. Otherwise, a unique function id f_i is assigned to the new function.
  - Identify all existing requirements that the new function f_i will contribute to. Put these requirements' ids into $R(f_i)$. $R(f_i)$ could be empty.
  - Identify input, enabling condition, output, and action for f_i. If f_i is a new transition, find out the starting state, terminating state for f_i, description is a 6-tuple $<s_s, s_t, i, c, o, a>$. If f_i is part of a transition t_i, the description is a 5-tuple $<t_i, i, c, o, a>$.
  - a_{fi} = add(f_i, description, $R(f_i)$), attach a_{fi} into M.
- Delete an existing function:
  - If there is an element for f_i in M, report a warning and stop.
  - d_{fi} = delete(f_i), attach d_{fi} into M.

- Change an existing function:
  - If there is an element for f_i in M, report a warning and stop.
  - Identify the transition id, new version of input, enabling condition, output, and action for f_i. The description is a 5-tuple $<t_i, i, c, o, a>$.
  - c_{fi} = change(f_i, description), attach c_{fi} into M.
- Add a new requirement: A unique requirement id r_i is assigned to the new requirement.
  - Case i): the new requirement r_i is given in terms of only existing functions.
    - Identify the sequence of functions $F(r_i)$ to satisfy r_i
    - There should be at least one function id in $F(r_i)$ ($F(r_i) \neq \varnothing$). Otherwise, report a warning.
    - a_{ri} = add(r_i, $F(r_i)$), add a_{ri} into M.
  - Case ii): r_i is given in terms of new functions to be added or some existing functions to be changed.
    - Identify all existing functions that the new requirement r_i will use.
    - For each new function introduced because of r_i,
      1) if there is no a_{fi} in M for this new function:
         a. A unique function id f_i is assigned to the new function.
         b. Put r_i into $R(f_i)$.
         c. Identify input, enabling condition, output, and action for f_i. If f_i is a new transition, find out the starting state, terminating state for f_i, description is a 6-tuple $<s_s, s_t, i, c, o, a>$. If f_i is part of a transition t_i, description is a 5-tuple $<t_i, i, c, o, a>$.
         d. a_{fi} = add(f_i, description, $R(f_i)$), attach a_{fi} into M.
      2) if there is a_{fi} in M: Update $R(f_i)$ to include r_i
    - For each function changed because of adding r_i, if there is no c_{fi} in M for this changed function, handle it according to the previous rule "Change an existing function"
    - Identify the sequence of functions $F(r_i)$ to satisfy r_i. There should be at least one function id in $F(r_i)$ ($F(r_i) \neq \varnothing$). If $F(r_i)$ is empty, report a warning.
    - a_{ri} = add(r_i, $F(r_i)$), insert a_{ri} into M.

- Delete an existing requirement:
  - If there is an element for r_i in M, report a warning.
  - d_{ri} = delete(r_i), insert d_{ri} into M.
- Change an existing requirement:
  - If there is an element for r_i in M, report a warning.
  - There could be several different types of changes on requirements. Each type needs to be handled differently.
    - Some functions that contributed to the requirement r_i are changed: for each of these functions, if there is no c_{fi} in M, handle it according to the previous rule "Change an existing function"
    - Some functions that contributed to the requirement r_i are deleted: for each of these functions, if there is no d_{fi} in M, handle it according to the previous rule "Delete an existing function", and update $F(r_i)$.
    - Some functions that used to contribute to r_i do not belong to $F(r_i)$: update $F(r_i)$.
    - r_i use new functions: For each new function introduced because of r_i,
      1) if there is no a_{fi} in M for this new function:
         a. A unique function id f_i is assigned to the new function.
         b. Put r_i into $R(f_i)$.
         c. Identify input, enabling condition, output, and action for f_i. If f_i is a new transition, find out the starting state, terminating state for f_i, description is a 6-tuple $<s_s, s_t, i, c, o, a>$. If f_i is part of a transition t_i, description is a 5-tuple $<t_i, i, c, o, a>$.
         d. a_{fi} = add(f_i, description, $R(f_i)$), attach a_{fi} into M.
      2) if there is a_{fi} in M: Update $R(f_i)$ to include r_i
      3) update $F(r_i)$.
    - Only the sequence of functions $F(r_i)$ to satisfy r_i is changed: update $F(r_i)$.
  - c_{ri} = change(r_i, $F(r_i)$), attach c_{ri} into M.

After the two round parsing, for each requirement or function, there is at most one element in M. The last step is sorting M to make sure that all elements on functions are ahead of elements on requirements. We would like to point out that M can contain

modifications for both *corrective maintenance* and *progressive maintenance.* For *corrective maintenance*, no mater if the specifications of the requirement/ function are changed or not, there should be debugging requests. Therefore, we will include one change element for the requirement/ function in M for the fix request.

2.8.3 Modifications on the EFSM Model

Based on previous definitions on requirements, functions and transitions, we can easily map modifications on requirements and functions to EFSM model.

2.8.3.1 Set of modifications on transitions M_T

Changing a transition t_i does not necessary affect DDs that t_i involves. In two situations changed transitions will affect DDs: i) changes happen on def or use of variables in transitions; ii) changed transitions cause changes on a definition-clear path. To do accurate DD analysis, we need to track changes on variables in transitions. Therefore, we introduce M_T to keep information of variables.

M_T has the structure as shown in Table 2-4. Definition of each column of Desc_old and Desc_new sections are the same as definition of elements of a transition in T.

Table 2-4. Set of modifications on transitions M_T

Transition ID	Desc_old						Desc_new					
	s_s	s_t	i	c	o	a	s_s	s_t	i	c	o	a
t_1												
t_2												
...												
t_p												

When we add a new transition, Desc_old section will be empty. Desc_new section will have the same information as the added transition. When we delete an existing transition, Desc_new section will be empty, and Desc_old section will keep the same information as the deleting transition.

2.8.3.2 Processing *M*

Let R_c, F_c, T_c be the set of requirements, functions, and transitions for current version of an SUT. We will take R_c, F_c, T_c, M as input, after processing M, we will get R_m, F_m, T_m, and M_T. R_m, F_m, and T_m have the same structure as R_c, F_c, and T_c, except that R_m, F_m, T_m all have a extra field that is used as a flag. Originally, R_m, F_m, and T_m are copies of R_c, F_c, and T_c. The extra columns in each of R_m, F_m, T_m is empty. M_T is empty initially.

For each type of elements in M, we take different actions.

- Add a new function: add (f_j, desc, $R(f_j)$),
  - If f_j is a complete transition,
    - t_i is a unique transition id to identify the new transition
    - add t_i into T_m, set the 6-tuple for t_i accordingly, and set the flag for t_i as NEW
  - If f_j is part of a transition $t_i \in T_c$,
    - If there is no element in M_T for t_i: add an element into M_T: change (t_i, desc_old, desc_new), where $t_i \in T_c$, desc_old is the old version of description of t_i in F_c, and desc_new is the new version of description of t_i with desc in (f_j, desc, $R(f_j)$) been included.
    - If there is an element in M_T for t_i: update desc_new of the element to include the given new description in add (f_j, desc, $R(f_j)$).
    - In T_m, set the flag for t_i as CHANGED, and update the 6-tuple of t_i accordingly
  - Add f_j into F_m, set description of f_j as <t_i, i, c, o, a>, and the flag for f_j as NEW.
  - In R_m, $\forall r \in R(f_j)$, add f_j into $F(r)$
- Delete an existing function: delete(f_j)
  - If f_j is a complete transition $t_i \in T_c$,
    - In T_m, set the flag for t_i as DELETED
  - If f_j is part of a transition $t_i \in T_c$,
    - If there is no element in M_T for t_i: add an element into M_T: change (t_i, desc_old, desc_new), where $t_i \in T_c$, desc_old is the old version of description of t_i, and desc_new is the old version of description of t_i with f_j been excluded.

- If there is an element in M_T for t_i: update desc_new of the element to exclude the description for f_j.
- In T_m, set the flag for t_i as CHANGED, and update the 6-tuple of t_i accordingly

  - In R_m, $\forall\ r \in R(f_j)$, update $F(r)$ to exclude f_j , and set the flag for r as CHANGED
  - Delete f_j from F_m

- Change an existing function: change(f_j, desc)
  - In F_m, set the flag for f_j as CHANGED, update description accordingly
  - In R_m, $\forall\ r \in R(f)$, set the flag for r as CHANGED
  - If transition $t_i \in T_c$ satisfies f_j,
    - If there is no element in M_T for t_i: add an element into M_T: change (t_i, desc_old, desc_new), where $t_i \in T_c$, desc_old is the old version of description of t_i, and desc_new is the old version of description of t_i with desc in change (f_j, desc) been included.
    - If there is an element in M_T for t_i: update desc_new of the element to include the given new description in change (f_j, desc).
    - In T_m, set the flag for t_i as CHANGED, and update the 6-tuple of t_i accordingly
- Add a new requirement: add(r_i, $F(r_i)$)
  - Add r_i into R_m, and set the flag for r_i as NEW
  - In F_m, $\forall f \in F(r_i)$, update $R(f)$ to include r_i
- Delete an existing requirement: delete(r_i)
  - In F_m, $\forall f \in F(r_i)$, update $R(f)$ to exclude r_i
  - In R_m, delete r_i
- Change an existing requirement: change(r_i, $F(r_i)$)
  - In R_m, set the flag for r_i as CHANGED
  - In F_m,
    - $\forall f \in F(r_i)$, update $R(f)$ to include r_i
    - $\forall f \in F_c(r_i)$, if $f \notin F(r_i)$, update $R(f)$ to exclude r_i

After this process, we get both M_T and T_m. The reason to keep M_T is for DD analysis. Changing a transition may not affect all DD related to this transition. Without M_T, we can only know which variables are really affected by comparing T_m with T_c. Dependence analysis will be based on T_m. That is, we take all transitions which are marked as NEW, CHANGED, or DELETED from T_m for dependence analysis.

2.9 Set Covering Problem (SCP) and Greedy Algorithm

Set covering problem (SCP) is a classical question in computer science and complexity theory. In this section, we will give an overview of the SCP problem and one of its solutions. This solution is adopted in our RTSG and RTSR methods.

2.9.1 Set Covering Problem

SCP is defined as follows [Johnson 06]: Given a collection A of subsets A_i of a base set U of n elements, find a minimum-sized subcollection C' whose union equals U.

In the set covering decision problem, the input is a pair (U, A) and an integer k; the question is whether there is a set covering of size k or less. In the set covering optimization problem, the input is a pair (U, A), and the task is to find a set covering which uses the fewest sets.

SCP was one of Karp's 21 NP-complete problems [Karp 72] shown to be NP-complete in 1972. It is one of the oldest and most studied NP-complete problems [Garey+79]. The decision version of set covering is NP complete, and the optimization version of set cover is NP hard.

SCP is very common in practical applications, a number of exact and heuristic algorithms have been proposed in the literature. The proposed algorithms include genetic algorithm, simulation annealing algorithm, Ant Colony Optimization algorithm, artificial neural networks algorithm, greedy algorithm, etc. More complex SCP approximation algorithms have been studied. For some approaches, improvements come with a significant computational cost. This is mostly due to the fact that many algorithms run the basic computation - which is roughly equivalent in time complexity to the greedy algorithm - for many initial states (e.g. repeatedly adding or deleting subsets from the current solution). Where the execution time is crucial or the SCP size is prohibitive for an

advanced heuristic approach, the greedy algorithm remains a viable option [Abdalla+00] [Guha+99].

The greedy algorithm for set covering chooses sets according to one rule: at each stage, choose the set which contains the largest number of uncovered elements. Obeying this rule, the classical greedy algorithm for approximating a minimum cover, at each step, simply chooses the subset A_i which includes (covers) the largest number of remaining elements of U, deletes these elements from U, and repeats this process until the ground set U is empty. In case of a tie, i.e., when several subsets have the same covering size, the set with smaller subscript is usually chosen.

It can be shown that this algorithm achieves an approximation ratio of $H(s)$, where s is the size of the largest set and $H(n)$ is the n-th harmonic number:

$$\mathrm{H}(n) = \sum_{k=1}^{n} 1/k \leq \ln n + 1$$

Research results show that the greedy algorithm is essentially the best-possible polynomial time approximation algorithm for set cover under plausible complexity assumptions. Lund and Yannakakis (1994) [Lund+94] showed that set covering cannot be approximated in polynomial time to within a factor of $(\log_2 n)/2 \approx 0.72 \ln n$, unless NP has quasi-polynomial time algorithms. Feige (1998) [Feige 98] improved this lower bound to $(1 - O(1)) \cdot \ln n$, under the same assumptions, which essentially matches the approximation ratio achieved by the greedy algorithm. Raz and Safra [Raz+97] established a lower bound of $c \cdot \ln n$, where c is a constant, under the weaker assumption that P $\neq$ NP. A similar result with a higher value of c was proved by Alon, Moshkovitz, and Safra [Alon+06] in 2006.

2.9.2 Probability-driven Greedy Algorithm

In [Rampone 01], S. Rampone presented a probability-driven greedy algorithm for set covering. This is a not-randomized algorithm which is based on a probability distribution that leads the greedy choice. S. Dhar et al.'s research results in [Dhar+03] and [Dhar+04] show very good empirical performances of this algorithm.

In classical greedy algorithm, greedy choice is in effect random. Let $A = \{A_1, \ldots, A_n\}$. The greedy choice can be viewed as implicitly based on a probability assignment on the subsets, such that

$$P(A_i \mid U) > P(A_k \mid U) \Leftrightarrow |A_i \cap U| > |A_k \cap U|, i, k \in \{1, \ldots, n\} \quad \text{(i)}$$

In general, this assignment can be improved. If there is only one subset A_i in A that covers an element U_j in U, that subset will be certainly included in any cover of U, not matter what size the subset has. By extension, probability of a subset A_i covering an element U_j to be included in a cover decreases in the number of alternatives, i.e. the number of subset A_k such that A_k covers U_j.

S. Rampone's probability-driven greedy algorithm [Rampone 01] formalizes the underlying probability distribution as follows: Let $A = \{A_1, \ldots, A_n\}$ and $U = \{U_1, \ldots, U_m\}$.

For each element U_j and each subset A_i, the probability of A_i given U_j is defined as:

$$P(A_i \mid U_j) = IA_i(U_j) / \sum_{k=1}^{n} IA_k(U_j), \text{ where } i, k \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, m\} \quad \text{(ii)}$$

where $IA_i(U_j)$ is the characteristic function of the A_i, that is 1 if $U_j \in A_i$ and is 0 otherwise.

By taking the mean on U, the probability of A_i with respect to the ground set U is:

$$P(A_i \mid U) = (1/m) \sum_{j=1}^{m} P(A_i \mid U_j), \text{ where } i \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, m\} \quad \text{(iii)}$$

Formula (iii) is applied to define a greedy approximation algorithm for SCP as follows: choose the subset A_i whose probability with respect to U is the maximum, delete the elements of A_i from U, and repeat this process until the ground set U is empty.

2.10 Chapter Summary

In this chapter, we first provided background information about regression testing in general and listed common regression test patterns. Then, we gave a review of the related existing literature on regression test generation and test suite reduction. Also, we categorized modifications according to the published literature and restated the CRTA assumption used in our research. Moreover, we gave the definition of the EFSM model, data dependence (DD) and control dependence (CD) in the EFSM model, and Static Dependence Graph (SDG). We also defined requirement and function, and formally

described modifications on requirements/ functions. Finally, we gave an overview of the SCP problem and discuss one of its solutions.

3. Requirement-Based Regression Test Generation Method

In [Korel+02], the authors identified two types of EMs: an addition of a transition and a deletion of a transition. They stated that any complex modification to the model can be expressed as a set of these two types of EMs. Notice that the sequence with which these EMs are applied to the original model is not relevant for the purpose of [Korel+02]. However, minor changes would be expensive in expressing as deletions and additions. Therefore, we will consider change of a transition in our research for the minor changes.

Within the context in [Korel+02], regression testing of an added transition is equivalent to selective testing with respect to the added transition. As for testing the deletion of a transition in that context, it is possible to imitate traversal of the deleted transition when the input interaction associated with the deleted transition is generated and the enabling condition of the deleted transition is facilitated to evaluate to *TRUE*. As a result, regression testing of a deleted transition is equivalent to selective testing with respect to the deleted transition. In our research, we extend this context to include changed transitions and show that regression testing of a changed transition is equivalent to selective testing with respect to the changed transition.

In our approach, the selective testing strategy will be based on EFSM dependencies that were defined in the previous chapter and new dependencies that will be defined later in this chapter.

Changes in the requirements lead to modifications in the EFSM model representing the requirements. It is noted in [Korel+02] that when a model is modified, three types of model-based regression testing need to be performed:

1. Testing the effects of the model on the modification (modified part of the model),
2. Testing the effects of the modification on model (the remaining part of the model), and
3. Testing the side-effects of the modification on the unmodified parts of the model.

The effects of the model on the modification and the effects of the modification on the model indicate direct interactions between the modification and the EFSM model. The side-effects indicate interactions between parts of the EFSM model which are not modified. Therefore, the first two types of regression testing ensure that the changed parts

of the SUT behave as intended, and the third type of regression testing ensures that the unchanged parts of the SUT are not affected adversely.

In the following sections, we discuss the three types of regression testing for effects of the three types of EMs: adding, deleting, and changing a transition.

3. 1 Effects of the Addition of a Transition

In a modified EFSM model, addition of a transition t_i may

- introduce new DDs and/ or new CDs representing the effects of the model on t_i which are called *Affecting DD* and *Affecting CD*, respectively [Xie 05],
- introduce new DDs and/ or new CDs representing the effects of t_i on the model which are called *Affected DD* and *Affected CD*, respectively [Xie 05].

We observe that since t_i did not exist in the original EFSM model, there were neither existing CDs nor existing DDs involving the added transition that could be eliminated. Besides being directly involved in forming both Affecting DDs/ Affecting CDs and Affected DDs/ Affected CDs in a modified EFSM model, an added transition t_i may also have indirect effects on a modified EFSM model. That is, in a modified EFSM model, addition of a transition t_i may

- introduce new DDs between other transitions which are called *Activation Dependence* [Korel+02] and *Data Activation Dependence* [Tahat 07] ,
- introduce new CDs between other transitions which are called *Control Activation Dependence of Type A* [Tahat 07],
- eliminate existing CDs between other transitions which are called *Control Activation Dependence of Type D* [Tahat 07].

3.1.1 Effects of the Model on the Added Transition t_i

In the modified EFSM model, addition of a transition may introduce new DDs and/ or new CDs, which represent the effects of the EFSM model on the added transition. Although Korel et al. introduced affecting interaction patterns, they have done so without defining new dependence types [Korel+02]. In the Master's thesis of Bo Xie [Xie 05], these new dependence types were named *affecting data dependence*/ *affecting control dependence*, and were defined as follows: Let t_i be an added transition, and v be a

variable in the modified EFSM model,

- there exists an *affecting data dependence* (*Affecting DD*) from t_j to t_i w.r.t. v in the modified EFSM model iff there is a DD(t_j, t_i, v) in the modified EFSM model
- there exists an *affecting control dependence* (*Affecting CD*) from t_j to t_i in the modified EFSM model iff there is a CD(t_j, t_i) in the modified EFSM model.

In the modified SDG, the Affecting DD/ Affecting CD are marked as *new dependencies per modification* (NDPMs).

3.1.2 Effects of the Added Transition t_i on the Model

Addition of a transition also may introduce new DDs and/ or new CDs in the modified EFSM model, which represent the effects of the added transition on the EFSM model. Korel et al. introduced affected interaction pattern without defining new dependence types. In [Xie 05], these new dependence types were named *affected data dependence* and *affected control dependence*: Let t_i be an added transition, and v be a variable in the modified EFSM model, then

- there exists an *affected data dependence* (*Affected DD*) from t_i to t_j w.r.t. v in the modified EFSM model iff there is a DD(t_i, t_j, v) in the modified EFSM model
- there exists an *affected control dependence* (*Affected CD*) from t_i to t_j in the modified EFSM model iff there is a CD(t_i, t_j) in the modified EFSM model.

In the modified SDG, the Affected DDs/ Affected CDs are identified and marked as NDPM.

As it was shown above, an added transition is directly involved in both Affecting CD/ DD and Affected CD/ DD. We observe that since the added transition did not exist in the original EFSM model, there were neither existing CDs nor existing DDs involving the added transition that could be eliminated.

3.1.3 Side-effects of the Added Transition t_i

Generally, besides direct effects, an EM may have side-effects on a modified EFSM model. In the scope of this thesis, such side-effects include:

A) introduction of new DDs between other transitions,

B) introduction of new CDs between other transitions,

C) elimination of existing DDs between other transitions, and

D) elimination of existing CDs between other transitions.

In this section, Section 3.2.3 and Section 3.3.3, we will organize our discussions about side-effects according to these four possible side-effects.

Besides being directly involved in forming both Affecting DDs/ Affecting CDs and Affected DDs/ Affected CDs in a modified EFSM model, an added transition may also have indirect effects on a modified EFSM model. Such indirect effects are: causing introduction of new DDs or CDs between other transitions, and causing elimination of existing CDs between other transitions. These indirect effects are discussed below.

A) introduction of new DDs between other transitions

An added transition may introduce new DDs between other transitions, i.e., some new paths can be formed by the insertion of the added transition in the modified EFSM model, and these new paths may possibly be definition-clear paths for some variables.

Korel et al. named such new DDs activation [data] dependence in [Korel+02]. Tahat named them data activation dependence [Tahat 07]. Bo Xie used Korel et al.'s definition in [Xie 05]. Korel et al.'s definition is given as follows: there exists an activation dependence between t_i and the data dependence (t_j, t_k, v) w.r.t. v iff: (1) there exists a definition-clear path from t_j to t_i, (2) t_i is triggered, and (3) there exists a definition-clear path from t_i to t_k w.r.t. v. We observe that there is an omission in Korel et al.'s definition of activation dependence: t_i must not have a def of v in the modified EFSM model. Otherwise, there will not be a definition-clear path from t_j to t_k w.r.t. v through t_i.

B) introduction of new CDs between other transitions

In addition, neither Korel et al. nor Xie noticed that an added transition may also introduce new CDs between other transitions. Tahat observed that an addition of a transition may introduce in the modified model new CDs between other transitions. He referred to such new CDs as control activation dependence of type A (addition of new control dependencies). His definition is given as follows: there exists a control activation dependence of type A between t_j and t_k w.r.t. added transition t_i in the modified model iff: (1) there does not exist a CD(t_j, t_k) in the original EFSM model, (2) there exists a CD(t_j, t_k) in the modified EFSM model, and (3) there is no CD(t_j, t_k) in EFSM model M', where M' is the model resulting from deleting t_i from the modified EFSM model. We observe that

there are two shortcomings in Tahat's definition of control activation dependence: there was no detail given about how the addition contributes to the control activation dependence, and the definition does not allow multiple modifications contributing to one control activation dependence.

In the EFSM shown in Figure 3-1, before t_3 is added, there is only one path from s_1 to *Exit*, which goes through t_1 and t_2. s_2 post-dominates s_1 and s_2 post-dominates t_1. There is no CD from t_1 to t_2. After t_3 is added, s_2 does not post-dominate s_1, but s_2 still post-dominates t_1. Therefore, a CD(t_1, t_2) is introduced by the addition of transition t_3.

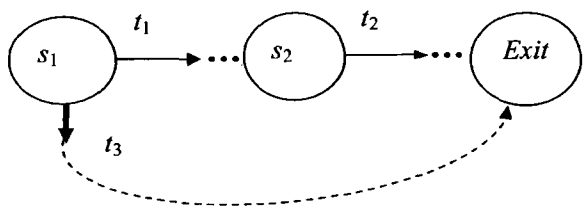


Figure 3-1. New CD introduced by added transition t_3

Hence, we correct their definition of activation [data] dependence and we refer to the new DDs introduced by an added transition as activation data dependence. In order to provide details for the CDs introduced by an added transition, we give our definition of what is defined as control activation dependence of type A by Tahat [Tahat 07] and we name it as activation control dependence. Our definitions are as follows: Let t_i be an added transition, and v be a variable in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then

- there exists an *activation data dependence* (*Activation DD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff (1) there does not exist a DD(t_j, t_k, v) in the original EFSM model, (2) there exists a definition-clear path from t_j to t_i w.r.t. v in the modified EFSM model, (3) there exists a definition-clear path from t_i to t_k w.r.t. v in the modified EFSM model, and (4) t_i does not have a def of v in the modified EFSM model.

- there exists an *activation control dependence* (*Activation CD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there does not exists a CD(t_j, t_k) in the original EFSM model, (2) there exists a CD(t_j, t_k) in the modified EFSM model, (3) if t_i is an added transition, i) t_i is in path(s) from s_1 to *Exit* without traversing s_2 in the modified EFSM model, when s_2 post-dominates s_1 in the original EFSM model; or ii) t_i is in path(s) from t_j to s_2 in the modified EFSM model, when there is no path from t_j to s_2 in the original EFSM model.

C) elimination of existing DDs between other transitions.

As for the elimination of the existing DDs, we agree with Korel et al. and Tahat that it is not possible for existing DDs between other transitions to be eliminated because of the addition of a transition. An added transition can only cause creation of new paths in a modified EFSM model. Thus, existing definition-clear paths and hence existing DDs between transitions won't be affected.

D) elimination of existing CDs between other transitions.

Korel et al. stated that "addition of a new transition cannot cause deletion of existing dependencies between parts of the model" in [Korel+02], while Bo Xie didn't discuss this issue in her thesis [Xie 05]. Tahat observed that an addition of a transition may cause 'removal' of some CDs that do exist in the original model [Tahat 07]. We agree with Tahat's statement. Added transitions can cause existing CDs between other transitions to be eliminated. In the EFSM shown in Figure 3-2, there are two paths from s_1 to *Exit*, one passes through states s_2 and s_3 (i.e., t_1, t_2 and t_3), the other one doesn't pass state s_3 (i.e., t_4). s_3 does not post-dominate s_1. Before t_5 is added, s_3 is on every path from s_1 to *Exit* through transition t_1, so s_3 post-dominates t_1. Therefore, there is a CD(t_1, t_3). But after adding t_5, s_3 does not post-dominate t_1 anymore, since s_3 is not on every path from s_1 to *Exit* through transition t_1. Thus, CD(t_1, t_3) is eliminated because of the new transition t_5.

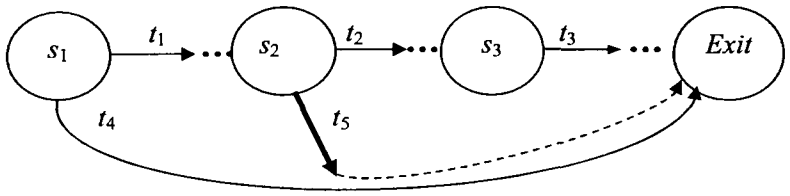


Figure 3-2. Existing CD deleted because of added transition t_5

Tahat referred to such CDs as *control activation dependence of type D* (deletion of control dependencies) and defined them as follows: there exists a control activation dependence of type D between t_j and t_k w.r.t. added transition t_i in the modified EFSM model iff: (1) there exists a CD(t_j, t_k) in the original EFSM model, (2) there does not exist a CD(t_j, t_k) in the modified EFSM model, and (3) there exists a CD(t_j, t_k) in M', where M' is the model resulting from deleting t_i from the modified EFSM model. We observe the same shortcomings in Tahat's definition of control activation dependence of type D as in his definition of control activation dependence of type A: there is no detail on how the addition contributes to the control activation dependence, and the definition does not allow multiple modifications contributing to one control activation dependence. We rename control activation dependence of type D as *activation ghost control dependence* and define it as follows: Let t_i be an added transition in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then

- there exists a *activation ghost control dependence* (*Activation GCD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there exists a CD(t_j, t_k) in the original EFSM model which ceases to exist in the modified EFSM model, (2) if t_i is an added transition, t_i is in path(s) from t_j to *Exit* without traversing s_2 in the modified EFSM model.

Based on the above discussions, we have shown that due to the addition of a transition t_i, new DDs or new CDs between other transitions may arise in the modified EFSM model. In addition, an existing DD(t_i, t_m) could be deleted because of addition of transition t_i.

Note that the EMs corresponding to addition of transition t_i referred by the above definitions are called *contributing modifications* of the Activation DD / CD / GCD.

In the modified *SDG*, all Activation DDs/ CDs and Activation GCDs are marked as NDPMs.

3.1.4 An Example for Testing the Addition of a Transition t_i

Consider the example EFSM shown in Figure 2-1. Suppose that a Balance Inquiry transaction has been added to the ATM system. This transaction is represented by transition $t_9 = <s_2, s_3, Balance, \varepsilon,$ Display(b), $\varepsilon >$ that has been added to the EFSM of

Figure 3-3-a. Corresponding changes in the SDG of the EFSM of the ATM system are shown in Figure 3-3-b.

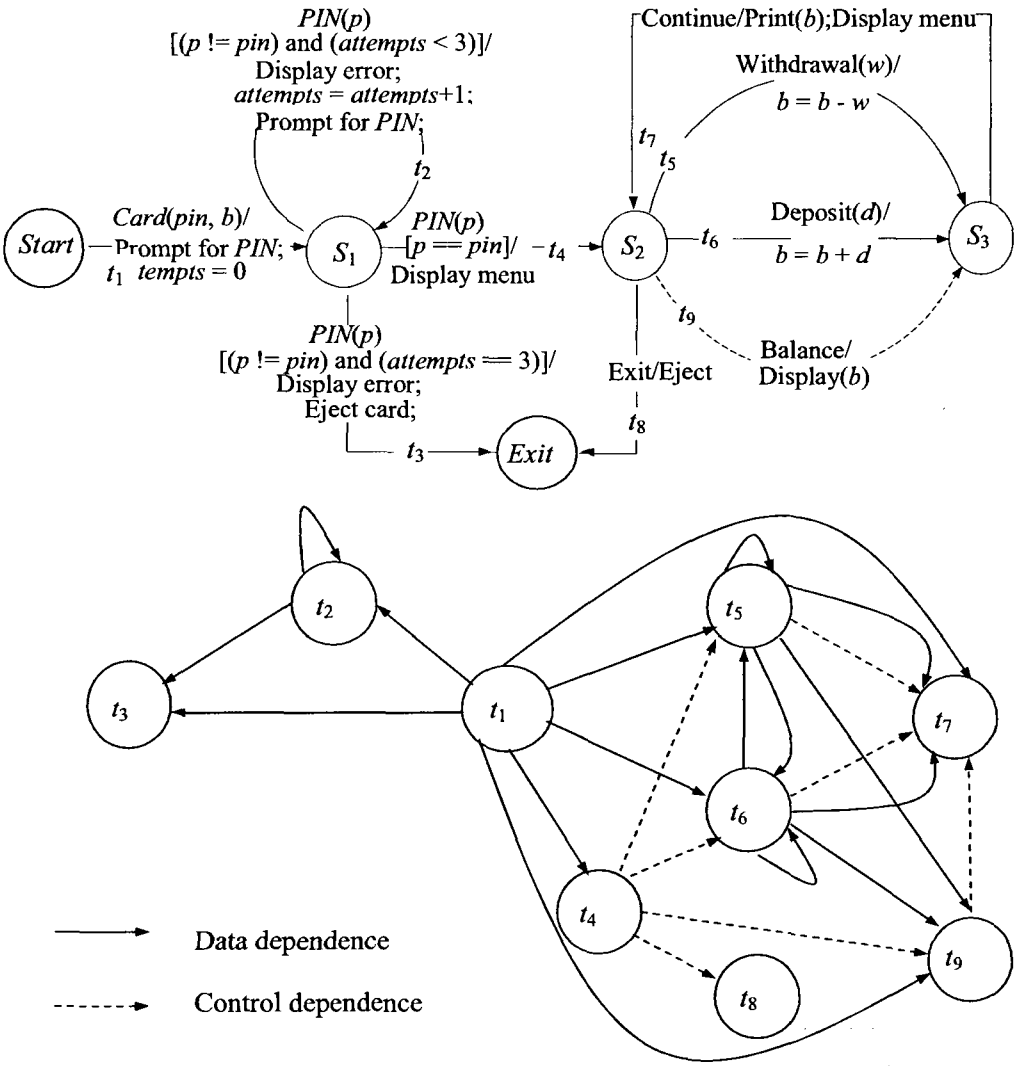


Figure 3-3. a) EFSM and b) SDG of the ATM system with balance transaction added

Note that the modified SDG differs from the original SDG w.r.t. the following DDs and CDs: (t_1, t_9, b), (t_1, t_7, b), (t_4, t_9), (t_5, t_9, b), (t_6, t_9, b), and (t_9, t_7). Among them, (t_4, t_9) is Affecting CD, (t_1, t_9, b), (t_5, t_9, b), (t_6, t_9, b) are Affecting DDs, (t_9, t_7) is Affected CD. There is one DD, (t_1, t_7, b), in which the added transition t_9 is not directly involved, but

arises in the modified EFSM model due to the introduction of t_9. This DD is Activation DD.

Table 3-1 below gives a summary of the effects of an added transition where *t* stands for "transition" and *ts* is an abbreviation of "transitions". The "Introduced" column lists whether a new dependence is formed due to the added transition. The "Eliminated" column shows if an existing dependence ceases to exist because of the added transition. "yes/ no" indicates the effects observed by Korel et al. and Xie. "YES/ NO" in upper case indicates the effects that are only observed by us. We follow the same notation in Table 3-2 and Table 3-3 as well.

Table 3-1. Effects of an added transition t_i

Type of Dependence		Introduced	Eliminated
DD	t_i depends on other *t*	yes	NO
CD	t_i depends on other *t*	yes	NO
DD	other *t* depends on t_i	yes	NO
CD	other *t* depends on t_i	yes	NO
DD	Between two other *ts*	yes	no
CD	Between two other *ts*	yes*	yes*

* Note that Korel and Xie did not observe the effects of an added transition that were observed by Tahat and us.

3.2 Effects of the Deletion of a Transition

As stated in Section 2.6, in the EFSM model, three elements are associated with each transition *t*: an input interaction $i(t)$, an enabling condition $c(t)$, and a sequence of actions. Since a deleted transition does not exist in a modified EFSM and cannot be used in real test cases, we adopt the scheme used by Korel et al. [Korel+02]: for each deleted transition t_i, a new transition td_{new}, with an empty sequence of actions, is added to the modified EFSM model at the starting state of t_i to represent the input interaction $i(t_i)$ and the enabling condition $c(t_i)$. That is, for each deleted transition t_i, there exists td_{new} in the modified EFSM model such that when the EFSM is at the starting state of t_i, event $i(t_i)$

occurs, and enabling condition $c(t_i)$ evaluates to *TRUE*, then td_{new} is triggered which brings the EFSM back to the starting state of t_i.

The inclusion of $i(t)$ and $c(t)$ in the modified model enforces the identification of dependencies involving the deleted transition. Keeping deleted transitions (dummy transitions in our case) in test cases facilitates identification of ghost dependencies covered by the test cases. Dummy transitions can also be used to identify obsolete test cases that are going to fail during testing.

More important, in regression testing, test cases containing deleted transitions (dummy transitions) can be used to ensure that the deleted transitions are indeed deleted. For the sake of safety, we would like to have this kind of test cases in an RTS. But without having dummy transitions in the modified models, test cases containing dummy transitions cannot be generated.

Therefore, we keep dummy transitions in the modified EFSM model for regression analysis. Dummy transitions can be easily removed after regression analysis is done to make a correct version of specification. They are not kept in the new version of the specification.

In a modified EFSM model, deletion of a transition t_i may

- eliminate existing DDs associated with t_i where t_i was dependent on another transition which are called *Affecting GDD* [Korel+02],
- eliminate existing CDs associated with t_i where t_i was dependent on another transition which we call *Affecting GCD*,
- eliminate existing DDs associated with t_i where some transitions were dependent on t_i which are called *Affected GDD* [Xie 05],
- eliminate existing CDs associated with t_i where some transitions were dependent on t_i which we call *Affected GCD*,
- introduce new CDs between other transitions which we called *Control Activation Dependence of Type A* [Tahat 07],
- eliminate existing DDs between other transitions which are called *Ghost Activation Dependence* [Xie 05] and *Data Activation Dependence* [Tahat 07],
- eliminate existing CDs between other transitions which are called *Control Activation Dependence of Type D* [Tahat 07].

3.2.1 Effects of the Model on the Deleted Transition t_i

Deletion of a transition may cause elimination of a DD associated with the deleted transition where the deleted transition was dependent on another transition. The eliminated DD is referred to as the *affecting ghost [data] dependence* in [Korel+02]. Korel et al. also gave the definition of *affecting ghost [data] dependence* in the same paper as follows: Let t_i be a deleted transition, and v be a variable in the original EFSM model, then

- there exists an *affecting ghost data dependence* (*Affecting GDD*) in the modified EFSM model from t_j to t_i iff: (1) there is a DD(t_j, t_i, v) (from t_j to t_i w.r.t. v) in the original EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v in the modified EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i in the modified EFSM model.

Bo Xie used this definition in her work [Xie 05]. This definition covers the case that a definition-clear path from t_j to the starting state of t_i w.r.t. v still exists in the EFSM model after deleting t_i. This may not always be the case since the definition-clear path from t_j to the starting state of t_i may not exist anymore, due to other modifications in the modified EFSM model. We observe that this case is covered by testing the side-effects of those particular modifications. Therefore, the definition of Affecting GDD does not need to cover this situation.

However, neither Korel et al. nor Xie noticed that a deleted transition may cause elimination of existing CDs. We introduce a new CD type as *affecting ghost control dependence* which arises due to the deletion of a transition. We define it as follows: Let t_i be a deleted transition, then

- there exists an *affecting ghost control dependence* (*Affecting GCD*) from t_j to t_i in the modified EFSM model iff: (1) there is a CD(t_j, t_i) (from t_j to t_i) in the original EFSM model, (2) the starting state of t_i does not post-dominate the starting state of t_j, and the starting state of t_i post-dominates t_j in the modified EFSM model, and (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i in the modified EFSM model.

If the starting state of t_i post-dominates the starting state of t_j, or the starting state of t_i does not post-dominate t_j in the modified EFSM model, there must be other modifications in the modified EFSM model which cause these changes. We observe that this situation is covered by testing the side-effects of those particular modifications. Therefore, the definition of Affecting GCD does not need to consider this situation.

In the modified SDG, all Affecting GDD/ Affecting GCD are marked as NDPMs.

3.2.2 Effects of the Deleted Transition t_i on the Model

Deletion of a transition may cause elimination of an existing DD associated with the deleted transition where some transition was dependent on the deleted transition. In [Korel+02] the authors only denoted an eliminated DD as an *affected ghost data dependence* which is defined by Xie [Xie 05] as follows: Suppose t_i is a deleted transition. There exists an *affected ghost data dependence* in the modified EFSM model from t_i to t_j iff: (1) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i, (2) suppose transition t_k is triggered (t_k must be a newly added transition, i.e., td_{new}), (3) there exists a definition-clear path from t_k to t_j w.r.t. v, and (4) t_k does not have a definition of v (a case when t_k has a definition of v is covered by testing the addition of t_k as this is a "new" dependence).

However, Xie's definition has a flaw. Suppose t_i is a deleted transition, and we add transition t_k with an empty sequence of actions to the modified model at the starting state of t_i as td_{new} (Figure 3-4). Since there was a DD(t_i, t_j, v) (from t_i to t_j w.r.t. v) in the original EFSM model, there was a definition-clear path w.r.t. v from the starting state of t_i to t_j through t_i. Therefore, without considering other modifications on the EFSM model, there is a definition-clear path w.r.t. v from the terminating state of t_i to t_j in the modified EFSM model. But from t_k to t_j, a definition-clear path w.r.t. v may not exist, because t_i has been deleted. This situation cannot be covered by testing side-effects of other modifications. Hence, the definition of *affected ghost data dependence* has to cover this situation.

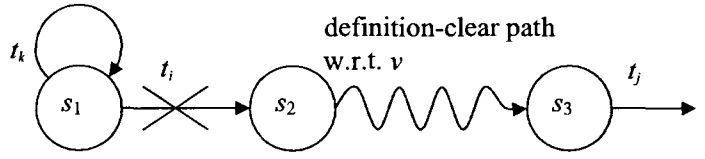


Figure 3-4. Effects of deleting transition t_i on (t_i, t_j, v)

Thus, we re-define *affected ghost data dependence* as follows: Let t_i be a deleted transition, v be a variable, and s be the terminating state of t_i in the original EFSM model, then

- there exists an *affected ghost data dependence* (*Affected GDD*) from t_i to t_j w.r.t. v in the modified EFSM model iff: (1) there exists a DD(t_i, t_j, v) in the original EFSM model, (2) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i, and (3) there exists a definition-clear path from s to t_j w.r.t. v in the modified EFSM model.

Next, In [Korel+02] the authors did not mention the potential for an existing CD to be eliminated. We introduce a new CD type to represent the case for elimination of an existing CD due to the deletion of a transition as *affected ghost control dependence*. Let t_i be a deleted transition, and s be the terminating state of t_i in the original EFSM model, then

- there exists an *affected ghost control dependence* (*Affected GCD*) from t_i to t_j in the modified EFSM model iff: (1) there exists a CD(t_i, t_j) in the original EFSM model, (2) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i, and (3) the starting state of t_j does not post-dominate the starting state of t_i and the starting state of t_j post-dominates s in the modified EFSM model.

Because there was a CD(t_i, t_j) in the original EFSM model, the starting state of t_j did not post-dominate the starting state of t_i, and the starting state of t_j post-dominated s. Deleting t_i cannot make the starting state of t_j to post-dominate the starting state of t_i, or the starting state of t_j not to post-dominate s. There must be other modifications in the modified EFSM model which cause these effects. This situation is covered by testing the side-effects of those particular modifications. Hence, the definition of Affected GCD does not need to cover this situation.

In the modified SDG, all the Affected GDD/ Affected GCD are marked as NDPMs.

3.2.3 Side-effects of the Deleted Transition t_i

A) introduction of new DDs between other transitions

In [Korel+02], the authors did not give a thorough discussion about the side-effects of a deleted transition. However, they stated that deletion of a transition could not introduce new DDs between other transitions. Tahat had the same observation in his thesis. We agree with this statement. Let's assume there is no definition-clear path from t_1 to t_2 w.r.t. v in the original EFSM model. That means from t_1 to t_2, there is no definition-clear path w.r.t. v originally. Deleting a transition t_3 can only make paths from t_1 to t_2 through t_3 infeasible. It won't affect other paths which do not pass t_3. Therefore, a definition-clear path w.r.t. v cannot be formed in the modified EFSM model, and a new DD cannot be introduced.

B) introduction of new CDs between other transitions

Tahat observed that deleting a transition might cause introduction of new CDs between other transitions. We agree with this observation. In Figure 3-5, t_1, t_4 are outgoing transitions from s_1, t_2, t_5 are outgoing transitions from s_2, and t_3 is an outgoing transition from s_3. Originally, there is a path from s_1 to *Exit* through t_1, t_2 and t_3, a path from s_1 to *Exit* through t_4 (doesn't pass through s_2 and s_3), and a path from s_2 to *Exit* through t_5 (doesn't pass through s_3). There is no CD(t_1, t_3) since s_3 doesn't post-dominate t_1. But after t_5 is deleted, s_3 post-dominates t_1. Therefore, a CD(t_1, t_3) is formed in the modified EFSM model.

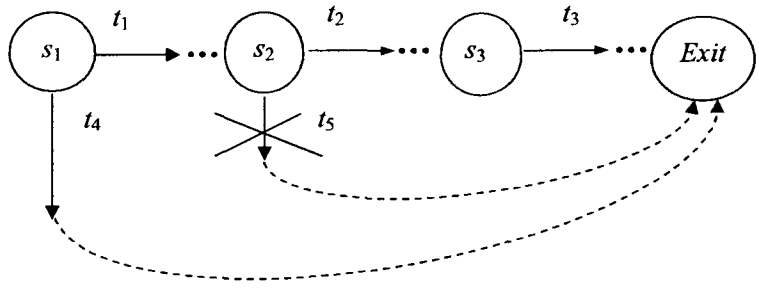


Figure 3-5. New CD introduced by deleted transition t_5

In order to capture the introduction of new CDs related to a deleted transition t_i, we need to revise the definition of Activation CD in Section 3.1.3 by adding point (4) at the end of the definition. The revised definition is as follows (changes are in bold): Let t_i be an added **or deleted** transition in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then

- there exists an *activation control dependence* (*Activation CD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there does not exists a CD(t_j, t_k) in the original EFSM model, (2) there exists a CD(t_j, t_k) in the modified EFSM model, (3) if t_i is an added transition, i) t_i is in path(s) from s_1 to *Exit* without traversing s_2 in the modified EFSM model, when s_2 post-dominates s_1 in the original EFSM model; or ii) t_i is in path(s) from t_j to s_2 in the modified EFSM model, when there is no path from t_j to s_2 in the original EFSM model, **and (4) if t_i is a deleted transition, t_i was in path(s) from t_j to *Exit* without traversing s_2 in the original EFSM model**.

C) elimination of existing DDs between other transitions

A deleted transition may cause existing paths to be eliminated, which in turn might lead to elimination of existing DDs between other transitions. This gives rise to another new DD type, which was defined as *ghost activation* [*data*] *dependence* by Korel et al. in [Korel+02], and used by Xie in her work [Xie 05]. Tahat used data activation dependence as the name of such DDs in his thesis. Their definition is as follows: Suppose transition t_i has been deleted from the original EFSM model. Then there exists td_{new} in the modified EFSM model, such that when the EFSM is at the starting state of t_i, event $i(t_i)$ occurs, and enabling condition $c(t_i)$ evaluates to *TRUE*, then td_{new} is triggered which brings the EFSM back to the starting state of t_i. Suppose also there exists a DD(t_j, t_k, v) (from t_j to t_k w.r.t. v) in the original EFSM model. Due to the deletion of transition t_i, DD(t_j, t_k, v) ceases to exist in the modified EFSM model. This dependence from t_i to the DD(t_j, t_k, v) is referred to as a *ghost activation dependence*. More formally: there exists a ghost activation dependence in the modified EFSM model from t_i to the DD(t_j, t_k, v) iff: (1) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v, (2) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to TRUE at the starting state of t_i, (3)

suppose transition t_n is triggered (t_n is a newly added transition i.e., td_{new}), and (4) there exists a definition-clear path from t_n to t_k w.r.t. v.

However, their definition has a flaw. Suppose t_i is a deleted transition and we add a self-loop transition td_{new} with an empty sequence of actions to the modified model at the starting state of t_i (Figure 3-6). Since deleting t_i causes the DD(t_j, t_k, v) to cease to exist, t_i must be in every definition-clear path from t_j to t_k w.r.t. v in the original EFSM model. Without considering other modifications on the EFSM model, there is a definition-clear path from t_j to the starting state of t_i w.r.t. v, and a definition-clear path from the terminating state of t_i to t_k w.r.t. v in the modified EFSM model. But from td_{new} to t_k, a definition-clear path w.r.t. v may not exist, because the terminating state of td_{new} is the starting state of t_i, and t_i has been deleted in the modified EFSM model. This case has to be covered by the definition.

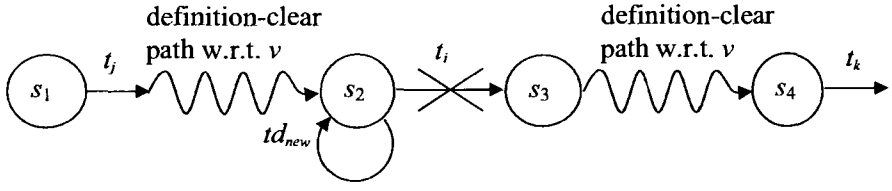


Figure 3-6. Effects of deleting transition t_i on DD(t_j, t_k, v)

We re-name the ghost activation dependence as *activation ghost data dependence* and redefine it as follows: Let t_i be a deleted transition, and v be a variable in the original EFSM model, then

- there exists an *activation ghost data dependence* (*Activation GDD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff: (1) there exists a DD(t_j, t_k, v) in the original EFSM model which ceases to exist in the modified EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v in the original EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to TRUE at the starting state of t_i, and (4) there exists a definition-clear path from the terminating state of t_i to t_k w.r.t. v in the original EFSM model.

D) elimination of existing CDs between other transitions.

Moreover, we observe that a deleted transition can also cause existing CDs between other transitions to be eliminated. Tahat made the same observation in [Tahat 07]. In the EFSM model shown in Figure 3-7, originally, s_2 does not post-dominate s_1 while s_2 post-dominates t_1. Therefore, there is a CD(t_1, t_2). But after t_3 is deleted, there is only one path from s_1 to *Exit*, which goes through t_1 and t_2. s_2 post-dominates s_1 and s_2 post-dominates t_1. There is no CD(t_1, t_2) any more.

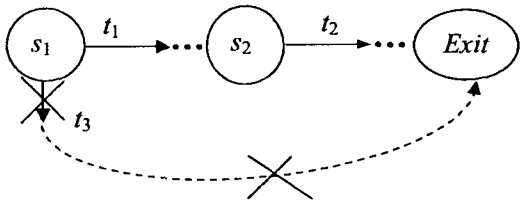


Figure 3-7. Existing CD deleted because of deleted transition t_3

In order to capture the elimination of existing CDs related to a deleted transition t_i, we need to revise the definition of Activation GCD in Section 3.1.3 by adding point (3) to the end of the definition. The revised definition is as follows (changes are in bold): Let t_i be an added **or deleted** transition in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then

- there exists a *activation ghost control dependence* (*Activation GCD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there exists a CD(t_j, t_k) in the original EFSM model which ceases to exist in the modified EFSM model, (2) if t_i is an added transition, t_i is in path(s) from t_j to *Exit* without traversing s_2 in the modified EFSM model, **and (3) if t_i is a deleted transition, t_i was in path(s) from s_1 to *Exit* without traversing s_2 in the original EFSM model**.

In the modified SDG, all Activation CD / Activation GCD are marked as NDPMs.

3.2.4 An Example for Testing the Deletion of Transition t_i

Consider the modified EFSM model shown in Figure 3-8-a). The Deposit transition is deleted from the modified EFSM model, resulting in the EFSM model shown in Figure 3-8-a) (represented by dashed arrow). Notice that transition t_{10} is added to the model to

indicate that the Deposit input is consumed in state S_2 and contains an empty sequence of actions, $td_{new} = t_{10}$. Figure 3-8-b) is the modified SDG, where t_6 is replaced by t_{10}. The modified SDG differs from the SDG in Figure 3-3-b) w.r.t. the following CDs and DDs: (t_1, t_6, b), (t_5, t_6, b), (t_6, t_6, b), (t_6, t_7), (t_6, t_5, b), (t_6, t_7, b). (t_4, t_6) is replaced by (t_4, t_{10}). Among them, (t_1, t_6, b), (t_5, t_6, b) are Affecting GDDs, (t_6, t_6, b), (t_6, t_5, b), (t_6, t_7, b) are Affected GDDs. (t_6, t_7) is Affected GCD. There is no side effect introduced by deleting t_6.

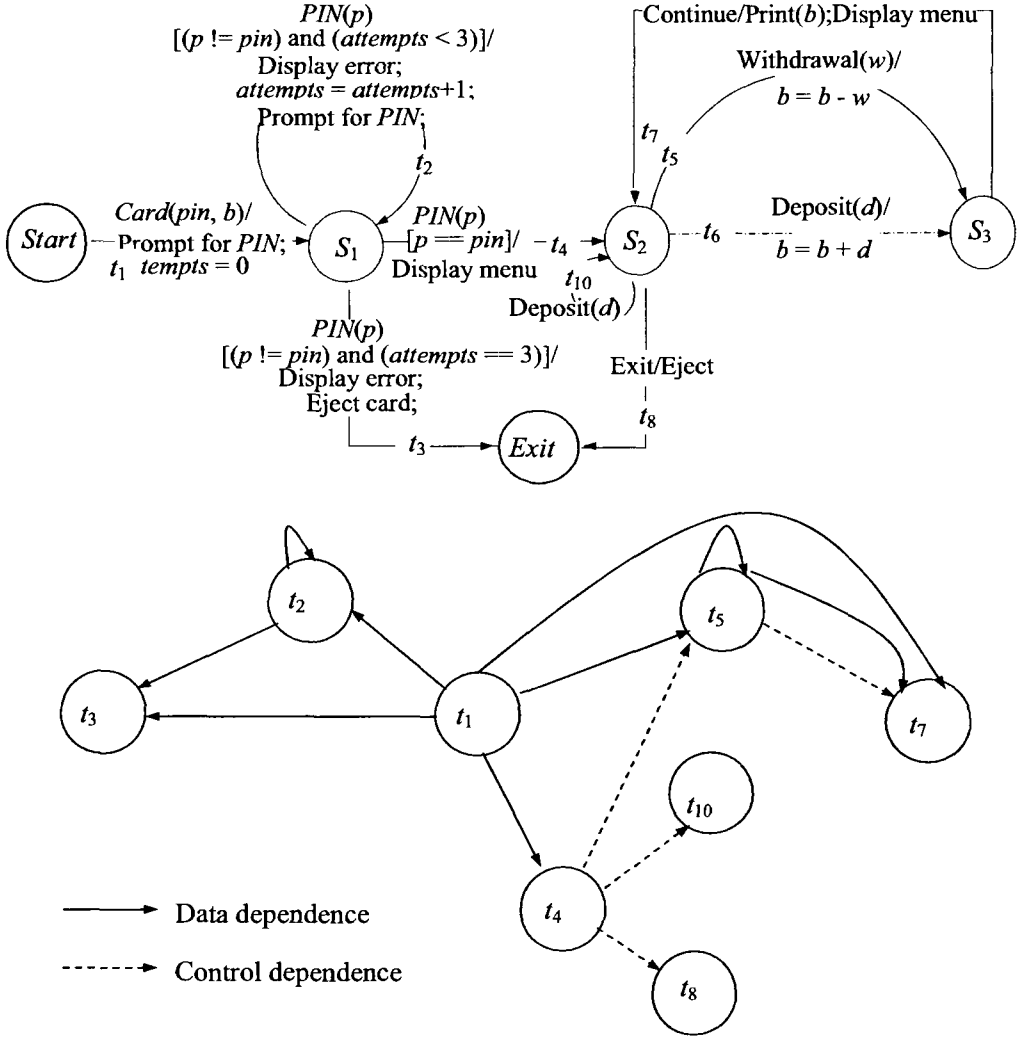


Figure 3-8. a) EFSM and b) SDG of the ATM system with deposit transaction deleted

Table 3-2 below gives a summary of the effects of a deleted transition.

Table 3-2. Effects of a deleted transition t_i

Type of Dependence		Introduced	Eliminated
DD	t_i depends on other t	no	yes
CD	t_i depends on other t	no	YES
DD	other t depends on t_i	no	yes
CD	other t depends on t_i	no	YES
DD	between two other ts	no	yes
CD	between two other ts	yes*	yes*

* Note that Korel and Xie did not observe the effects of a deleted transition that were observed by Tahat and us.

3.3 Effects of a Changed Transition

In our research, we consider change of a transition to be one of the types of EMs to save the cost of expressing each minor change as a pair of deletion and addition. By using Change as a new type of EM, we save the subsequent costs related to expressing each minor change as a pair of deletion and addition. These costs are firstly, the cost of analyzing all EMs, which is reduced because fewer EMs need to be analyzed. Secondly, fewer NDPMs and IPs are identified. Thus, the sizes of reduced/generated test suites are less, resulting in lower costs of maintenance, test suite execution, and test result analysis. For a transition t represented as a 6-tuple $<s_s, s_t, i, c, o, a>$, *minor change* refers to changes on i, c, o, or a of t. If starting state or terminating state of a transition is changed, we will consider this change as a major change instead of a minor change, and will express it as a pair of deletion and addition. Major changes in the transitions are not referred to as "changed" transitions. It was assumed in Section 2.6 that for each transition, there is at most one EM in the set of EMs. Hence, when a change on a transition is expressed as a pair of deletion and addition, the added transition will be assigned a unique transition ID.

Since we only deal with minor changes on a transition, a changed transition will not introduce new paths, or cause existing paths to be deleted in a modified EFSM model. Therefore, it cannot introduce new CDs in which the changed transition is involved, nor cause elimination of existing CDs associated with the changed transition. Neither can it introduce new CDs between other transitions, nor cause existing CDs between other transitions to be eliminated.

On the other hand, a changed transition may cause introduction of new DDs, or cause existing DDs to be eliminated because minor changes to a transition may alter defs or uses of variables. Examples of changes in different elements of a transition t_i are as follows:

- Change on the input of t_i: from Card(PIN) to Card(PIN, b).
- Change on the condition of t_i: from $a < b$ to $a < b + 1$.
- Change on the output of t_i: from Print(b) to Print($b/2$).
- Change on the action of t_i: from $n = m+r$ to $n = i\text{-}m+r$.

Based on the definition of EFSM, possible definitions and uses of a variable v in a transition can occur in the input of t_i: def(v), in the condition of t_i: use(v), In the output of t_i: use(v), and in the action of t_i: def(v) and use(v). Possible minor changes to a transition may introduce a new def(v), eliminate an existing def(v), revise an existing def(v), introduce a new use(v), and eliminate an existing use(v). Revision of an existing use(v) is not possible because when an expression with a use(v) is changed, it is the expression itself to be revised, not the use(v). If this expression is used to define a variable w in actions a of t_i, then the def(w) is revised and thus, there is an effect on DDs. On the other hand, if this expression is used in an enabling condition c or in an output o, then c or o is revised. In this case, changes to the transition t_i do not affect any du-pair and thus, there is no effect on DDs.

In a modified EFSM model, changes in a transition t_i may

- introduce new DDs representing the effects of the model on t_i which we call *Affecting DD*,
- eliminate existing DDs associated with t_i where t_i was dependent on another transition which we call *Affecting GDD*,
- introduce new DDs representing the effects of t_i on the model which we call

Affected DD,

- eliminate existing DDs associated with t_i where some transitions were dependent on t_i which we call *Affected GDD*,
- introduce new DDs between other transitions which we call *Activation DD*,
- eliminate existing DDs between other transitions which we call *Activation GDD*.

3.3.1 Effects of the Model on the Changed Transition t_i

Introduction of new DDs or elimination of existing DDs associated with a changed transition t_i w.r.t. the effects of a modified EFSM model on t_i are captured by revisions of definitions of Affecting DD and Affecting GDD (given in Sections 3.1.1 and 3.2.1, respectively). Recall that, for an added transition, a new DD which represents an effect of the model on the added transition was referred to as Affecting DD in Xie's work [Xie 05] (see Section 3.1.1). In order to capture the introduction of new DDs associated with a changed transition, we need to revise Xie's definition to include changed transitions, and mark the changes in bold as follows: Let t_i be an added **or a changed** transition, and v be a variable in the modified EFSM model, then

- there exists an *affecting data dependence* (***Affecting DD***) from t_j to t_i w.r.t. v in the modified EFSM model iff there is a DD(t_j, t_i, v) in the modified EFSM model, **and if t_i is a changed transition, changes in t_i include addition of a use of v (before v is possibly defined at t_i), or deletion of a def of v before uses of v at t_i.**

Recall also that, for a deleted transition, a DD associated with the deleted transition where the deleted transition was dependent on another transition was referred to as *affecting ghost data dependence* in [Korel+02] (see Section 3.2.1). We modify their definition to include changed transitions, and mark the changes in bold as follows: Let t_i be a deleted **or a changed** transition, and v be a variable in the original EFSM model, then

- there exists an *affecting ghost data dependence* (*Affecting GDD*) from t_j to t_i in the modified EFSM model iff: (1) there is a DD(t_j, t_i, v) (from t_j to t_i w.r.t. v) in the original EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v in the modified EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i in the

modified EFSM model, **and (4) when t_i is a changed transition, t_i does not have a use of v (before v is possibly defined at t_i) in the modified EFSM model.**

In the modified SDG, all Affecting DDs / Affecting GDDs are marked as NDPMs.

3.3.2 Effects of the Changed Transition t_i on the Model

A changed transition may introduce new DDs or may cause elimination of DDs in which the changed transition depends on other transitions. Both cases represent the effects of the changed transition on the original EFSM model. Similar to what we have done in the previous section, we modify the definitions of Affected DD and Affected GDD (see sections 3.1.2 and 3.2.2) to cover the effects of the changed transitions on the EFSM model, and mark the changes in bold as follows:

Let t_i be an added **or a changed** transition, and v be a variable in the modified EFSM model, then

- there exists an *affected data dependence* (***Affected DD***) from t_i to t_j w.r.t. v in the modified EFSM model iff there is a DD(t_i, t_j, v) in the modified EFSM model, **and if t_i is a changed transition, changes in t_i includes addition or revision of a statement in which the last def of v appears.**

Let t_i be a deleted **or a changed** transition, v be a variable, and s be the terminating state of t_i in the original EFSM model, then

- there exists an *affected ghost data dependence* (***Affected GDD***) from t_i to t_j w.r.t. v in the modified EFSM model iff: (1) there exists a DD(t_i, t_j, v) in the original EFSM model, (2) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i, (3) there exists a definition-clear path from s to t_j w.r.t. v in the modified EFSM model, **and (4) if t_i is a changed transition, t_j has a def of v in the original EFSM model, but does not have a def of v in the modified EFSM model**

In the modified SDG, all Affected DDs / Affected GDDs are marked as NDPMs.

3.3.3 Side-effects Introduced by the Changed Transition t_i

We have pointed out previously that a changed transition cannot introduce new CDs between other transitions, or cause existing CDs between other transitions to be

eliminated. Therefore, it is not possible for a changed transition to have side-effects B) and D). But a changed transition may affect DDs between other transitions.

A) introduction of new DDs between other transitions,

A changed transition may introduce new DDs between other transitions. For example, suppose there were a definition-clear path from t_j to t_i w.r.t. v and a definition-clear path from t_i to t_k w.r.t. v, and v was defined in the original EFSM model. If changes on t_i include deleting all defs of v, new definition-clear paths from t_j to t_k w.r.t. v may be introduced. In this case, an Activation DD may be introduced by the changed transition. Thus, we revise the definition of Activation DD and mark the changes in bold as follows: Let t_i be an added **or a changed** transition, and v be a variable in the modified EFSM model, then

- there exists an *activation data dependence* (*Activation DD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff (1) there does not exist a DD (t_j, t_k, v) in the original EFSM model, (2) there exists a definition-clear path from t_j to t_i w.r.t. v in the modified EFSM model, (3) there exists a definition-clear path from t_i to t_k w.r.t. v in the modified EFSM model, and (4) t_i does not have a def of v in the modified EFSM model, **and (5) if t_i is a changed transition, t_i has a def of v in the original EFSM model**

C) elimination of existing DDs between other transitions

Also, a changed transition may cause an existing DD between other transitions to be eliminated. If changes on transition t_i include adding def of a variable v, an existing definition-clear path from t_j to t_k w.r.t. v that passes through t_i will be eliminated. If this path is the only definition-clear path from t_j to t_k w.r.t. v, existing DD(t_j, t_k, v) is eliminated by the changed transition. In this case Activation GDDs may be introduced. Thus, we revise the definition of Activation GDD and mark the changes in bold as follows: Let t_i be a deleted **or a changed** transition, and v be a variable in the original EFSM model, then

- there exists an *activation ghost data dependence* (*Activation GDD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff: (1) there exists a DD(t_j, t_k, v) in the original EFSM model which ceases to exist in the modified EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v in the

original EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to TRUE at the starting state of t_i, (4) there exists a definition-clear path from the terminating state of t_i to t_k w.r.t. v in the original EFSM model, **and (5) if t_i is a changed transition, t_i does not have a def of v in the original EFSM model, but has a def of v in the modified EFSM model.**

In the modified SDG, all Activation DDs / Activation GDDs are marked as NDPMs.

3.3.4 An Example for Testing Changed Transition t_i

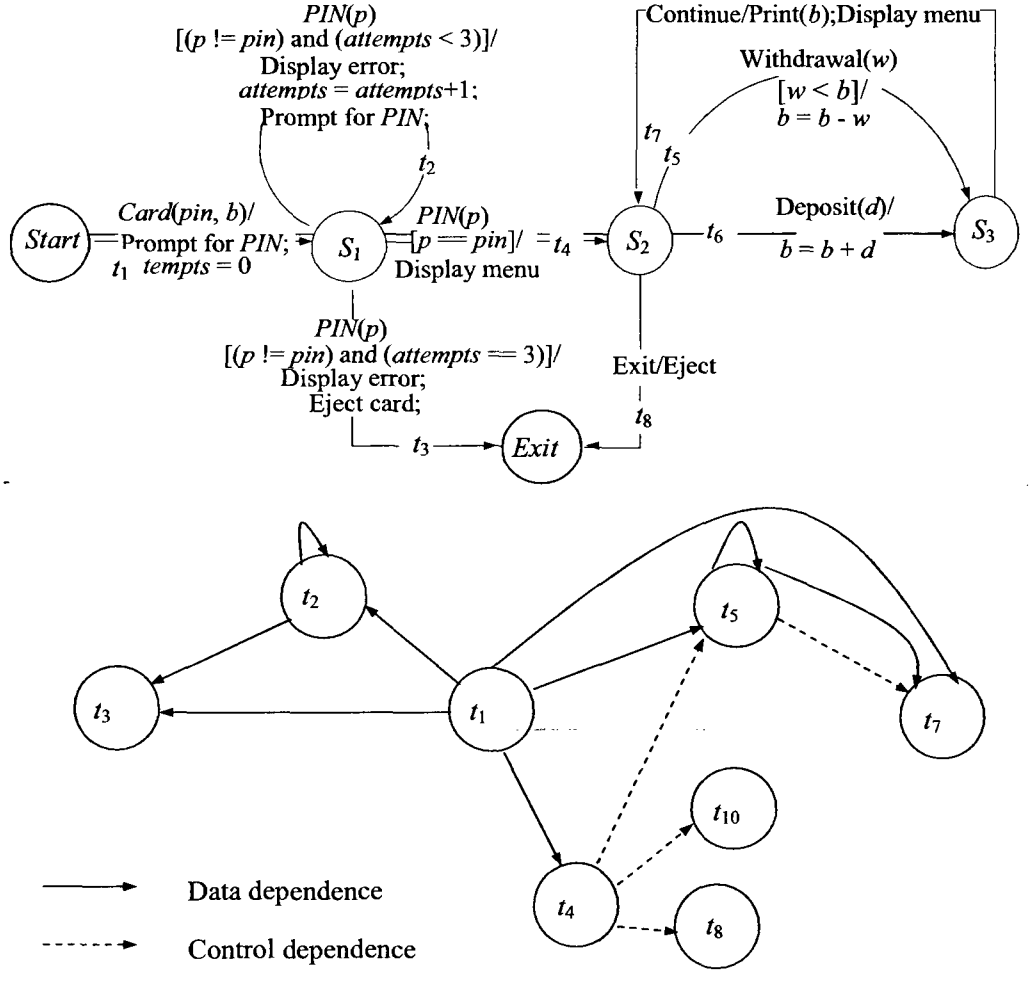


Figure 3-9. a) EFSM and b) SDG of the ATM system with changes on transaction t_5

For the example EFSM shown in Figure 3-9-a, suppose that transition t_5 is modified to $<s_2, s_3, Withdrawal(w), w < b, , b = b - w>$. Condition "$w < b$" is added to make sure that the amount withdrawn does not exceed the account balance. A use of w and a use of b are added. The SDG is not changed by this modification, which is shown in Figure 3-9-b. In this case, there is no side-effect. There is also no Affected DD. (t_1, t_5, b), (t_5, t_5, b), (t_6, t_5, b) are Affecting DDs.

Table 3-3 summarized the effects of a changed transition.

Table 3-3. Effects of a changed transition t_i

Type of Dependence		Introduced	Eliminated
DD	t_i depends on other t	YES	YES
CD	t_i depends on other t	NO	NO
DD	other t depends on t_i	YES	YES
CD	other t depends on t_i	NO	NO
DD	between two other ts	YES	YES
CD	between two other ts	NO	NO

3.4 Summary of Effects of Modifications

So far, for each EM type, we have examined all three types of regression testing that need to be performed. All three possible types of dependencies have been considered, namely:

i) new dependencies that are introduced in the modified model,

ii) dependencies that exist in both the original model and the modified model, and

iii) dependencies that disappear from the modified model.

Both data dependence and control dependence are examined. Our definitions of Activation DD/GDD/CD/GCD take into account multiple modifications contributing to the Activation DD/GDD/CD/GCD. Hence, within the scope of this thesis, our work is now complete as far as definitions of NDPMs are concerned.

In Section 3.1, 3.2, and 3.3, we presented the following twelve different dependencies stemming from by addition, deletion, or change of a transition in the EFSM model. These dependencies belong to three types of regression testing, containing four dependencies

each. Since some definitions were introduced and later modified during the discussion, we would like to summarize them in Appendix A.

Table 3-4. New dependencies raised by addition/ deletion/ change of a transition

EMs		**Effects of the model on modified transition**	**Effects of modified transition on the model**	**Side-effects of modified transition**
Addition	Possible Impact	*Affecting DD* *Affecting CD* (Section 3.1.1)	*Affected DD* *Affected CD* (Section 3.1.2)	*Activation DD* *Activation CD* *Activation GCD* (Section 3.1.3)
	Impossible Impact	*Affecting GDD* *Affecting GCD* (Section 3.1.1)	*Affected GDD* *Affected GCD* (Section 3.1.2)	*Activation GDD* (Section 3.1.3)
Deletion	Possible Impact	*Affecting GDD* *Affecting GCD* (Section 3.2.1)	*Affected GDD* *Affected GCD* (Section 3.2.2)	*Activation CD* *Activation GDD* *Activation GCD* (Section 3.2.3)
	Impossible Impact	*Affecting DD* *Affecting CD* (Section 3.2.1)	*Affected DD* *Affected CD* (Section 3.2.2)	*Activation DD* (Section 3.2.3)
Change	Possible Impact	*Affecting DD* *Affecting GDD* (Section 3.3.1)	*Affected DD* *Affected GDD* (Section 3.3.2)	*Activation DD* *Activation GDD* (Section 3.3.3)
	Impossible Impact	*Affecting CD* *Affecting GCD* (Section 3.3.1)	*Affected CD* *Affected GCD* (Section 3.3.2)	*Activation CD* *Activation GCD* (Section 3.3.3)

Table 3-4 summarizes our discussions about dependencies resulting from modifications on the EFSM model. In this table, we categorize NDPMs by types of

regression testing and types of EMs. Cells of this table list the absence and presence of possible impacts and give section numbers where the impacts were discussed.

Bo Xie presented an algorithm for generating SDG for the modified EFSM model and algorithms for the dependencies introduced by EMs and used in her thesis [Xie 05]. We need to slightly change Xie's algorithm for generating SDG for the modified EFSM model, since besides addition and deletion, we take minor change as one of the types of EMs in our research. Also, we need to add algorithms for the new dependencies introduced by us. Detailed SDG generation algorithm is presented in Appendix B.

3.5 Chapter Summary

In this chapter, we discussed effects of three types of EMs in the EFSM model. We characterized twelve types of new dependencies per modification (NDPMs) to capture the effects of the model on the EMs, the effects of the EMs on the model, and the side-effects caused by the EMs. An algorithm to generate SDG for the modified EFSM model which identifies NDPMs introduced by each of the three types of EMs is presented in Appendix B.

4. Regression Test Suite Generation

In Chapter 3, we presented twelve different dependencies, namely *new dependencies per modification* (NDPMs), arising from additions, deletions, or changes of transitions in the EFSM model. These NDPMs belong to three types of regression testing, containing four dependencies each. In summary, the NDPMs to test the effects of the model on the modified transition (Affecting DD, Affecting CD, Affecting GDD, and Affecting GCD), and the NDPMs to test the effects of the modified transition on the model (Affected DD, Affected CD, Affected GDD, and Affected GCD) capture interactions between model elements that affect or are affected by the EMs, respectively. Therefore, the NDPMs related to these two types of regression testing test the direct effects of the EMs on the changed part of the SUT. The NDPMs to test the side-effects introduced by the modified transition (Activation DD, Activation CD, Activation GDD, and Activation GCD) capture interactions that stem from the side-effects introduced by the EMs. Hence, the NDPMs related to this type of regression testing focuses on the possible unintentional effects of the EMs on the unchanged part of the SUT.

Clearly, an RTS must contain test cases to test both direct and indirect effects of the modifications on the EFSM model. Such effects can be captured by the occurrences of NDPMs related to the three types of regression testing discussed in the previous chapter.

In this chapter, we propose a regression test suite generation approach (RTSG method) to generate regression test cases and construct RTSs that cover all occurrences of NDPMs and EMs.

4.1 Activating Path and Complete Test Case Design

In our RTSG method, we are looking for a minimal set of complete test cases to test all occurrences of NDPMs. A complete test case is a sequence of transitions which starts at the initial state, and ends at the exit state. We use a sequence of transitions (test sequence) to represent a test case.

4.1.1 Activating Path for NDPMs

In our research, a path (a sequence of transitions) that covers an occurrence of a CD/ DD

is named as *activating path* (AP) of the CD/ DD.

For a CD(t_i, t_j), the starting state of t_j is on every path from t_i to *Exit*. Hence, any path from t_i to the starting state of t_j with t_j appended covers an occurrence of CD(t_i, t_j) and is an AP for the CD. For a DD(t_i, t_j, v), we need to find a definition-clear path from t_i to t_j w.r.t. v which is an AP for the DD.

Depending on the types of NDPMs and EMs, we get APs to cover the NDPMs based on rules listed below. In our discussion, *OM* stands for the original EFSM model, and *MM* stands for the modified EFSM model. ss_{ti} is the starting state of t_i, and st_{ti} is the terminating state of t_i. Case 1, 2, and 3 for Activation CD or Activation GCD refer to different types of contributions of an EM specified in Activation CD or Activation GCD's definitions.

1. Activating paths for NDPMs associated with added transition t_i for
   - Affecting_DD(t_j, t_i, v): a definition-clear path from t_j to t_i w.r.t. v in *MM*
   - Affecting_CD(t_j, t_i): a path from t_j to t_i in *MM*
   - Affected_DD(t_i, t_j, v): a definition-clear path from t_i to t_j w.r.t. v in *MM*
   - Affected_CD(t_i, t_j): a path from t_i to t_j in *MM*
   - Activation_DD t_i --> DD(t_j, t_k, v): a definition-clear path from t_j to t_k through t_i w.r.t. v in *MM*
   - Activation_CD t_i --> CD(t_j, t_k):
     - AP to traverse the CD(t_j, t_k): a path from t_j to t_k in *MM*
     - AP to test Case 1 (t_i is in path(s) from the starting state of t_j to *Exit* without traversing the starting state of t_k in *MM*): a path from the starting state of t_j to *Exit* through t_i in *MM*.
     - AP to test Case 2 (there is no path from t_j to the starting state of t_k in *OM*, and t_i is in path(s) from t_j to the starting state of t_k in *MM*): a path from t_j to t_k through t_i in *MM*
   - Activation_GCD --> CD(t_j, t_k)
     - AP to traverse the CD(t_j, t_k): a path from t_j to t_k in *OM* (replace any deleted transition in the path by its td_{new}. If a replacement happens, we expect this test case to fail, and there must exist Activation_CD t_m --> CD(t_j, t_k), where t_m is a deleted transition)

- AP to test Case 1 (t_i is in path(s) from t_j to *Exit* without traversing the starting state of t_k in *MM*): a path from t_j to *Exit* through t_i in *MM*

2. Activating paths for NDPMs associated with deleted transition t_i for

As we have discussed in Chapter 3, a deleted transition t_i is replaced by td_i in *MM*, while $td_i = < ss_{ti}, st_{ti}, i_{ti}, c_{ti}, \varepsilon, \varepsilon >$.

- Affecting_GDD(t_j, t_i, v): a definition-clear path from t_j to ss_{ti} w.r.t. v in *MM* appended with td_i.
- Affecting_GCD(t_j, t_i): a path from t_j to ss_{ti} in *MM* appended with td_i.
- Affected_GDD(t_i, t_j, v): td_i appended with a definition-clear path from st_{ti} to t_j w.r.t. v in *MM*.
- Affected_GCD(t_i, t_j): td_i appended with a path from st_{ti} to t_j in *MM*.
- Activation_GDD t_i --> DD(t_j, t_k, v): a definition-clear path from t_j to t_k through t_i w.r.t. v in *OM*, with t_i and any other deleted transitions replaced by their td_{new}.
- Activation_CD t_i --> CD(t_j, t_k)
  - AP to traverse the CD(t_j, t_k): a path from t_j to t_k in *MM*
  - AP to test Case 3 (t_i is in path(s) from t_j to *Exit* without traversing the starting state of t_k in *OM*): a path from t_j to Exit through t_i in *OM*, with t_i replaced by td_i.
- Activation_GCD t_i --> CD(t_j, t_k)
  - AP to traverse the CD(t_j, t_k): a path from t_j to t_k in *OM*, with any deleted transition in the path replaced by td_i
  - AP to test Case 2 (t_i is in path(s) from the starting state of t_j to *Exit* without traversing the starting state of t_k in *OM*): a path from the starting state of t_j to *Exit* through t_i in *OM*, with t_i replaced by td_i.

3. Activating paths for NDPMs associated with changed transition t_i for

- Affecting_DD(t_j, t_i, v): a definition-clear path from t_j to t_i w.r.t. v in *MM*
- Affecting_GDD(t_j, t_i, v): a definition-clear path from t_j to ss_{ti} w.r.t. v in *MM* appended with t_i.
- Affected_DD(t_i, t_j, v): a definition-clear path from t_i to t_j w.r.t. v in *MM*
- Affected_GDD(t_i, t_j, v): t_i appended with a definition-clear path from st_{ti} to t_j w.r.t. v in *MM*.

- Activation_DD t_i --> DD(t_j, t_k, v): a definition-clear path from t_j to t_k through t_i w.r.t. v in *MM*
- Activation_GDD t_i --> DD(t_j, t_k, v): a definition-clear path from t_j to t_k through t_i w.r.t. v in *OM*.

We notice that, a CD / DD may not have an AP because of path predicates. For example, for DD(t_1, t_3, *attempts*) in the simplified ATM model shown in Figure 2-1, there is only one definition-clear path from t_1 to t_3 w.r.t. *attempts*: t_1, t_3. But, this path is infeasible, because t_3's enabling condition *attempts* == 3 cannot be satisfied by the def of *attempts* in t_1. In this case, DD(t_1, t_3, *attempts*) doesn't have a feasible AP. In our research, if a CD / DD has a feasible AP, then this CD / DD is *applicable*.

4.1.2 Complete Test Case Design

In our approach, a regression test case is a complete test case. Suppose we want to test CD(t_4, t_5) in the example ATM model. As shown in the previous section, an AP for a CD(t_i, t_j) or DD(t_i, t_j, v) starts with t_i, and ends with either t_j or *Exit* state. AP for CD(t_4, t_5) starts with t_4, and ends with t_5. Besides the AP, t_4 needs a preamble and t_5 needs a postamble to form a complete test case. A preamble for t_i, $PR(t_i)$, is a path that starts at *Start* of the EFSM model and ends at the starting state of t_i. A postamble for t_j, $PT(t_j)$, is a path that starts at the terminating state of t_j and ends at *Exit* of the EFSM model. A complete test case is formed by concatenation of a $PR(t_i)$, an AP for the NDPM under test, and a $PT(t_j)$ if the AP ends with t_j. Deleted transitions in generated complete test cases are replaced by corresponding td_{new}.

When finding an AP, a preamble and postamble, we aim to find a shortest executable path for each of them, and which does not contain any predicate. If we fail to find such a path, then we choose a shortest path and try eventually to make it executable. Many existing algorithms, such as Bellman-Ford algorithm [Weisstein] and Dijkstra's algorithm [Dijkstra 59] to solve shortest path problem, can help solve our problem.

But this will not ensure that the resulting complete test case is executable. If this is the case, then alternative shortest paths should be found. It must be noted that determining the executability of a complete test case is in general undecidable. Fortunately, executability of a complete test case can be determined in most practical

EFSMs due to the absence of non-linear inequalities representing the path predicates [Bourhfir+97]. There are general approaches such as constraint solving and theorem proving that can assist solving this problem [Hierons+08].

Besides finding an executable complete test case, we also need to determine the expected results of the complete test case. For example, should we expect the test case to pass or fail? What should we observe during the test execution? These are critical information for testers to determine if the SUT behaves as expected.

- For test cases designed to test NDPMs associated with an added transition, we expect them to pass
- For test cases designed to test NDPMs associated with a deleted transition t_i
  - If $ss_{ti} \neq st_{ti}$, we expect the test cases to fail.
  - If $ss_{ti} = st_{ti}$,
    - If there is an output in t_i, the test cases will pass, but the output will be absent. We need to attach a note to the test cases that says "Observe the absence of outputs".
    - If there is not an output in t_i, then the test cases will pass. We need to attach a note to the test cases that says "Observe the absence of effect of actions".
- For test cases designed to test NDPMs associated with a changed transition, we expect them to pass.

4.2 Regression Test Suite Construction

Often, a specification-based test suite is constructed from an EFSM model to fulfill some pre-defined coverage criterion such as a control-flow or data-flow coverage criterion, such as all-nodes criterion, all-edges criterion, and all-uses criterion [Rapps+85]. Quite a number of EFSM-based test generation methods have been proposed in the literature. For instance, [Bourhfir+97] presented an algorithm named EFTG to generate executable test cases for EFSM models using UIO (unique input output) sequence and all-uses criteria. [Sarikaya+86], [Ural+91], [Huang+95] use data flow testing techniques in test case design for EFSM specifications.

However, an RTS may not need to target the same coverage as an original test suite for two reasons. First, only part of the SUT will be tested by the RTS. Regression test

cases are designed to test the SUT to verify that modifications have not caused unintended effects and that the SUT complies with the changes in the requirements. Secondly, frequent regression testing during software maintenance suggests that test cost may be reduced by minimizing the size of the RTS.

In Chapter 3, we have defined twelve NDPMs for EMs on the EFSM model. Since the set of EMs is finite and the number of dependencies that a transition introduces is bounded, the number of NDPMs is bounded in an EFSM model. Our research is based on dependence analysis. So it is logical to use dependence coverage as a criterion for the RTSG method. Because of the limitation on size of RTS, the test coverage goal we set for the generated RTS in this thesis is to cover each NDPM and EM at least once. Our RTSG method is as follows:

Given an original EFSM model *OM* and a set of EMs *M*, the proposed RTS generation method has the following four steps:

Step (1) applies *M* to *OM* to get the modified EFSM model *MM*.

Step (2) identifies a set of NDPMs by considering each EM in *M* and using *OM* and *MM* according to the definitions of NDPMs.

Step (3) generates a set of complete test cases (sequences of transitions which start at *Start*, and end at *Exit* of the EFSM) to cover each NDPM and each EM in *M* at least once.

Step (4) uses the probability-driven greedy algorithm (refer to Section 2.9.2) to select a minimum set of complete test cases from the generated test cases that cover all NDPMs and EMs in *M*.

The first step of the RTSR method is to apply *M* to *OM* to derive the *MM*. NDPM identification is done after *M* has been applied. Therefore, all modifications have been made to *OM* before analysis begins.

Notice that in Step (3), we want the generated test cases cover not only all NDPMs, but also all EMs in *M*. We have observed that, when changes to a transition do not involve any du-pair, there may not be any DD affected by the changed transition. Moreover, added or deleted transitions may not affect any CD. Therefore, it is not necessary to have an NDPM associated with an EM. For example, adding t_3 in Figure 4-1 does not affect any CDs. If there is neither def nor use w.r.t. any variable in t_3, DDs are

not affected either. Therefore, there is not a NDPM associated with the addition of t_3.

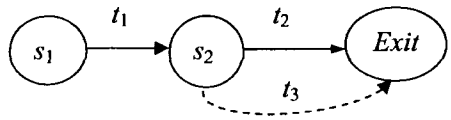


Figure 4-1. Added Transition t_3 without affecting CDs

But, every EM in a given set of EMs should be tested by an RTS. In this case, any executable complete test case that traverses a transition corresponding to an EM can be a regression test case for this EM. After generating complete test cases to cover all NDPMs, our method scans the set of EMs, and generates complete test cases for any untested EMs.

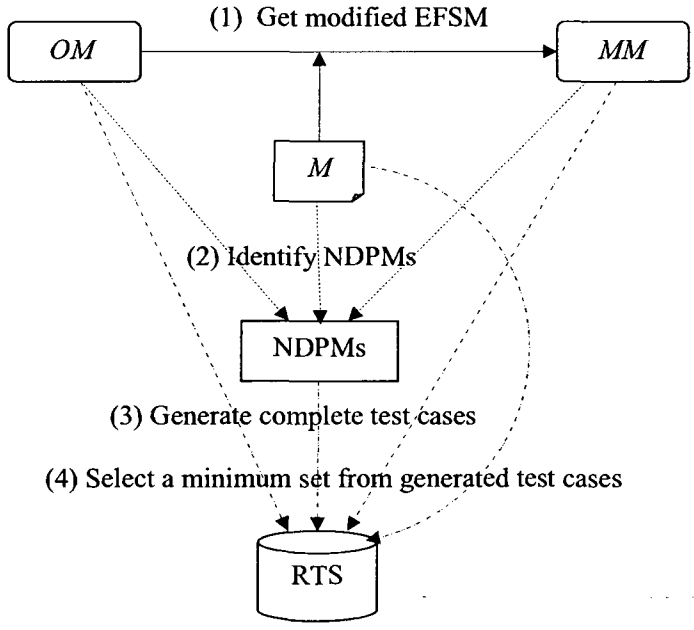


Figure 4-2. RTSG method

Figure 4-2 shows the four steps in the RTSG method. Step (1) of the proposed method is a straight-forward process to get *MM*, and Step (2) implements our discussions in Chapter 3. Step (3) constructs complete test cases to test all identified NDPMs and EMs based on discussion in Section 4.1.2. However, each test case generated in Step 3

can cover multiple NDPMs and EMs. Some test cases may be redundant. We need to select a minimum set of test cases that cover all NDPMs and EMs, which is an SCP problem. In Step (4) we apply the probability-driven greedy algorithm to optimize results of the RTSG method. In the next section, we will discuss how to adopt the probability-driven greedy algorithm into the RTSG method.

We have the following theorems for the RTSG method.

THEOREM 1. In the RTSG method, the order of applying the set of EMs to the original EFSM model does not affect the set of identified NDPMs.

PROOF. Step (1) of the RTSG method applies the set of EMs *M* to *OM* to get the modified EFSM model *MM*. Notice that *MM* is the final version of the EFSM model with all EMs in *M* applied to *OM*. We are not interested in any "middle version" during the process. *OM* and *MM* are compared to identify NDPMs in Step (2). But during this process, *M* is not applied to *OM* again. Therefore, the order of applying EMs to *OM* doesn't make any difference to the set of identified NDPMs.

THEOREM 2. In the RTSG method, the order of applying the set of EMs to the original EFSM model does not affect the generated RTS.

PROOF. By Theorem 1, the order of applying the set of EMs *M* to *OM* doesn't make any difference in set of identified NDPMs. In Step (3) and (4), the identified NDPMs and *M* are used to construct an RTS. But during this process, *M* is not applied to *OM* again. Hence, the order of applying EMs to *OM* doesn't make any difference in final generated RTS.

4.3 Adopting Probability-driven Greedy Algorithm into the RTSG Method

Step (3) of the RTSG method constructs a set of complete test cases to that cover all NDPMs and EMs. In order to gain efficiency, when a complete test case is constructed for an NDPM under test, we identify all NDPMs covered by this test case. These NDPMs will not be considered in the process any more. However, all NDPMs covered by a test case *ts* may be covered by test cases generated later. In this situation, *ts* is redundant, and should not be included in the generated RTS. Hence, after a set of complete test cases that cover all the NDPMs and EMs has been generated, we need to select a minimum subset from them that cover all the NDPMs and EMs, which is an SCP problem.

Step (4) of the RTSG method adopts the probability-driven greedy algorithm introduced in Section 2.9.2 as a solution. During complete test case construction in Step (3) of the RTSG method, relationships between test case ts_p and its coverage of NDPMs and EMs can be tracked in a table *TB* as follows:

For each identified NDPM and each EM in *M*, add a row into *TB* to represent it

For a newly constructed complete test case ts_p

1). add a column for ts_p in *TB*

2). for each NDPM *ndpm*

if *ndpm* is covered by ts_p then assign Value 1 to cell (*ndpm*, ts_p)

else assign Value 0 to cell (*ndpm*, ts_p)

3). for each EM *em*

if *em* is covered by ts_p then assign Value 1 to cell (*em*, ts_p)

else assign Value 0 to cell (*em*, ts_p)

Table 4-1 shows an example of *TB* constructed in Step (3).

Table 4-1. Test cases vs. covered NDPMs / EMs

NDPM / EM	ts_1	ts_2	ts_3	...	ts_n
$NDPM_1$	1	1	0	...	1
$NDPM_2$	0	1	0	...	0
...	...	...	...	...	...
$NDPM_m$	1	0	1	...	0
EM_1	1	1	0	...	1
EM_2	0	1	0	...	0
...	...	...	...	...	...
EM_l	1	0	1	...	0

Rampone's algorithm only required basic computation. Considering its good performance, in Step (4) of our RTSG method, we adapt Rampone's probability-driven greedy algorithm to select a minimum set of test cases from the generated test cases that cover all NDPMs / EMs.

In our case, the universe U is the set of NDPMs / EMs to be tested. Each test case ts constructed in Step (3) covers several NDPMs / EMs, which is a subset A_i of U. Therefore, the set of all generated complete test cases is the family A of subset of U.

Refer to formula (ii) in Section 2.9.2, in the RTSG method, U_j is an NDPM or EM under test, which is represented by a row of table TB. m is the number of identified NDPMs plus the number of EMs in M, which equals to the number of rows of table TB. A_i is a generated test case, which is represented by a column of table TB. n is the number of all generated test cases, which equals to the number of columns of table TB.

Test case selection algorithm to be used in Step (4), and the detailed algorithm for the RTSG method is presented in Appendix C.

4.3 Example

As we presented in Chapter 3, EMs in the example EFSM are adding t_9, deleting t_6, and changing t_5. Note that t_{10} is the td_{new} for t_6.

The shortest executable preamble and postamble for each transition in T_m are listed in Table 4-2.

Table 4-2. Shortest executable preamble and postamble

Transition t_i	**Preamble $PR(t_i)$**	**Postamble $PT(t_i)$**
t_1	ε	t_4, t_8
t_2	t_1	t_4, t_8
t_3	t_1, t_2, t_2, t_2	ε
t_4	t_1	t_8
t_5	t_1, t_4	t_7, t_8
t_7	(1) t_1, t_4, t_5 (2) t_1, t_4, t_6 (3) t_1, t_4, t_9	t_8
t_8	t_1, t_4	ε
t_9	t_1, t_4	t_7, t_8
t_{10}	t_1, t_4	t_8

Accordingly, all NDPMs in the modified EFSM are listed in Table 4-3 as follows:

Table 4-3. NDPMs for the modified ATM system

NDPM#	Dependence	EMs
a	(t_1, t_5, b)	change t_5
b	(t_1, t_7, b)	delete t_6
c	(t_1, t_9, b)	add t_9
d	(t_1, t_{10}, b)	delete t_6
e	(t_5, t_5, b)	change t_5
f	(t_5, t_9, b)	add t_9
g	(t_{10}, t_5, b)	change t_5
h	(t_{10}, t_7, b)	delete t_6
i	(t_{10}, t_{10}, b)	delete t_6
j	(t_4, t_9)	add t_9
k	(t_4, t_{10})	delete t_6
l	(t_9, t_7)	add t_9
m	(t_{10}, t_7)	delete t_6

Applying our proposed method, a generated RTS is as shown in Table 4-4.

Table 4-4. RTS for the modified ATM system

TC#	Complete Test Cases	Covered NDPMs
1	t_1, t_4, t_9, t_7, t_8	*b, c, m, k*
2	t_1, t_4, t_5, t_7, t_9, t_7, t_8	*a, e, f, k, l*
3	t_1, t_4, t_{10}, t_5, t_7, t_8	*d, g, j*
4	t_1, t_4, t_{10}, t_9, t_7, t_8	*c, d, h, j, k, l, m*
5	t_1, t_4, t_{10}, t_{10}, t_8	*d, i, j*

Thus, 5 regression test cases are sufficient to cover all NDPMs for the example EFSM.

Note that many different possible RTSs could be generated because of different orders of applying NDPMs to the algorithm. This is indeed expected because there is not a unique set of complete test cases to satisfy a given control or data flow oriented test coverage criterion. It should also be noted that the probability-driven greedy algorithm may yield different reduced RTSs depending on the order of feeding the generated complete test cases.

Our regression test suite is designed to test EMs and associated NDPMs. The test suite we design for added/ deleted/ changed transitions may not satisfy the original coverage criteria of designing the original test suite.

4.4 Chapter Summary

In this chapter, we focused on RTS generation. We first defined activating paths (APs) for the twelve types of NDPMs. Then we discussed how to construct complete test cases using the APs. Also, we proposed an RTS generation (RTSG) method. Moreover, we showed how to adopt the probability-driven greedy algorithm in our RTSG method. Finally, we gave an example for using our method to generate an RTS.

5. Regression Test Suite Reduction

In this Chapter, we propose a method for reducing the size of a given RTS according to a given set of EMs on an EFSM model. Korel et al. presented an approach in [Korel+02], to use interaction patterns to reduce the size of a given RTS. In this paper, the authors presented a requirement-based regression test suite reduction approach that used EFSM model dependence analysis to reduce a given RTS. The modifications on an original EFSM model are classified as a set of EMs: addition of a transition and deletion of a transition. For each EM, data and control dependencies are used to capture potential interactions between EFSM transitions, and three interaction patterns are computed: (1) affecting interaction pattern, (2) affected interaction pattern, (3) side-effect interaction pattern. Later, Xie has formalized their approach, defined sub-types of some of the dependencies introduced by Korel et al., and proposed algorithms to exploit these dependencies to reduce a given RTS [Xie 05].

Our method is an extension of the work of Korel et al. [Korel+02] and Xie [Xie 05] for test suite reduction. Our work differs from the work of Korel et al. [Korel+02] and Xie [Xie 05] in terms of its scope and coverage. Firstly, as we have pointed out in Chapter 2, their approach considers only two types of EMs, i.e., an addition or deletion of a transition. Instead of expressing minor changes in a transition as a pair of deletion and addition, we consider such changes as another EM type, i.e., change in a transition. Secondly, although Korel et al. and Xie defined some data and control dependencies arising from each EM on the EFSM model, they have not identified some dependence types. Instead, we adopt the notions introduced in their work, utilize some of their definitions for control and data dependencies, revise some of their definitions for control and data dependencies, and define new dependencies that have not been identified by them. Twelve NDPMs have been defined in Chapter 3. Using these revised and new definitions of NDPMs, we will refine the definitions of interaction patterns.

In RTS reduction, we are not interested in executability of test cases in a given RTS. We assume that executability of test cases is ensured by test case designers of the given RTS. Hence, in this chapter, we do not discuss this issue.

In the following sections, we first discuss interaction patterns, and then propose a regression test suite reduction method (RTSR method) to reduce the size of a given RTS, while maintaining the level of coverage with respect to a given set of EMs.

5.1 Interaction Patterns

When a model is modified, three types of model-based regression testing need to be performed [Korel+02]: 1) Testing the effects of the model on the modification, 2) Testing the effects of the modification on the model, and 3) Testing the side-effects of the modification on the unmodified parts of the model. For these three types of regression testing, [Korel+02] introduced three types of interaction patterns related to each modification: 1) an affecting interaction pattern, 2) an affected interaction pattern, and 3) a side-effect interaction pattern.

The goal of these three types of regression testing is to test the different interactions or interaction patterns between functional elements of the model with respect to a transition that represents an elementary modification. Within software, "pattern" most often refers to the key aspects of a common design structure. But in our discussion, the term "pattern" in "interaction pattern" refers to its generic definition, which is "*an abstraction from a concrete form that recurs in specific non-arbitrary contexts*", as defined by *IEEE* Computer Society. In our research, "*interaction pattern*" (IP) is defined as *an abstraction from interactions that recur between functional elements of the model with respect to a transition that represents an elementary modification.*

Since there are three types of regression testing, [Korel+02] introduced three types of IPs related to each EM:

1. An affecting interaction pattern (Affecting IP),
2. An affected interaction pattern (Affected IP), and
3. A side-effect interaction pattern (Side-effect IP)

The Affecting IP captures interactions between model elements that affect the modification. The Affected IP captures interactions between model elements that are affected by the modification. Finally, the Side-effect IP captures interactions between model elements that occur because of indirect effects introduced by the modification.

When an original EFSM model is modified, the original SDG changes accordingly. Given a test case, the three IPs can be computed: the Affecting IP, the Affected IP, and the Side-effect IP. In the following subsections, we discuss how to compute the three IPs for effects of the three types of EMs: adding, deleting, and changing a transition.

5.1.1 Interaction Patterns for Added Transition t_i

For each added transition t_i, up to three IPs can be constructed.

The *affecting interaction pattern (Affecting IP)* for a given test case related to the added transition t_i is constructed as follows:

Procedure P1 (Modified SDG *G*, test case *ts*, Affecting *IP*)

1. Mark dependencies encountered in *G* during the traversal of *ts*.
2. Form *G'* by eliminating the unmarked dependencies in *G*.
3. Identify and mark the Affecting DDs / Affecting CDs in *G'*.
4. Traverse backwards from the added transition t_i through the marked Affecting DDs / Affecting CDs and mark those Affecting DDs / Affecting CDs that are traversed. From these traversed Affecting DDs / Affecting CDs, traverse backwards and mark dependencies in *G'* that "affect" the added transition t_i
5. Remove all unmarked dependencies from *G'*.
6. Return *G'* as the resulting *affecting IP*.

The *affected interaction pattern (Affected IP)* for a given test case related to the added transition t_i is constructed as follows:

Procedure P2 (Modified SDG *G*, test case *ts*, Affected *IP*)

1. Mark dependencies encountered in *G* during the traversal of *ts*.
2. Form *G'* by eliminating the unmarked dependencies in *G*.
3. Identify and mark the Affected DDs / Affected CDs in *G'*.
4. Traverse forward from the added transition t_i through the marked Affected DDs / Affected CDs and mark those Affected DDs / Affected CDs that are traversed. From these traversed Affected DDs / Affected CDs, traverse forward and mark dependencies in *G'* that are "affected" by the added transition t_i.
5. Remove all unmarked dependencies from *G'*.
6. Return *G'* as the resulting *affected IP*.

The *side-effect interaction pattern (Side-effect IP)* for a given test case related to the added transition t_i is constructed as follows:

Procedure P3 (Modified SDG G, test case *ts*, Side-effect *IP*)

1. Mark dependencies encountered in G during the traversal of *ts*.
2. Form G' by eliminating the unmarked dependencies in G.
3. Identify and mark the Activation DDs / Activation CDs / Activation GCDs in G'.
4. Traverse forward from the added transition t_i through the marked Activation DDs / Activation CDs / Activation GCDs and mark those Activation DDs / Activation CDs / Activation GCDs that are traversed. From these traversed Activation DDs / Activation CDs / Activation GCDs, traverse forward and mark dependencies in G' that are "affected" by the added transition t_i.
5. Remove all unmarked dependencies from G'.
6. Return G' as the resulting *side-effect IP*.

5.1.2 Interaction Patterns for Deleted Transition t_i

Three types of IPs are defined for any deleted transition. The procedures for constructing an *affecting, affected* and *side-effect interaction patterns* for a given test case related to the deleted transition t_i are the same as the procedures $P1$, $P2$, $P3$ given in Section 5.1.1 except that

- "Affecting DDs / Affecting CDs" in $P1$ are replaced by "Affecting GDDs / Affecting GCDs"
- "Affected DDs / Affected CDs" in $P2$ are replaced by "Affected GDDs / Affected GCDs"
- "Activation DDs / Activation CDs / Activation GCDs" in $P3$ are replaced by "Activation CDs / Activation GDDs / Activation GCDs", respectively.

5.1.3 Interaction Patterns for Changed Transition t_i

Similar to added transitions and deleted transitions, three types of IPs can be calculated for any changed transition. The procedures for constructing *affecting, affected* and *side-effect interaction patterns* for a given test case related to the changed transition t_i are the same as the procedures $P1$, $P2$, $P3$ given in Section 5.1.1 except that

- "Affecting DDs / Affecting CDs" in $P1$ are replaced by "Affecting DDs / Affecting GDDs",
- "Affected DDs / Affected CDs" in $P2$ are replaced by "Affected DDs / Affected GDDs",
- "Activation DDs / Activation CDs / Activation GCDs" in $P3$ are replaced by "Activation DDs / Activation GDDs".

5.1.4 An Example for Interaction Patterns

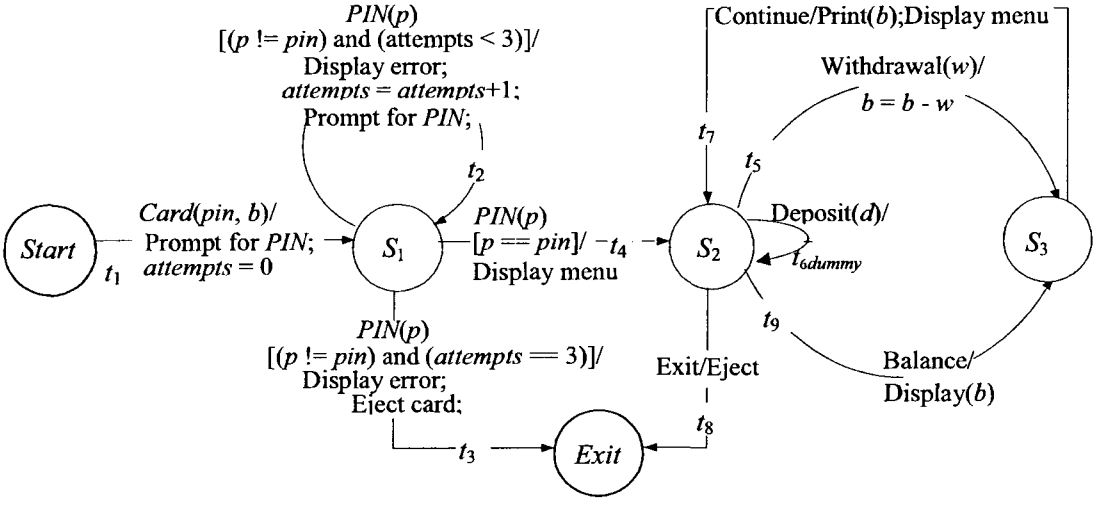
We use the simplified ATM EFSM model shown in Figure 2-1 as an example. Suppose the set of EMs to be applied on the EFSM model is $M = \{m_1, m_2, m_3\}$

m_1: add t_9: <S_2, S_3, *Balance*,ε, Display(b), ε>

m_2: delete t_6

m_3: change t_5 to: <S_2, S_3, *Withdrawal*(w), $w < b$, ε, $b = b - w$>.

Notice that transition t_{6dummy} is added to the model as td_{new} for t_6 to indicate that the Deposit input is consumed in state S_2 and contains an empty sequence of actions. The modified EFSM model is shown in Figure 5-1-a). Corresponding changes in the SDG of the EFSM of the ATM system are shown in Figure 5-1-b. There are fifteen NDPMs identified in the SDG after M applied.



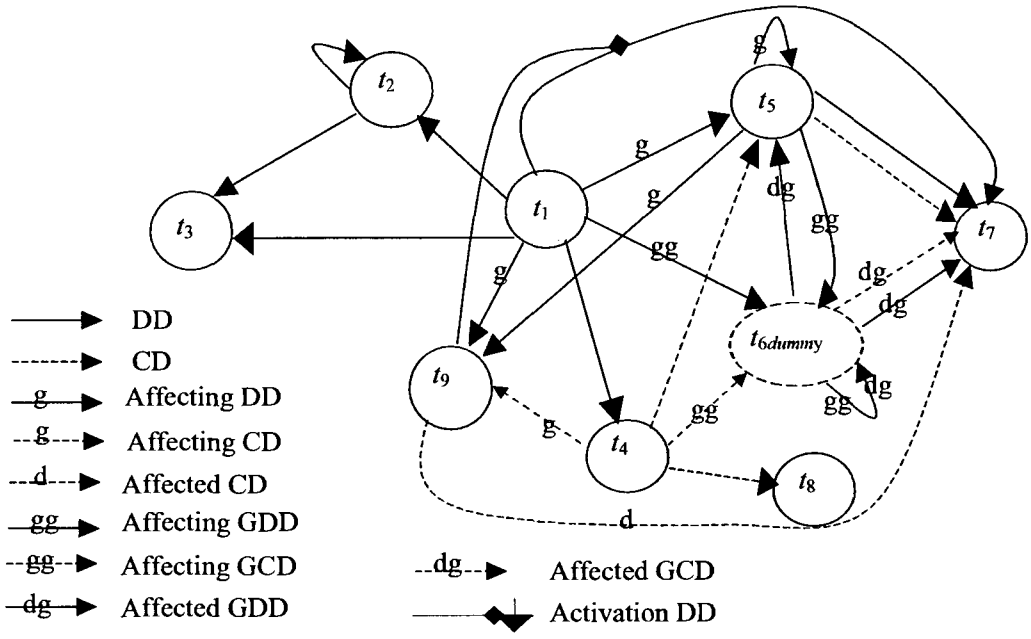


Figure 5-1. a) EFSM and b) SDG of the modified ATM system

Given a regression test case t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8, the sub-SDG covered by the test case is shown in Figure 5-2.

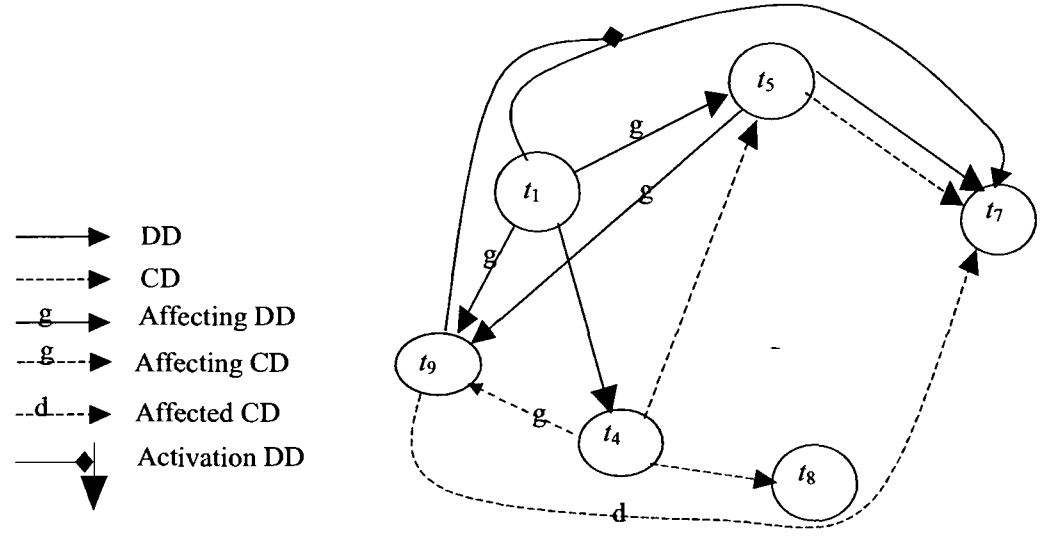


Figure 5-2. Sub-SDG for test case t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8

Using Procedure P1, P2, and P3, we compute up to three IPs for each EM in *M*. For example, to compute Affecting IP for the test case t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8 w.r.t. m_1 (adding t_9), we first traverse backwards from t_9 through the marked Affecting DDs / Affecting CDs in the sub-SDG. (t_1, t_9, *b*), (t_5, t_9, *b*), and (t_4, t_9) are traversed and marked. Then we traverse backwards from (t_1, t_9, *b*), (t_5, t_9, *b*), and (t_4, t_9) in the sub-SDG. All dependencies in the sub-SDG are considered now. As a result, (t_1, t_4, *pin*), (t_1, t_5, *b*), and (t_4, t_5) are traversed and marked. These six dependencies are included in the Affecting IP, since they "affect" t_9.

IPs w.r.t. m_1 and m_3 are shown in Figure 5-3 and Figure 5-4, respectively. For m_3, only Affecting IP is generated. There is no Affected IP and Side-effect IP. m_2 is not covered by this test case. Therefore, there is not IP w.r.t. m_2.

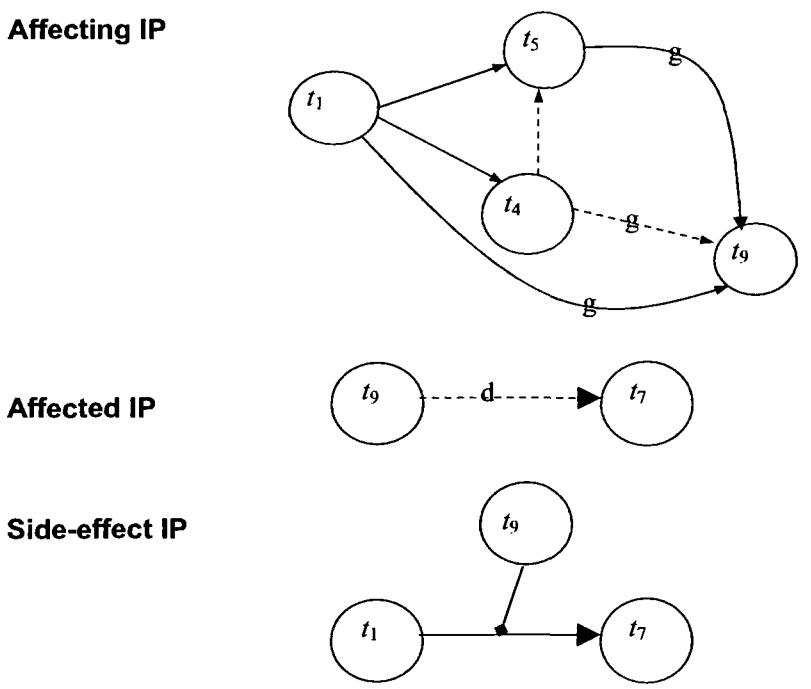


Figure 5-3. IPs for test case t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8 w.r.t. m_1

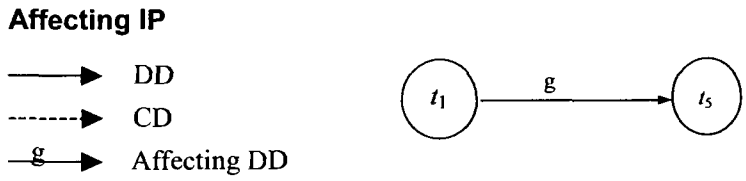


Figure 5-4. IP for test case t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8 w.r.t. m_3

Detailed algorithm to construct IPs for a given test case can be found in Appendix D.

5.2 Regression Test Suite Reduction Method

In [Korel+02], for each given test case, during the traversal of the test case within the modified SDG, up to 3**N* IPs are computed, where *N* is the number of EMs. Two test cases are considered to be "equivalent" w.r.t. a transition corresponding to an EM if, during traversal of the EFSM model, these tests exhibit the same IP of certain type w.r.t. the transition. Only those test cases that at least one of its 3**N* IPs does not exist for any other test cases in the reduced test suite are included in the reduced RTS. Bo Xie formalized Korel's work in her thesis in [Xie 05] and provided an implementation.

Our research is an extension of Korel et al. and Bo Xie's work for test suite reduction purpose. In previous chapters, we have introduced new dependencies caused by EMs. Also, we have introduced a new EM type Change. Similar to Korel et al. and Bo Xie's work, a test sequence in the given RTS can traverse within the modified SDG to construct up to 3**N* IPs. Our goal is to reduce a given RTS by identifying equivalent test cases w.r.t. the transition under test and removing these test cases from the selective test suite. Only those test cases that at least one of its 3**N* IPs does not exist for any other test cases are included in the reduced RTS.

For example, a given RTS for the modified ATM model shown in Figure 5-1 contains the following two test cases to test m_1, m_2, m_3 in the example we used in Section 5.1.4:

Test #1: t_1, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8 — Valid PIN entered, check the balance then perform a withdrawal transition. Check the balance again at the end.

Test #2: t_1, t_2, t_4, t_9, t_7, t_5, t_7, t_9, t_7, t_8 — Invalid PIN entered, followed by a valid PIN entered, check the balance then perform a withdrawal transition. Check the balance again at the end.

For these two test cases, the IPs w.r.t. m_1 and m_3 are the same. Both test cases do not cover m_2. Therefore, Test #1 and Test #2 are considered as equivalent test cases. At most one of them would be kept in the reduced RTS.

We have observed that, it is not necessary to have an IP calculated for every EM, even if the corresponding transition of an EM is traversed by test cases in the given RTS. There might be two reasons: i) as we have discussed in Section 4.2, there may not be any NDPM associated with the EM; and ii) test cases in the given RTS may not traverse any NDPM associated with the EM. In this situation, we still want the reduced RTS to cover the EM. For this purpose, the RTSR method will keep at least one test case traversing the EM in the reduced RTS, if there is any.

Our RTSR method is as follows: Given an original EFSM model *OM*, its static dependence graph SDG_O, a regression test suite *RTS*, and a set *M* of EMs, the proposed RTSR method has the following steps:

Step (1) applies *M* to *OM* to get the modified EFSM model *MM*, and then generates SDG_M for *MM* using *OM*, *MM*, *M* and the definitions of NDPMs,

Step (2) for each EM *m* in *M* and for each test case ts_p in RTS,

- constructs Affecting IP and Affected IP for ts_p by using SDG_O and SDG_M according to the definitions of IPs, if the transition corresponding to *m* appears in ts_p.
- constructs Side-effect IP for ts_p by using SDG_O and SDG_M according to the definitions of IPs, if the transition corresponding to *m* appears in ts_p, or a sequence of transitions t_j, ..., t_k appears in ts_p, where there is an Activation CD/ GCD (t_j, t_k) contributed by *m*.

For each constructed *ip*, track the relationships between the *ts* and *ip*

Step (3) uses the probability-driven greedy algorithm (refer to Section 2.9.2) to select a minimum set of test cases that cover all IPs and all EMs that were traversed by the given RTS.

Similar to the RTSG method, Step (1) of the RTSR method applies *M* to *OM* to derive the *MM*. NDPM identification is done after *M* has been applied. Therefore, all modifications have been made to *OM* before analysis begins

Figure 5-5 shows the process of our RTSR method.

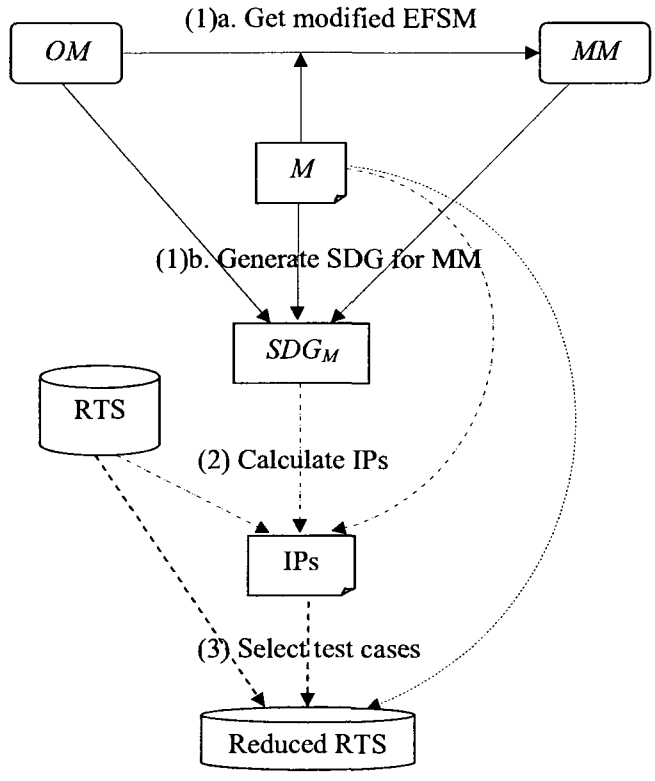


Figure 5-5. RTSR method

Step (1) of the proposed method is a straight-forward process to get *MM*. Step (2) implements our discussions in Section 5.1. In Step (3) we apply the probability-driven greedy algorithm to select test cases from the given RTS.

We have the following theorems for the RTSR method.

THEOREM 3. In the RTSR method, the order of applying the set of EMs to the original EFSM model does not affect the resulting SDG of the modified EFSM model.

PROOF. Similar to proof of Theorem 1 (refer to Section 4.2), Step (1)a of the RTSR method applies the set of EMs *M* to *OM* to get the modified EFSM model *MM*. *MM* is

the final version of the EFSM model with all EMs in M applied. We are not interested in any "middle version" during the process. In Step (1)b, OM and MM are compared to construct SDG_M. But in this step, M is not applied to OM again. Therefore, the order of applying the set of EMs to OM doesn't make any difference to the result SDG_M.

THEOREM 4. In the RTSR method, the order of applying the set of EMs to the original EFSM model does not affect the set of calculated IPs.

PROOF. By Theorem 3, the order of applying the set of EMs M to OM doesn't make any difference in the SDG_M. In Step (2), the SDG_M is used to calculate IPs for each given test case w.r.t. each EM. During this calculation, M is not applied to OM again. Hence, the order of applying the set of EMs to OM doesn't make any difference in the result set of IPs.

THEOREM 5. In the RTSR method, the order of applying the set of EMs to the original EFSM model does not affect the reduced RTS.

PROOF. By Theorem 4, the order of applying the set of EMs M to OM doesn't make any difference in set of IPs. In Step (3), the calculated IPs and M are used to select test cases from the given RTS. But during this selection, M is not applied to OM again. Hence, the order of applying the set of EMs to OM doesn't make any difference in final reduced RTS.

In the next section, we will discuss how to adopt the probability-driven greedy algorithm into the RTSR method.

5.3 Adopting Probability-driven Greedy Algorithm into the RTSR Method

In Step (3) of the RTSR method, we need to select the minimum set of test cases that cover all IPs. This is an SCP problem. We can adopt the probability-driven greedy algorithm introduced in Section 2.9.2 as a solution.

In Step (2) of our RTSR method, relationships between test case ts_p and its construct IP_q can be tracked in a table TB as follows:

for each EM in M, add a row into TB to represent it

for each test case ts_p in the given RTS, have a column of TB represents it

for each EM em in M

if em is covered by ts_p then assign Value 1 to cell (em, ts_p)

else assign Value 0 to cell (em, ts_p)

if IP_q is a new IP (which does not exist in TB), then

1). add a row for IP_q into TB

2). assign Value 1 to cell (IP_q, ts_p)

3). assign Value 0 to all other cells in the row of IP_q

if IP_q is an existing IP in TB, assign Value 1 to cell (IP_q, ts_p)

Table 5-1 shows an example of TB constructed in Step (2).

Table 5-1. Test cases vs. covered EMs and IPs

IP	ts_1	ts_2	ts_3	...	ts_n
EM_1	0	0	1	...	1
EM_2	0	0	0	...	0
...	...	...	...	...	...
EM_l	0	1	0	...	0
IP_1	1	1	0	...	1
IP_2	0	1	0	...	0
...	...	...	...	...	...
IP_m	1	0	1	...	0

In Step (3) of our RTSR method, we adapt Rampone's probability-driven greedy algorithm to select the minimum set of test cases from the given RTS that cover all IPs generated in Step (2).

In our case, the universe U is the set of IPs generated in Step (2). Each test case ts in the given RTS covers several IPs, which is a subset A_i of U. Therefore, the RTS is the family A of subset of U.

Refer to formula (ii) in Section 2.9, in the RGTR method, U_j is an IP, which is represented by a row of table TB. m is the number of EMs in M plus the number of calculated IPs, which equals to the number of rows of table TB. A_i is a test case in the given RTS, which is represented by a column of table TB. n is the number of test cases in the given RTS, which equals to the number of columns of table TB.

Appendix E contains the test case selection algorithm to be used in Step (3), and the detailed algorithm for the RTSR method.

5.4 Chapter Summary

In this chapter, we focused on RTS reduction approach. We first refined definitions of interaction patterns (IPs) and showed how to identify IPs for a given test case. Then we discussed our RTS reduction (RTSR) method. Finally, we showed how to adopt the probability-driven greedy algorithm in our RTSR method. It should be noted that the probability-driven greedy algorithm may yield different reduced RTSs depending on the order of applying test cases in the give RTS.

6. Case Studies

In order to investigate the effectiveness of the RTSG method and the RTSR method proposed in Chapters 4 and 5, respectively, we carried out case studies on seven EFSM models. The goals of these case studies are to evaluate: i) the extent of reduction of the size of a given RTS obtained by the RTSR method, ii) the size of RTSs generated by the RTSG method to cover all applicable NDPMs, and iii) the effect of introducing 'minor change' as a type of EM on the performance of the RTSG and RTSR methods. Three studies are done on each EFSM model. These seven EFSM models and the details of the studies carried out on these models are presented below.

Sometimes, in research papers, the size of a test suite is defined to be the total length of inputs. This is done to take into account the greater cost of longer test cases over shorter ones. Since our goal is to minimize the overall cost of regression testing, we try to minimize the number of regression test cases, because they will need to be managed and maintained. Therefore, in our case studies, the size of a test suite always refers to the number of test cases in the test suite

6.1 Case Study Overview

In this section, the seven EFSM models, and studies carried out on the seven models are briefly discussed. Generally, there are three studies done on each EFSM model. Study 1 and Study 2 focus on applying the RTSR method to reduce the size of a given RTS, while Study 3 applies the RTSG method to generate an RTS based on the available information.

6.1.1 IBM NTC model

The conventional view sees random testing as a second-class alternative to systematic testing. It is acknowledged that automatic test input generation in random testing is attractive, but only when an operational profile is known, and only if there is an automated oracle to judge the many test results that arise. In our case studies, we do not compare our techniques with the random testing approach. Instead, we compare our techniques with IBM's actual regression test design technique. IBM testers have to adhere rigorously to IBM's test process guidelines to ensure quality of RTSs. IBM's

skilled and knowledgeable testers are very familiar with the functionality of the SUT and have many insights concerning possible operational profiles. Hence, we believe that it is a good choice to compare our methods to IBM's regression test technique.

In collaboration with IBM Ottawa Lab Rational modeling tools system verification test (SVT) team, we were able to apply the proposed methods to one of IBM product features to be identified here as NTC. We used NTC version 1 as the original model, and NTC version 2 as the modified model. Based on the specification design document provided by the SVT team, we built the EFSM models NTCv1 and NTCv2, respectively. There were 20 states and 85 transitions in NTCv1, 22 states and 97 transitions in NTCv2. The set M of EMs that brought NTCv1 to NTCv2 consisted of 12 added transitions and 9 changed transitions.

We received two test suites TS_{ibm} and RTS_{ibm} from the SVT team. TS_{ibm} was the full test suite for the NTC feature, and RTS_{ibm} was the RTS selected by the SVT team from TS_{ibm} to test NTCv2.

Using the IBM NTC model, we conducted case studies to answer the following questions:

- Can the RTSR method be used to reduce IBM's RTS for NTCv2?
- Can the RTSR method be used as a regression test selection technique and select an RTS from IBM's full test suite for the NTC feature?
- Can the RTSG method generate an RTS for the modified IBM NTCv2 model?
- What is the fault detecting capability of RTSs constructed by our RTSG method and RTSR method w.r.t. error-prone state and transition coverage for NTCv2?

We conducted three studies on the IBM NTC model to answer these questions:

- Study 1: Reduce RTS_{ibm} to RTS_{ribm}
- Study 2: Reduce TS_{ibm} to TS_{ribm}.
- Study 3: Generate RTS_g to test NTCv2.

As part of the studies, we used defect information available to measure defect detection capability of the two reduced test suites, TS_{ribm} and RTS_{ribm}, and compared their effectiveness to the original test suites. We measured defect detection capability for the generated RTS, RTS_g, and compare it to the IBM's RTS, RTS_{ribm}. Also, we compared

RTS_g, TS_{ibm}, RTS_{ibm}, TS_{ribm} and RTS_{ribm} w.r.t. EM and NDPM coverage. The details of these comparisons are given in Section 6.2.4.

6.1.2 The simplified ATM model

This EFSM model is given in Figure 2-1. Xie used this model in her thesis [Xie 05]. There were 5 states and 8 transitions in the original EFSM model.

With the simplified ATM model, we conducted case studies to answer the following questions:

- We have stated that the RTSR method was an extension of the work of [Xie 05] for RTS reduction in Chapter 5. Compared to Xie's results, does our RTSR method show improvement?
- Can the RTSG method generate an RTS for the modified ATM model?

Three studies are done on this model.

- Study 1: Reduce a given RTS against a set of two EMs.

  Xie applied two EMs to the simplified ATM model. Since we have revised definitions of some NDPMs, introduced new NDPMs, and proposed a different RTSR method, we would like to apply our RTSR method to the same set of EMs and the same RTS used in Xie's work and compare our results to Xie's results.

- Study 2: Reduce a given RTS against a set of five EMs.

  Because we have introduced changes to a transition as a new type of EM, we want to examine the effectiveness of the proposed RTSR method on reducing test cases for a changed transition. Also, we want to have more types of NDPMs than we do in Study 1. Hence, we applied three more EMs to the simplified ATM model in this study.

- Study 3: Generate an RTS for the set of the five EMs used in Study 2.

6.1.3 The Five EFSM models

The five EFSM models include a different ATM model (ATM_Tahat), Cruise Control System, Fuel Pump, ISDN, and TCP Dialer system model.

These five EFSM models were provided by Dr. Luay Tahat. For each system, Tahat provided an original EFSM model *OM*, a modified EFSM model *MM*, and an RTS designed to test *MM*.

With the five EFSM models, we conducted case studies to answer the following questions:

- How do our methods perform on large test suites?
- Does using changes to a transition as a new EM type make a noticeable difference on the results of both the RTSR method and the RTSG method?
- Can the RTSG method generate an RTS for all five modified EFSM models?

By comparing *MM* to *OM*, we obtained a set M_O of EMs. In Tahat's research, change in a transition was not considered as a type of EMs. A change in a transition t_i was expressed as a pair of modifications (i.e., deletion of t_i and addition of a replacing transition $t_{ireplace}$). Since we consider changes to a transition as a new type of EM in this thesis, we wanted to examine the differences introduced by using change in a transition as a type of EMs. Therefore, we prepared another set of EMs, called M_C, using change in transition t_i to replace corresponding pair of deletion and addition.

The given RTSs were designed to test *MM*. But only transitions existing in *MM* appear in the given RTS. Deleted transitions do not appear in the given RTS. For a changed transition t_i, $t_{ireplace}$ instead of t_i appears in the given RTS. Based on the given RTS, we prepared two test suites to be used in our case studies, one test suite RTS_O to cover M_O, and the other test suite RTS_C to cover M_C. Details about RTS_O and RTS_C preparation are presented in Section 6.4.

Having obtained RTS_O and RTS_C, for each of the five EFSM models, we carried out three case studies:

- Study 1: Reduce RTS_O to RTS_{Or} using M_O.
- Study 2: Reduce RTS_C to RTS_{Cr} using M_C.
- Study 3: Generate RTS_{Og} and RTS_{Cg} using M_O and M_C, respectively.

As part of Study 1 and Study 2, we compared the two sets of results in terms of number of NDPMs identified, IPs generated, and RTS reduction ratio. Also, we compared the two RTSs, RTS_{Og} and RTS_{Cg}, generated in Study 3 in terms of their size and the number of NDPMs identified.

We summarize the EFSM models used in our case studies in Table 6-1.

Table 6-1. Size of EFSM models used in case studies

Model	EFSM Model Size		Studies	# of EMs	Given Test Suite
	Original EFSM	Modified EFSM			# of test cases
IBM NTC model	20 states 85 transitions	22 states 97 transitions	Study 1	21	49
			Study 2	21	74
			Study 3	21	-
Simplified ATM	5 states 8 transitions	5 states 8 transitions	Study 1	2	93
			Study 2	5	189
			Study 3	5	-
ATM_Tahat	8 states 19 transitions	8 states 22 transitions	Study 1	5	1273
			Study 2	4	834
			Study 3	5 and 4	-
Cruise Control	5 states 14 transitions	5 states 17 transitions	Study 1	5	1691
			Study 2	4	1000
			Study 3	5 and 4	-
Fuel Pump	13 states 22 transitions	13 states 25 transitions	Study 1	5	1369
			Study 2	4	922
			Study 3	5 and 4	-
ISDN	18 states 79 transitions	18 states 81 transitions	Study 1	4	998
			Study 2	3	900
			Study 3	4 and 3	-
TCP-Dialer	18 states 44 transitions	18 states 46 transitions	Study 1	4	1230
			Study 2	3	1003
			Study 3	4 and 3	-

In the following sections, we present results of our case studies on each of the seven EFSM models.

6.2 The IBM NTC Model

In this section, we discuss the results of applying our RTSR and RTSG methods onto the IBM NTC model in detail.

As we have mentioned in Section 6.1.1, there were 20 states and 85 transitions in NTCv1 whereas there were 22 states and 97 transitions in NTCv2. The set M of EMs that brought NTCv1 to NTCv2 consisted of 21 EMs where 12 EMs were additions, and 9 EMs were changes. To avoid exposing IBM confidential data, we cannot provide NTCv1 and NTCv2 in this thesis. We only list the 21 EMs to be applied to NTCv1 in Appendix F, Table APP-2. Also, names of features, parameters, and constant values have been replaced by substitutes in Table APP-2.

A full SVT test suite designed by IBM testers to test NTCv1 contained 57 system test cases. 17 new system test cases were designed to test new or changed functionality in NTCv2. Therefore, the full SVT test suite TS_{ibm} to test NTCv2 contains 74 system test cases. 49 system test cases were selected from TS_{ibm} by the testers to form SVT RTS RTS_{ibm} which was used for NTCv2 regression testing. All system test cases in the full test suite TS_{ibm} are listed in Appendix F, Table APP-3.

We should point out that the IBM SVT team might not hit every change in the NTC model in SVT testing. Their test process relies on the function verify test (FVT) team to test all functionality of the SUT. The SVT team runs end-to-end test cases. Because of time constraints, only main-flow functions and some alternative-flow functions are tested by SVT test cases. As a result, TS_{ibm} ignores some EMs, which are alternative-flow functions.

After applying the 21 EMs to NTCv1, we identified 106 applicable NDPMs. A list of the identified NDPMs w.r.t. to the twenty-one EMs is shown in Appendix F, Table APP-4.

6.2.1 Study 1: Reduce RTS_{ibm}

The IPs generated for RTS_{ibm} against each EM are listed in Appendix F, Table APP-5. In total, there are 17 IPs generated. For each of the IPs, we also give equivalent test cases w.r.t. the IP.

In Section 5.2 we have stated that, for a given test suite, some EMs may not yield any IP for two reasons: i) there may not be any NDPM associated with the EM; and ii) test cases in the given RTS may not traverse any NDPM associated with the EM. In this case study, there is no IP generated for RTS_{ibm} against 11 EMs: m_3, m_4, m_5, m_7, m_9, m_{11}, m_{12}, m_{17}, m_{19}, m_{20}, and m_{21}. In Table APP-4, we see that there is no NDPM associated with m_{17}. Therefore, an IP cannot be generated against m_{17} because of the reason i). For the remaining ten EMs in the list, RTS_{ibm} does not traverse any NDPM associated with the EMs.

Table 6-2. Selected system test cases in RTS_{ribm} vs. covered IPs

Test # / IP #		1	26	44	48	49
m_1	1				√	
m_2	2					√
	3					√
m_6	4			√		
	5			√		
m_8	6					√
	7					√
m_{10}	8			√	√	
	9			√		
	10				√	
m_{13}	11					√
	12					√
m_{14}	13	√	√			
m_{15}	14	√				
m_{16}	15			√		
	16					√
m_{18}	17		√			

According to the proposed RTSR method, each selected system test case has to cover at least one unique IP. Based on Table APP-5, our RTSR method selected five system test cases from RTS_{ibm} that cover all 17 IPs. The five system test cases are: Test 1, Test 26, Test 44, Test 48 and Test 49. Relationships between the selected system test cases and covered IPs are shown in Table 6-2. This table clearly shows that each selected test case in RTS_{ribm} covers unique IPs.

RTS_{ibm} reduction results are summarized against EMs in Table 6-3. In the third column, numbers in parenthesises are ids of selected regression test cases.

Table 6-3. RTS_{ibm} reduction results vs. EMs

EM Under Test	**Number of regression test cases**	**Number of reduced regression test cases (test #)**	**List of eliminated test cases (specified by Test #)**
m_1 (Add t_{87})	1	1 (48)	
m_2 (Add t_{88})	2	1 (49)	50
m_3 (Add t_{89})	0	0	
m_4 (Add t_{90})	0	0	
m_5 (Add t_{91})	0	0	
m_6 (Add t_{92})	4	1 (44)	45, 46, 47
m_7 (Add t_{93})	0	0	
m_8 (Add t_{94})	2	1 (49)	50
m_9 (Add t_{95})	0	0	
m_{10} (Add t_{96})	5	2 (44, 48)	45, 46, 47
m_{11} (Add t_{97})	0	0	
m_{12} (Add t_{98})	0	0	
m_{13} (change t_{12})	7	1 (49)	50, 61, 63, 64, 66, 67
m_{14} (change t_{15})	23	1 (1)	2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 26, 34, 35, 39, 41, 57, 58, 66, 67, 68, 73
m_{15} (change t_{16})	12	1 (1)	2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

m_{16} (change t_{31})	40	3 (1, 44, 49)	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17, 19, 20, 24, 26, 34, 35, 36, 37, 38, 39, 40, 41, 45, 46, 47, 50, 52, 57, 58, 61, 63, 64, 66, 67
m_{17} (change t_{39})	4	0	61, 62, 63, 64
m_{18} (change t_{44})	7	1 (26)	34, 35, 39, 41, 57, 58
m_{19} (change t_{78})	0	0	
m_{20} (change t_{81})	0	0	
m_{21} (change t_{84})	3	0	68, 70, 73

We notice that, for some EMs, all test cases that traverse the EM may be eliminated. For example, for m_{17} (change t_{39}), all four test cases that traverse this EM are eliminated. This is because there is no NDPM w.r.t. the EM (i.e., m_{17}), or even the test cases traverse the EM, they do not test any NDPMs w.r.t the EM (i.e., m_{21}). Hence, no IP is generated, and none of the test cases is selected into the reduced RTS.

Since we do not want to decrease EM coverage of the given RTS, after reducing a given RTS, Step (3) of the proposed RTSR method (refer to Section 5.2 for details) selects one test case from the removed test cases that traverse the transition corresponding to the EM for each untested EM, and puts the test case into the reduced RTS. For the IBM NTC model, we select Test 61 to test m_{17} (change t_{39}) and Test 68 to test m_{21} (change t_{84}). The final reduced RTS contains seven test cases.

Using the proposed RTSR method, we reduced the size of RTS_{ibm} from 49 to seven system test cases as shown in Table 6-4. The reduced RTS was named as RTS_{ribm}, which only contains 14.29% of test cases of RTS_{ibm}.

It is noticed that RTS_{ibm} is highly redundant (refer to Appendix F, Table APP-3). The same test sequences may need to be tested by several system test cases that carry different test data. We agree that this fact contributes to the high reduction ratio of RTS_{ribm}. But it is not the only cause of our case study results. There are 22 unique test cases in RTS_{ibm}. RTS_{ribm} keeps seven of the unique test cases as shown in Table 6-4. Only considering unique test cases, the proposed RTSR method eliminates 68.18% of the

unique test cases from RTS_{ibm}. This reduction ratio is still high. Therefore, results of this study are good not solely due to the high redundancy of RTS_{ibm}, but also due to our proposed method.

Table 6-4. Reduced RTS RTS_{ribm}

Test #	Test Sequences
1	t1, t2, t15, t16, t37, t31
26	t1, t2, t15, t44, t45, t37, t31
44	t1, t2, t96, t92, t31
48	t1, t2, t96, t87 / t1, t2, t86
49	t1, t2, t86/ t1, t2, t88, t94, t12, t31
61	t1, t2, t38, t39, t12, t31
68	t1, t2, t15, t84

6.2.2 Study 2: Reduce TS_{ibm}

The proposed RTSR method can also be used as a regression test selection method, if we apply it to a full test suite. In this study, we applied the RTSR method as a regression test selection method to TS_{ibm} containing 74 system test cases.

IPs generated by RTSR method for each EM with the equivalent system test cases in TS_{ibm} w.r.t. IPs are shown in Appendix F, Table APP-6.

We notice that Table APP-6 and Table APP-5 contain the same set of IPs. Therefore, TS_{ibm} has more test cases than RTS_{ibm}, but doesn't cover more IPs. Because the RTSR method looks for a minimum set of test cases that cover all calculated IPs, and TS_{ibm} is a super set of RTS_{ibm}, the reduced RTS from TS_{ibm} should not contain more than seven test cases.

As expected, seven system test cases were selected to form a reduced RTS TS_{ribm} as shown in Table 6-5. A reduction of 90.54% of test cases of TS_{ibm} is thus obtained.

Similar to Study 1, TS_{ibm} is highly redundant (refer to Appendix F, Table APP-3). The redundancy contributes to the high reduction ratio of TS_{ribm}. But it is not the only cause of our case study results. There are 30 unique test cases in TS_{ibm}. TS_{ribm} keeps seven of the

unique test cases as shown in Table 6-5. Only considering unique test cases, the proposed RTSR method achieves a high reduction ratio by eliminating 76.67% of the unique test cases from TS_{ibm}. Therefore, results of this study are good not solely due to the high redundancy of TS_{ibm}, but also due to our proposed RTSR method.

Table 6-5. Reduced RTS TS_{ribm}

Test #	Test Sequences
1	t1, t2, t15, t16, t37, t31
25	t1, t2, t11, t32, t56, t64, t37, t31 / t1, t2, t15, t44, t45, t37, t31
44	t1, t2, t96, t92, t31
48	t1, t2, t96, t87 / t1, t2, t86
49	t1, t2, t86 / t1, t2, t88, t94, t12, t31
61	t1, t2, t38, t39, t12, t31
68	t1, t2, t15, t84

Comparing Table 6-5 to Table 6-4, we notice that six out of seven test cases appear in both tables. The only difference is Test 25 instead of Test 26 is selected to be in TS_{ribm}. That is because both Test 25 and Test 26 cover IP #13 and IP #17, and Test 25 is not in RTS_{ribm} (refer to Tables APP-3, APP-5, APP-6). Since Test 25 has been processed before Test 26 by the RTSR method, it is chosen to be in TS_{ribm}.

According to the proposed RTSR method, each selected system test case has to cover at least one unique IP shown in Table APP-6. Relationships between system test cases in TS_{ribm} and covered IPs are shown in Table 6-6. This table clearly shows that each selected test case in TS_{ribm} covers unique IPs.

From this table, we can see that Test 61 and Test 68 do not cover a unique IP. But, as we have discussed in Study 1 (refer to Section 6.2.1), we do not eliminate these two system test cases because they traverse two untested EMs m_{17} (Change t39) and m_{21} (Change t84).

Table 6-6. Selected system test cases in TS_{ribm} vs. covered IPs

IP # \ Test #		1	25	44	48	49	61	68
m_1	1				√			
m_2	2					√		
	3					√		
m_6	4			√				
	5			√				
m_8	6					√		
	7					√		
m_{10}	8			√	√			
	9			√				
	10				√			
m_{13}	11					√	√	
	12					√	√	
m_{14}	13	√	√					√
m_{15}	14	√						
m_{16}	15			√				
	16					√		
m_{18}	17		√					

TS_{ibm} reduction results versus EMs are summarized in Table 6-7. In the third column, numbers in parenthesises are ids of selected test cases. The results presented in this table show that, the proposed RTSR method can be used as an RTS selection method.

Table 6-7. TS_{ibm} reduction results vs. EMs

EM Under Test	Number of regression test cases	Number of reduced test cases (test #)	List of eliminated test cases (specified by Test #)
m_1 (Add t_{87})	1	1 (48)	
m_2 (Add t_{88})	2	1 (49)	50
m_3 (Add t_{89})	0	0	
m_4 (Add t_{90})	0	0	
m_5 (Add t_{91})	0	0	
m_6 (Add t_{92})	4	1 (44)	45, 46, 47
m_7 (Add t_{93})	0	0	
m_8 (Add t_{94})	2	1 (49)	50
m_9 (Add t_{95})	0	0	
m_{10} (Add t_{96})	5	2 (44, 48)	45, 46, 47, 48
m_{11} (Add t_{97})	0	0	
m_{12} (Add t_{98})	0	0	
m_{13} (change t_{12})	9	1 (49)	50, 53, 61, 62, 63, 64, 66, 67
m_{14} (change t_{15})	39	1 (1)	2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 25, 26, 27, 28, 29, 30, 33, 34, 35, 39, 41, 42, 51, 53, 54, 55, 56, 57, 58, 60, 66, 67, 68, 70, 73
m_{15} (change t_{16})	14	1 (1)	2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16
m_{16} (change t_{31})	60	3 (1, 44, 49)	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 63, 64, 66, 67
m_{17} (change t_{39})	4	1 (61)	62, 63, 64
m_{18} (change t_{44})	16	1 (25)	26, 27, 28, 29, 30, 33, 34, 35, 39, 41,

			42, 51, 54, 58, 60
m_{19} (change t_{78})	0	0	
m_{20} (change t_{81})	0	0	
m_{21} (change t_{84})	3	1 (68)	70, 73

6.2.3 Study 3: Generate RTS_g

Using the proposed RTSG method, we generated an RTS RTS_g to test all 106 applicable NDPMs and all 21 EMs. 35 system test cases in RTS_g are listed in Appendix F, Table APP-7. Since there is no transition deleted from NTCv1, we do not expect any of the generated regression test cases to fail.

We have noticed that there is no NDPM identified against m_{17} in Section 6.2.1 (refer to Table APP-4). But we still want to have regression test cases to test this EM. This EM is not ignored by the RTSG method. In RTS_g Test 35 is generated to cover the corresponding changed transition t_{39}.

Relationships between the generated regression test cases and covered NDPMs and EMs are shown in Appendix F, Table APP-8. This table clearly shows that the RTSG method successfully generates an RTS for the modified IBM model, each generated test case in RTS_g covers unique NDPMs / EMs, and all applicable NDPMs and EMs are covered by one or more test cases.

So far, in this case study we have five test suites to test NTCv2, namely, TS_{ibm}, RTS_{ibm}, TS_{ribm}, RTS_{ribm}, and RTS_g. One of the test suites is a full test suite, and the other four test suites are RTSs. In the next section, we compare the five test suites in terms of their size, NDPM coverage, EM coverage and error detection capability.

6.2.4 Effectiveness of RTSs

An effective RTS should contain test cases to test both direct and indirect effects of the modifications. In our research, such effects can be captured by the occurrences of NDPMs related to the three types of regression testing discussed in Chapter 3. Therefore, in order to assess the effectiveness of RTS_{ribm}, TS_{ribm}, and RTS_g, we measured NDPM coverage of these test suites. In Chapter 5, we have stated that we assumed executability

of test cases was ensured by test case designers of the given RTS. Therefore, IBM's test suites should not contain non-executable test cases. Only applicable NDPMs can be covered by IBM's test suites. As subsets of IBM's test suites, RTS_{ribm} and TS_{ribm} can only cover applicable NDPMs, too. Besides, the RTSG method only considers applicable NDPMs. As a result, RTS_g cannot cover an NDPM that is not applicable. Therefore, in this thesis, when we measure NDPM coverage, we only count applicable NDPMs.

Moreover, since an EM may not always have an associated NDPM, we measured EM coverage of RTS_{ribm}, TS_{ribm}, and RTS_g. NDPM coverage and EM coverage were also measured against TS_{ibm} and RTS_{ibm} for comparison.

Table 6-8 presents measurements for TS_{ibm}, RTS_{ibm}, TS_{ribm}, RTS_{ribm}, and RTS_g. For TS_{ibm} and RTS_{ibm}, we also give the number of unique test cases in parenthesises in the column "Number of system test cases".

Table 6-8. EM and NDPM coverage measurement

Test Suite	**Number of system test cases**	**EM Coverage**	**NDPM Coverage (applicable NDPMs only)**
SVT Full Test Suite (TS_{ibm})	74 (30)	57.14%	30.19%
SVT Regression Test Suite (RTS_{ibm})	49 (22)	57.14%	30.19%
Reduced Regression Test Suite Selected from TS_{ibm} (TS_{ribm})	7	57.14%	30.19%
Reduced Regression Test Suite (RTS_{ribm})	7	57.14%	30.19%
Generated Regression Test Suite (RTS_g)	35	100%	100%

From Table 6-8, we observed that the two IBM test suites had the same NDPM coverage of 30.19% (32 out of 106 applicable NDPMs), and EM coverage of 57.14% (12 out of 21 modifications). Both RTS_{ribm} and TS_{ribm} achieved the same NDPM coverage and

EM coverage as the original test suites. As expected, the RTSR method does not decrease NDPM coverage and EM coverage in test suite reduction. Also, since the RTSR method only does test case selection, it is impossible for the reduced test suite to achieve higher coverage than the original test suite.

From TS_{ibm}, IBM's testers manually selected RTS_{ibm}, and the RTSR method automatically selected TS_{ribm}. Both RTS_{ibm} and TS_{ribm} achieved the same NDPM coverage and EM coverage. IBM's testers' manual selection does not decrease NDPM coverage and EM coverage of RTS_{ibm}. Hence, in terms of NDPM coverage and EM coverage, RTS_{ibm} has good quality. This fact increases our confidence to compare our methods with IBM's regression test techniques instead of random testing approaches. Comparing TS_{ribm} with RTS_{ibm}, TS_{ribm} achieves the same coverage as RTS_{ibm} by fewer test cases. The size of TS_{ribm} is 1/7 of RTS_{ibm} (or less than 1/3 of RTS_{ibm} without considering redundant test cases).

The generated RTS_g achieved higher NDPM coverage and EM coverage than the original IBM test suites and the reduced test suites constructed by the RTSR method. This is also an expected result, since the goal of the RTSG method is to generate test cases to cover all applicable NDPMs and EMs. We have observed that RTS_g is the largest test suite among the five test suites, if we only count unique test cases in TS_{ibm} and RTS_{ibm}. Considering RTS_g's high coverage, its size is still reasonable.

We did not measure the average number of NDPMs covered by each generated test case in RTS_g, because this measurement lacks reliability; for example, different orders of inputting NDPMs to the RTSG method can result in different RTSs.

To determine the defect detection capability of the reduced RTSs and the generated RTS, we collected NTC defect information from the IBM defect tracking system. There are six SVT defects in NTCv2 related to requirements. Only two of these defects were detected in IBM's NTCv2 regression testing phase, while the other four defects were detected later, for example, in test phases of NTC version 3. We located these six defects in the EFSM model and found 3 states and 4 transitions relating to these six defects. These states and transitions are named as error-prone states and error-prone transitions. SVT system test cases to test the NTC feature are executed and verified manually by testers.

To detect a defect, two conditions have to be satisfied. First, a system test case has to traverse an error-prone state or transition. Secondly, the tester who runs this system test case has to observe unexpected outputs. But, during execution of a system test case, a tester may fail to observe all unexpected outputs. Also, in practice, it may be difficult to recognize that unexpected behavior has occurred, and therefore, the presence of a fault may go undetected. For example, it appears that several error-prone states and transitions in the NTC feature were traversed by test cases in RTS_{ibm}. But defects in the error-prone states and transitions were not reported. Satisfaction of the second condition is out of the scope of our research. Therefore, in our measurement, we only considered the first condition. Of course, data selection is very important in test case design, but again, that is not the focus of this thesis. An error-prone state or error-prone transition is considered as covered if at least one system test case in a test suite traverses this state or transition.

It is not necessary to physically execute an RTS to determine if an error-prone state or transition is covered by the RTS. Since we consider traversal of error-prone states and transitions as the only condition to detect faults, it is not necessary to physically execute the RTS to determine if a fault can be detected by the RTS. As a result, we didn't execute any RTS in this case study. Based on the defect information we collected from IBM, we can easily determine which defects would be detected and which defects would be missed by the RTSs.

Table 6-9. Traversal of error-prone states and transitions

Test Suite	Error-prone State Coverage	Error-prone Transition Coverage	Total Coverage
TS_{ibm}	100.00%	75.00%	85.71%
RTS_{ibm}	100.00%	75.00%	85.71%
TS_{ribm}	100.00%	75.00%	85.71%
RTS_{ribm}	100.00%	75.00%	85.71%
RTS_g	100.00%	100.00%	100.00%

Details shown in Table 6-9 indicate that the reduced RTSs give the same error-prone state coverage of 100% (3 out of 3 error-prone states) and error-prone transition coverage

of 75% (3 out of 4 error-prone transitions) as the original test suites. The overall error-prone state and transition coverage for TS_{ibm}, RTS_{ibm}, TS_{ribm}, and RTS_{ribm} are all 85.71% (6 out of 7 error-prone states and transitions).

However, such a substantial reduction may not be realized in every case. For example, in an EFSM model, there may be states and transitions which are neither modified nor related to any modification. But if a corrective defect exists in the states or transitions, these states or transitions are also error-prone states or error-prone transitions. A given RTS may contain test cases traversing these states or transitions. But the reduced RTS may eliminate all these test cases, since these test cases may not cover unique IPs. In this case, the reduced RTS may not traverse the error-prone states or error-prone transitions. Thus, the reduced RTS may have less capability to detect corrective defects than the original RTS. Nonetheless, the results from our studies indicate that the proposed RTSR method significantly reduces given test suites without decreasing the coverage of NDPMs and EMs, because our method selects test cases for all IPs and EMs that are traversed by the given RTS.

The generated RTS achieved higher error-prone transition coverage than the original IBM test suites. RTS_g achieved 100% coverage for both error-prone state and error-prone transition coverage. This is an encouraging result. Previously we have discussed EM coverage against the two IBM test suites. We noticed that only 57.14% of EMs were covered by TS_{ibm} and RTS_{ibm} (refer to Table 6-8 for details). But in NTCv2, one transition corresponding to an uncovered EM is an error-prone transition. Omission of this EM decreases error-prone transition coverage of TS_{ibm} and RTS_{ibm}, and consequentially, decreases defect detection capability of TS_{ibm} and RTS_{ibm}. The RTSG method generates test suites to cover all applicable NDPMs and EMs. Hence, in this case study, RTS_g has higher error-prone transition coverage than TS_{ibm} and RTS_{ibm}, because none of the EMs is ignored. As a result, RTS_g has higher fault detection capability than TS_{ibm} and RTS_{ibm}.

As a conclusion, in this case study, the RTSR method reduced IBM's RTS for the NTC feature successfully. Also, this method was used as a regression test selection technique and successfully selected an RTS to test NTCv2 from the IBM's full test suite. The RTSG method successfully generated an RTS to test NTCv2. In terms of defect detecting capability, the reduced RTSs constructed by the RTSR method achieved the

same error-prone state and transition coverage as the original IBM test suites, and the RTS generated by the RTSG method achieved higher error-prone transition coverage than the IBM test suites.

6.3 The Simplified ATM Model

In this section, we present results of the three studies that have been done on the simplified ATM model.

6.3.1 Study 1

Xie used this EFSM model in her thesis [Xie 05] for RTS reduction. We have stated that the proposed RTSR method is an extension of Xie's work in Chapter 5. Comparison to Xie's results helps us determine if the RTSR method shows improvement. In this study, we applied the proposed RTSR method to the same set of EMs and the same RTS used in Xie's work and compare our results to Xie's results.

The RTS used by Xie contained 93 test cases as listed in Appendix G, Table APP-9. The set of EMs used by Xie contained two EMs:

m_1: adding t_9: <S_2, S_3, *Balance*, ε, Display(b), ε>;

m_2: deleting t_6.

T6dummy is the td_{new} to replace the deleted transition t_6.

1. Xie's results

Xie's RTS reduction results versus EMs are summarized in Table 6-10 [Xie 05]. The second column of Table 6-10 indicates the number of regression test cases in the given RTS that test the EM, and the third column indicates the number of selected regression test case to be in the reduced RTS that test the EM.

Table 6-10. Xie's RTS reduction results vs. EMs

EM under test	Number of regression test cases that test the EM	Number of selected regression test cases that test the EM	List of eliminated test cases (specified by test #)
Add t_9	86	5	3, 5, 6, 9, 10, 11, 12, 13, 14, 15, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
Delete t_6	79	11	8, 9, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 64, 65, 66, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92

The 15 test cases contained in Xie's reduced RTS are listed in Table 6-11.

Table 6-11. Xie's reduced RTS

Test #	Test Sequences
2	T1 T4 T5 T7 T9 T7 T8
3	T1 T4 T5 T7 T6dummy T9 T7 T8
4	T1 T4 T6dummy T5 T7 T8
5	T1 T4 T6dummy T5 T7 T9 T7 T8
6	T1 T4 T6dummy T5 T7 T6dummy T9 T7 T8
7	T1 T4 T6dummy T9 T7 T5 T7 T8

8	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T8
17	T1 T4 T5 T7 T5 T7 T9 T7 T8
18	T1 T4 T5 T7 T5 T7 T6dummy T9 T7 T8
19	T1 T4 T5 T7 T6dummy T5 T7 T8
20	T1 T4 T5 T7 T6dummy T5 T7 T9 T7 T8
21	T1 T4 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
33	T1 T4 T6dummy T5 T7 T5 T7 T6dummy T9 T7 T8
46	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T8
47	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T9 T7 T8

2. Our results

We identified 13 NDPMs for the addition of t_9 and the deletion of t_6 as shown in Appendix G, Table APP-10. IPs generated for these two EMs, with the equivalent test cases w.r.t. certain IPs are listed in Appendix G, Table APP-11. Relationships between test cases in our reduced RTS and IPs covered by the test cases are shown in Appendix G, Table APP-12.

[Xie 05] identified 11 NDPMs and calculated 21 IPs w.r.t. addition of t_9 and deletion of t_6. Our RTSR method identified 13 NDPMs and calculated 18 IPs. The two NDPMs that were missed by Xie but were identified by us are both Affected GCD. As we have stated in Section 3.2.2, Affected GCD is introduced and defined by us. Hence, Affected GCD cannot be identified by Xie. Xie didn't give details about the 21 IPs she obtained, so we cannot compare our IPs with hers. But we expect differences between these two set of IPs, because of the new NDPM definitions we introduced and changes in NDPM definitions we made.

For ease of comparison with Xie's results, we present our results in tables that have the same format used for presenting Xie's results. Our RTS reduction results are summarized against EMs in Table 6-12. In the third column, numbers in parenthesises are ids of selected test cases.

Table 6-12. Our RTS reduction results vs. EMs

EM under test	Number of regression test cases that test the EM	Number of selected regression test cases that test the EM (selected test #)	List of eliminated test cases (specified by test #)
Add t_9	86	7 (5, 18, 21, 25, 32, 33, 46)	2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 20, 22, 23, 24, 26, 27, 28, 29, 30, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
Delete t_6	79	8 (5, 18, 21, 25, 31, 32, 33, 46)	3, 4, 6, 7, 8, 9, 12, 13, 14, 15, 19, 20, 24, 26, 27, 28, 29, 30, 31, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 64, 65, 66, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92

The eight test cases contained in our reduced RTS are listed in Table 6-13.

In Xie's results (Table 6-10), her reduced RTS contains five test cases to test the addition of t_9, and 11 test cases to test the deletion of t_6. But only one of these test cases tests both the addition of t_9 and the deletion of t_6. Xie's reduced RTS contains 15 test cases in total (refer to Table 6-11 for details). As shown in Table 6-12, our reduced RTS contains seven test cases to test the addition of t_9, and eight test cases to test the deletion of t_6. Because seven of these test cases test both the addition of t_9 and the deletion of t_6, our reduced RTS contains eight test cases in total, as shown in Table 6-13.

Table 6-13. Our reduced RTS

Test #	Test Sequences
5	T1 T4 T6dummy T5 T7 T9 T7 T8
18	T1 T4 T5 T7 T5 T7 T6dummy T9 T7 T8
21	T1 T4 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
25	T1 T4 T5 T7 T6dummy T9 T7 T5 T7 T8
31	T1 T4 T6dummy T5 T7 T5 T7 T8
32	T1 T4 T6dummy T5 T7 T5 T7 T9 T7 T8
33	T1 T4 T6dummy T5 T7 T5 T7 T6dummy T9 T7 T8
46	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T8

Comparing our reduced RTS to Xie's reduced RTS, Test 2, 3, 4, 6, 7, 8, 17, 19, 20, 47 were selected by Xie, but were eliminated by us. Test 25, 32, 34 were selected by us, but were eliminated by Xie. Five test cases (Test 5, 18, 21, 33, 46) were selected by both Xie and us.

In Xie's work, her algorithm takes one EM and calculates IPs for the first test case in a given RTS. If there is an IP calculated, this test case will be selected. All calculated IPs will be categorized by their types (Affecting IP, Affected IP, and Side-effect IP). Then the algorithm takes the next test case and calculates IPs w.r.t. the EM. This test case will be selected only if a unique IP (has not been calculated before) is calculated. Newly calculated IPs will be categorized by their types, too. The calculation and selection procedure will be repeated until all EMs and all test cases in the given RTS have been processed.

Xie's algorithm selects a test case having a unique IP that has not been calculated before. But if we look back to previously selected test cases, we may find out that these test cases may not have unique IPs anymore, since all of the IPs they cover might have been covered by test cases selected later. Hence, Xie's algorithm cannot ensure that every selected test case covers a unique IP.

Our RTSR method calculates IPs for all test cases in the given RTS w.r.t. all EMs. After all IPs have been calculated, the probability-driven greedy algorithm is adopted to

solve the test cases selection problem as an SCP problem. As a result, our RTSR method ensures that every selected test case covers a unique IP.

Our reduced RTS contains only 8.60% of test cases in the given RTS, while Xie's reduced RTS contains 16.13% of the test cases. These results show that, for the same EFSM model, same set of EMs and same RTS, our RTSR method reduced the size of the given RTS more effectively than Xie's algorithm did.

6.3.2 Study 2

We introduced changes to a transition as a new type of EM. Since this type of EM was not covered in Study 1, we would like to investigate the effectiveness of the proposed RTSR method on reducing test cases designed for changed transitions. Also, we want to cover NDPMs types which were not identified in Study 1. Hence, besides the two EMs used by Xie, we introduced three other EMs to the simplified ATM model:

m_3: changing t_5 to: <S_2, S_3, *Withdrawal*(w), $w < b$, ε, $b = b - w$>;

m_4: adding t_{10}: <*Start*, *Exit*, *CardError*, ε, ε, Display error; Eject card>;

m_5: deleting t_3.

To test the five EMs, we constructed an RTS, based on the RTS used in Study 1. We refer to the RTS used in Study 1 as RTS_1, and the newly constructed RTS as RTS_2. We noticed that T2 never appeared in Test 1-92 in RTS_1. But we would like to test the effects of T2 on other transitions in Study 2. Therefore, we replaced sequence T1 T4 in Test 1-92 in RTS_1 by T1 T2 T4 to make new test cases. For example, Test 1 in RTS_1 was T1 T4 T5 T7 T8. We replaced the sequence T1 T4 by T1 T2 T4. The new test case Test 1: T1 T2 T4 T5 T7 T8 was formed and included into RTS_2.

Since T3 in Study 2 was a deleted transition, we replaced T3 in Test 93 in RTS_1 by T3dummy. Together with three new test cases: Test 94, 95 and 96, RTS_2 covers all paths traversing T3. Test 97 is designed to traverse the added transition T10. Since this is the only path traversing T10, RTS_2 achieves 100% path coverage for the new transition T10.

As a result, RTS_2 contains 97 test cases to test the five EMs as shown in Table 6-14.

Table 6-14. RTS used in Study 2 for the simplified ATM model

Test id	Test Sequences
1-92	Replace (T1 T4) by (T1 T2 T4) in Test 1-92 in RTS_1
93	T1 T2 T2 T2 T3dummy
94	T1 T3dummy
95	T1 T2 T3dummy
96	T1 T2 T2 T3dummy
97	T10

Twenty-six NDPMs were identified for the five EMs as shown in Table APP-13 in Appendix G. One of the 26 NDPMs is not applicable. Thirty-six IPs were generated for the five EMs. The generated IPs, with the equivalent test cases w.r.t. an IP, are shown in Table APP-14 in Appendix G. By comparing Table APP-11 to Table APP-14, we can see that in both Study 1 and Study 2, the RTSR method calculated the same IPs for t_9 and t_{6dummy}. Adding T2 into Test 1-92 does not introduce new IPs for t_9 and t_{6dummy}. This is because there is no dependence from t_2 to t_9 and from t_2 to t_{6dummy}. Table APP-14 also lists IPs calculated in Study 2 for t_5, t_{10}, and t_3.

After applying the proposed RTSR method, we obtained a reduced RTS containing 15 test cases listed in Table 6-15. The reduction ratio is 84.54%. Our RTS reduction results against EMs are summarized in Table APP-15 in Appendix G. Relationships between test cases in our reduced RTS and IPs covered by the test cases are shown in Appendix G, Table APP-16.

Table 6-15. Reduced RTS in Study 2

Test #	Test Sequences
1	T1 T2 T4 T5 T7 T8
3	T1 T2 T4 T5 T7 T6dummy T9 T7 T8
4	T1 T2 T4 T6dummy T5 T7 T8
5	T1 T2 T4 T6dummy T5 T7 T9 T7 T8
10	T1 T2 T4 T9 T7 T5 T7 T8

18	T1 T2 T4 T5 T7 T5 T7 T6dummy T9 T7 T8
21	T1 T2 T4 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
31	T1 T2 T4 T6dummy T5 T7 T5 T7 T8
32	T1 T2 T4 T6dummy T5 T7 T5 T7 T9 T7 T8
33	T1 T2 T4 T6dummy T5 T7 T5 T7 T6dummy T9 T7 T8
49	T1 T2 T4 T6dummy T9 T7 T5 T7 T6dummy T5 T7 T8
92	T1 T2 T4 T6dummy T9 T7 T8
93	T1 T2 T2 T2 T3
94	T1 T3dummy
95	T1 T2 T3dummy

In this study, ten types of NDPMs were identified. Compared to Study 1, two more types of NDPMs are identified. Moreover, the proposed RTSR method significantly reduced the number of test cases testing changed transitions.

6.3.3 Study 3

For the simplified ATM model, we generated eight regression test cases to test NDPMs associated with five EMs used in Study 2. Since there are deleted transitions in this EFSM model, we expect some generated test cases to fail during execution. The generated regression test cases are listed in Table 6-16.

Table 6-16. Generated RTS for the Simplified ATM model

Test #	Test Sequences	Expect Results
1	t1 t4 t9 t7 t8	
2	t1 t4 t5 t7 t9 t7 t8	
3	t1 t4 t5 t7 t6dummy t7 t8	We expect this test case to fail
4	t1 t4 t6dummy t7 t6dummy t7 t8	We expect this test case to fail
5	t1 t4 t6dummy t7 t5 t7 t8	We expect this test case to fail
6	t1 t4 t5 t7 t5 t7 t8	
7	t1 t2 t2 t2 t3dummy	We expect this test case to fail
8	t10	

Relationships between the generated regression test cases and NDPMs covered by the test cases are shown in Appendix G, Table APP-17.

In this study, the RTSG method generated eight test cases to cover all 25 applicable NDPMs. This is an encouraging result. Considering the number of NDPMs under test, the size of the generated RTS is relatively small.

6.4 The Five EFSM Models

In this section, we present our studies on the five EFSM models provided by Tahat (referred as the Five EFSM models). For each EFSM model, we prepared two set of EMs: M_O and M_C, and two RTSs: RTS_O to test M_O, and RTS_C to test M_C. Three studies have been done on these five EFSM models.

By comparing *MM* to *OM*, we obtained a set M_O of EMs. In Tahat's research, change in a transition was not considered as a type of EMs. A change in a transition t_i was expressed as a pair of EMs (i.e., deletion of t_i and addition of a replacing transition $t_{ireplace}$). In *MM*, since a changed transition t_i does not exist any more, t_i has been replaced by its replacing transition $t_{ireplace}$. Hence, there are only two types of EMs in Tahat's research, namely, addition and deletion of a transition.

Since we consider changes in a transition as a new type of EM in this thesis, we wanted to examine the differences introduced by using a change in a transition as a type of EMs. Therefore, we prepared another set of EMs, called M_C. We first made a copy of M_O as M_C. Then we identified the pair of deletion and addition corresponding to each changed transition in M_O. Finally, in M_C, each pair of deletion of t_i and addition of $t_{ireplace}$ was replaced by a change in transition t_i.

The given RTSs are designed to test *MM*. According to Tahat, they are created using specification-based testing methods, i.e., equivalence class partitioning and boundary-value analysis. In addition, test cases are derived based on model-based testing, i.e., transition coverage and partial path coverage. The RTSs contain also a small number of randomly generated test cases. But only transitions existing in *MM* appear in a given RTS (deleted transitions do not appear). For a changed transition t_i, $t_{ireplace}$ instead of t_i appears in a given RTS.

Since a given RTS does not contain any test case for deleted transitions, based on the given RTS, we prepared two test suites to be used in our case study, one test suite RTS_O to cover M_O, and the other test suite RTS_C to cover M_C.

In both Tahat's and our research, for each deleted transition t_i, a new transition t_{idummy} with an empty sequence of actions is added to MM at the starting state of t_i to represent input interaction $i(t_i)$ and enabling condition $c(t_i)$. We prepared RTS_O and RTS_C as follows:

1) We copied all test cases in the given RTS to RTS_O and RTS_C.
2) For each deleted transition t_i, we manually designed new test cases traversing t_{idummy}. These new test cases were included in RTS_O and RTS_C.
3) For each changed transition t_i, we identified all test cases traversing $t_{ireplace}$.
   a) In RTS_O, we wanted to have test cases traversing both $t_{ireplace}$ and t_{idummy}. So we made a copy of all test cases traversing $t_{ireplace}$, and then revised the copy by replacing every $t_{ireplace}$ using t_{idummy}. Thus, RTS_O contained test cases traversing both $t_{ireplace}$ and t_{idummy}.
   b) In RTS_C, we replaced every $t_{ireplace}$ by t_i.

6.4.1 Study 1: Reduce RTS_O Using M_O

Because of the size of the RTSs to be reduced, it is not practical to provide details of our studies as we did in Section 6.2 and 6.3 for the Five EFSM models. In this section and the next section, we only summarize RTS reduction results we obtained for the Five EFSM models.

Table 6-17 presents results of reducing RTS_O using M_O with the percentage of reduction.

The results indicate that based on the NDPMs and IPs defined in this thesis, the proposed RTSR method can significantly reduce the size of the given test suites, even changes in a transition is not considered as a type of EM. Moreover, the Five EFSM models all have large test suites (refer to Table 6-1). The RTSR method performs well with these large test suites.

Table 6-17. Measurement of reduced RTS_O for M_O

Model	Size of Original RTS	Number of Identified NDPMs	Number of Computed IPs	Size of Reduced RTS	Percentage of Reduction
ATM-2	1273	44	67	57	95.52%
Cruise Control	1691	205	384	285	83.15%
Fuel Pump	1369	25	15	10	99.67%
ISDN	998	20	112	107	89.28%
TCP-Dialer	1230	21	63	55	95.53%
Average					92.63%

6.4.2 Study 2: Reduce RTS_C Using M_C

Table 6-18 summarizes results of reducing RTS_C using M_C in terms of the percentage of reduction obtained by applying the proposed RTSR method. The results indicate that the proposed RTSR method significantly reduces given test suites without decreasing the coverage of NDPMs and EMs.

Table 6-18. Measurement of reduced RTS_C for M_C

Model	Size of Original RTS	Number of Identified NDPMs	Number of Computed IPs	Size of Reduced RTS	Percentage of Reduction
ATM_Tahat	834	18	24	15	98.20%
Cruise Control	1000	127	139	152	84.80%
Fuel Pump	922	19	12	9	99.09%
ISDN	900	10	54	49	94.56%
TCP-Dialer	1003	11	27	24	97.61%
Average					94.85%

Comparing Table 6-18 to Table 6-17, we can see that for four out of the five EFSM models, the proposed RTSR method has a higher reduction ratio in Study 2. Also, the average percentage of reduction in Study 2 is higher than the average percentage of

reduction in Study 1. This indicates that considering changes in a transition as a type of EM, the amount of reduction of the given test suites can be increased.

From Table 6-17 and Table 6-18, we can also see that for all five EFSM models, fewer NDPMs are identified, and fewer IPs are generated for M_C. Note that there is one changed transition in each of these five EFSM models. For example, for the Cruise Control system, in M_C, m_1, m_2, and m_3 are addition of transitions, and m_4 is a changed transition. m_1, m_2, and m_3 in M_O are identical to m_1, m_2, and m_3 in M_C. m_4 in M_C is expressed as m_4 and m_5 in M_O (refer to Appendix H for details). In Study 1, w.r.t. m_4 and m_5 in M_O, 44 and 39 NDPMs are identified, respectively. In Study 2, only 5 NDPMs are identified w.r.t. m_4 in M_C.

Table 6-19 shows the numbers of NDPMs identified for the Five EFSM models in Study 1 and Study 2. The numbers of NDPMs identified w.r.t. additions and deletions that do not represent changes in transitions in M_O and M_C are summed up in columns under the title "Due to additions and deletions".

Table 6-19. Number of NDPMs vs. EMs

Model	# of NDPMs identified in Study 1				# of NDPMs identified in Study 2		
	Due to additions and deletions	Due to additions to express changes in transitions	Due to deletions to express changes in transitions	Total	Due to additions and deletions	Due to changes in transitions	Total
ATM_ Tahat	10	21	13	44	10	8	18
Cruise Control	122	44	39	205	122	5	127
Fuel Pump	10	5	5	25	15	4	19
ISDN	10	5	5	20	10	0	10
TCP-Dialer	10	6	5	21	10	1	11

Table 6-19 indicates that using change in a transition as a new type of EM significantly decreased the number of NDPMs. There are three reasons for this result:

1) For an added or deleted transition, any dependence that the transition was

involved in is an NDPM;

2) For a changed transition, CDs that the transition was involved in are not NDPMs (as we have discussed in Section 3.3). But these CDs are identified as NDPMs for both the addition and the deletion that are used to express the change;
3) Some DDs that the changed transition was involved in may not be identified as NDPMs, since the du-pairs may have not been affected by the changes.

Table 6-20. Type and number of NDPMs identified for changed transitions

NDPMs / EMs		Ag DD	Ag CD	Ag GDD	Ag GCD	Ad DD	Ad CD	Ad GDD	Ad GCD	Total
ATM Tahat	A	10	1	n/a	n/a	9	1	n/a	n/a	21
	D	n/a	n/a	6	1	n/a	n/a	5	1	13
	C	4	n/a	0	n/a	4	n/a	0	n/a	8
Cruise Control	A	17	3	n/a	n/a	16	8	n/a	n/a	44
	D	n/a	n/a	12	3	n/a	n/a	16	8	39
	C	5	n/a	0	n/a	0	n/a	0	n/a	5
Fuel Pump	A	5	0	n/a	n/a	0	0	n/a	n/a	5
	D	n/a	n/a	5	0	n/a	n/a	0	0	5
	C	2	n/a	2	n/a	0	n/a	0	n/a	4
ISDN	A	0	2	n/a	n/a	0	3	n/a	n/a	5
	D	n/a	n/a	0	2	n/a	n/a	0	3	5
	C	0	n/a	0	n/a	0	n/a	0	n/a	0
TCP-Dialer	A	1	3	n/a	n/a	0	2	n/a	n/a	6
	D	n/a	n/a	0	3	n/a	n/a	0	2	5
	C	1	n/a	0	n/a	0	n/a	0	n/a	1

Table 6-20 presents a detailed classification of NDPMs that resulted due to expressing a change in a transition as a pair of deletion and addition versus a new type of EM. Since no Activation DD/CD/GDD/GCD is identified w.r.t. either the pairs of deletion and addition in M_O, or the changes in M_C, we do not have columns for these four NDPMs in Table 6-20. For readability, in the titles we use "A" to refer to "Additions to

express changes in transitions", "D" to refer to "Deletions to express changes in transitions", "C" to refer to "Changes in transitions", "Ag" to stand for "Affecting", and "Ad" to stand for "Affected".

Results of this study show that treating a change in a transition as a new type of EM eliminates the superfluous NDPMs resulting from representing a change in a transition as a pair of deletion and addition. Consequently, the reduction ratio for the given test suites can be increased.

We also observed that in both Study 1 (refer to Table 6-17) and Study 2 (refer to Table 6-18), compared to the other EFSM models, the Cruise Control system has a relatively low reduction ratio, and the Fuel Pump system has a very high reduction ratio. This may be caused by special characteristics of the EFSM models and the original test suites; however, we are not able to determine the causes. We will consider this problem in our future work.

6.4.3 Study 3: Generate RTS Using M_O and M_C

For each of the five EFSM models, we have identified two sets of NDPMs, one for M_O, and the other on for M_C. For each set of NDPMs, we generated an RTS to test them.

In Table 6-21, we list the number of identified NDPMs and generated RTSs for the five EFSM models by using M_O and M_C respectively. There is no non-applicable NDPM identified for the Five EFSM models.

Table 6-21. Measurement of generated RTSs

Model	M_O		M_C	
	Number of NDPMs	**Size of Generated RTS**	**Number of NDPMs**	**Size of Generated RTS**
ATM_Tahat	44	17	18	5
Cruise Control	205	60	127	39
Fuel Pump	25	5	19	5
ISDN	20	12	10	7
TCP-Dialer	21	12	11	7

Details of the NDPMs and the generated RTSs are listed in Appendix H.

In Study 3, we have observed that by using changes to a transition as a type of EM, the number of identified NDPMs is decreased significantly. Consequently, the size of generated RTSs to test the NDPMs is decreased significantly.

In our research, we do not measure the average number of NDPMs covered by each generated test case. There are two reasons:

i) This measurement lacks reliability; for example, different orders of inputting NDPMs to the RTSG can result in different RTSs (as stated in Section 6.2.3).

ii) M_O and M_C realize the same set of changes to the original EFSM models. But superfluous NDPMs identified for M_O are omitted for M_C. Hence, M_C may get a lower score than M_O in measuring the average number of NDPMs covered per generated test case. Thus, measuring the number of NDPMs covered per generated test case will not yield reliable results. It is therefore more important to point out that M_C results in a smaller RTS to test the same set of changes on the EFSM model.

6.5 Summary of Case Studies

In the above three case studies, the proposed RTSR method reduced all of the given RTSs successfully and significantly. Results indicate that all EMs, NDPMs, and IPs that were covered by the given RTSs remain covered by the reduced RTSs. Moreover, in studies on the IBM NTC model, RTS reduction ratio does not decrease fault detection capability in terms of error-prone state coverage and error-prone transition coverage.

Table 6-22 summarizes test suite reduction ratios presented in previous sections w.r.t. the RTSR method. Numbers in parenthesises for the IBM NTC model indicate RTS reduction ratios when we only consider unique test cases. The average ratio of reduction in our case studies is 92.11% (or 89.97% if we only consider unique test cases in studies on the IBM NTC model).

Table 6-22. Summary of case studies for RTS reduction

Model		Percentage of Reduction
IBM NTC model	Study 1	85.71% (68.18%)
	Study 2	90.54% (76.67%)
Simplified ATM	Study 1	91.40%
	Study 2	84.54%
ATM_Tahat	Study 1	95.52%
	Study 2	98.20%
Cruise Control	Study 1	83.15%
	Study 2	84.80%
Fuel Pump	Study 1	99.67%
	Study 2	99.09%
ISDN	Study 1	89.28%
	Study 2	94.56%
TCP-Dialer	Study 1	95.53%
	Study 2	97.61%
Average		92.11% (89.97%)

Table 6-23 summarizes results presented in previous sections w.r.t. RTSG method. Only applicable NDPMs are counted in this table.

Table 6-23. Summary of case studies for RTS generation

Model		Number of EMs	Number of NDPMs	Size of Generated RTS
IBM NTC model		21	106	35
Simplified ATM		5	25	8
ATM_Tahat	M_O	5	44	17
	M_C	4	18	5
Cruise Control	M_O	5	205	60
	M_C	4	127	39

Fuel Pump	M_O	5	25	5
	M_C	4	19	5
ISDN	M_O	4	20	12
	M_C	3	10	7
TCP-Dialer	M_O	4	21	12
	M_C	3	11	7

The results presented above show that the proposed RTSG method can be used to generate an RTS successfully and effectively. The number of EMs plus the number of applicable NDPMs gives an upper bound on the number of test cases in the generated RTS. In our case studies, the number of generated test cases to cover all EMs and NDPMs was far below this bound.

Moreover, in our studies on the IBM NTC model, the RTS generated by our RTSG method had higher fault detection capability in terms of error-prone state coverage and error-prone transition coverage.

For systems other than the IBM model, we have neither the source nor the object code for implementations of these models. In principle, we can implement these models, and execute test cases on the implementations. But this approach is not a rigorous approach, since it is based on an individual implementation that may have unrealistic coding defects. Hence, in this thesis, we were only interested in fault detection capability for the IBM NTC model.

7. Conclusion

7.1 Summary of Contributions

In this thesis, we have presented a regression test suite generation (RTSG) method and a regression test suite reduction (RTSR) method with respect to a given set of EMs (i.e., additions, deletions, and changes of transitions) on an EFSM model based on dependence analysis. We characterized twelve types of new dependencies per modification (NDPMs) to capture the effects of the model on the EMs, the effects of the EMs on the model, and the side-effects caused by the EMs. Then we presented a method to identify NDPMs introduced by each of the three types of EMs.

The proposed RTSG method constructs an RTS to cover all occurrences of NDPMs arising in a given EFSM model due to a given set of EMs. To reduce the size of a given RTS, we presented procedures to construct interaction patterns (IPs) introduced by each of the three types of EMs. The proposed RTSR method eliminates test cases that do not cover unique IPs in a given EFSM model based on a given set of EMs from a given RTS. Both the generated RTS and the reduced RTS still facilitate testing both direct effects of the EMs on the changed parts of the SUT and indirect effects of the EMs on the unchanged parts of the SUT.

Note that the dependence analysis, test case generation procedures and test case reduction procedures in our methods do not rely on a tester's technical skills nor require manual intervention. Moreover, by applying our RTSG method, an RTS can be constructed from the EFSM model and the set of modifications on the model instead of selecting test cases from the original test suite. Thus, overhead for maintaining an RTS is reduced.

In our case studies, we have shown that the RTSG method can be used to efficiently generate an RTS. The generated RTS has relatively few test cases, but still covers all EMs and NDPMs. We have also shown that the RTSR method can reduce the size of a given RTS significantly. The RTSR method also reports all IPs that have been covered by the reduced RTS. Results of our case studies also show that using changes to a transition as a type of EM significantly decreases the number of identified NDPMs. Consequently,

the size of the generated RTS and the reduced RTS required to cover the NDPMs is decreased.

The level of detail in the models may affect the amount of reduction (in RTS reduction) and the size of the generated RTS (in RTS generation) because the numbers of transitions and du-pairs may have an effect on the numbers of NDPMs and IPs.

Note that our definition of control dependence requires an exit state in the EFSM model. If an exit state does not exist in the EFSM model, there will be an initial state instead of the start state and the exit state defined in the EFSM model. A complete transition sequence starts from the initial state and returns to the initial state. We can eaily adopt the definition of control dependence by using the initial state to replace the exit state.

7.2 Directions for Future Research

In this thesis, we concentrated on the EFSM system models. However, this approach can be extended to other models described in formal description languages such as SDL. EFSM and SDL are very popular for modeling complex, state-based, event-driven, real time, and interactive systems, for example, telecommunication, multimedia, distributed and embedded systems and industrial control systems. It would be worthwhile to adapt the presented RTSG and RTSR methods to different types of system models.

Also, it would be interesting to expand this research to further study the fault detection effectiveness of the presented RTSR method to see whether there would be any significant loss of effectiveness when a given RTS is reduced. Moreover, it would be interesting to conduct further studies into the effectiveness of the presented RTSG method to see the fault detection effectiveness of generated RTSs in a variety of contexts. One would like to perform more case studies on large models with realistic fault information to further investigate the fault detection effectiveness of both the RTSR and RTSG methods.

As well, one would be interested in investigating causes of extremely high or extremely low RTS reduction ratios. The high ratios of reduction may very well be associated with the high level of replications in the given RTSs. It would also be

interesting to define and compare criteria other than dependence coverage for the RTSG method.

In this thesis, we only studied RTS reduction by using static dependence analysis, which disregards the repetitions of the same dependence between transitions. More sophisticated IPs may be obtained by using dynamic dependence analysis [Vaysburg+02], which is similar to the method described in this thesis except that repetitions of the same dependence during the traversal of a test case are taken into account. It would be interesting to extend our present work on the RTSR method from static dependence analysis to dynamic dependence analysis.

References

[Abdalla+00] Michel Abdalla, Yuval shavitt, A. Wool, Key management for restricted multicast using broadcast encryption, IEEE/ACM Transactions on Networking (TON), Volume 8, Issue 4, pp. 443 – 454, August 2000.

[Alon+06] N. Alon, D. Moshkovitz, and M. Safra. Algorithmic construction of sets fork-restrictions, ACM Transactions on Algorithms (TALG), v.2 n.2, p.153-177, April 2006.

[Binkley 97] David Binkley, Semantics Guided Regression Test Cost Reduction. IEEE Transaction on Software Engineering, 23(8), August, 1997, pp 498-516.

[Binder 00] Robert V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley Publishing Inc., 2000.

[Bourhfir+97] C. Bourhfir, R. Dssouli, E. M. Aboulhamid, and N. Rico, Automatic executable test case generation for extended finite state machine protocols, in Proceedings of IFIP WG 6.1 10th International Workshop on Testing of Communicating Systems (IWTCS'97), September, 1997, pp. 75-90.

[Briand+02] L.C. Briand, Y. Labiche, G. Soccar, Automating Impact Analysis and Regression Test Selection Based on UML Designs, in Proceedings of the International Conference on Software Maintenance (ICSM'02), pp. 0252, 2002.

[Brown+90] P. A. Brown and D. Hoffman, The Application of Module Regression Testing at TRIUMF, Nuclear Instruments and Methods in Physics Research, Section A, A293 (1-2), August 1990, pp. 337-381.

[Chemli 06] Olfa Chemli, Requirement Based Test Suite Generation Using Dependence Analysis, Master's thesis, University of Ottawa, 2006.

[Chen 02] Yanping Chen, Specification-based Regression Test Selection with Risk Analysis, Master's thesis, University of Ottawa, 2002.

[Chen+02] Yanping Chen, Robert L. Probert, D. Paul Sims, Specification-based Regression Test Selection with Risk Analysis, in Proceedings of CASCON'02, September 2002, pp. 60 – 73.

[Dhar+03] S. Dhar, M.Q. Rieck, S. Pai, and E.J. Kim, "Various Distributed Shortest Path Routing Strategies for Wireless Ad Hoc Networks", in Proceedings of 5th Int. Work.

on Distributed Computing – Lecture Notes in Computer Science, 2918, Springer Verlag, 2003.

[Dhar+04] S. Dhar, M.Q. Rieck, S. Pai and E.J. Kim, "Distributed Routing Schemes for Ad Hoc Networks Using d-SPR Sets", Journal of Microprocessors and Microsystems, Special Issue on Resource Management in Wireless and Ad Hoc Mobile Networks, vol. 28(8), 2004, pp. 427-437.

[Dijkstra 59] E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, Numerische Math. 1, 269-271, 1959.

[Farooq+07] Q. Farooq, M. Z. Z. Iqbal, and Z. I Malik, An Approach for Selective State machine Based Regression Testing, AMOST'07, 2007

[Feige 98] U. Feige, A Threshold of ln n for Approximating Set Cover, Journal of the ACM (JACM), v.45 n.4, p.634 - 652, July 1998.

[Fine+04] S. Fine, S. Ur & A. Ziv., Probabilistic regression suites for functional verification, in Proceedings of IEEE Design Automation Conference DAC' 04, 2004.

[Garey+79] Micheal R. Garey, and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, New York, NY: W.H.Freeman, 1979.

[Guha+99] Sudipto Guha and Samir Khuller, Greedy strikes back: improved facility location algorithms, Journal of Algorithms, 31: 228--248, 1999.

[Gupta+92] R. Gupta, M.j. Harrold, and M.L.Soffa. An Approach to regression testing using slicing, in Proceedings of the conference on software maintenance, 1992.

[Hamlet 06] Dick Hamlet, When only random testing will do, in Proc. of the 1st international workshop on Random testing, July 2006, pp. 1 – 9.

[Harrold+93] Mary Jean Harrold, R. Gupta, and M. L. Soffa, A Methodology for Controlling the Size of a Test Suite, ACM Transactions on Software Engineering and Methodology, 2(3), July 1993, pp. 270-285.

[Harrold+01] Mary Jean Harrold, James A. Jones, Tongyu Li, and Donglin Liang, Regression Test Selection for Java Software, in Proceedings of the ACM Conference on OO Programming, Systems, Languages, and Applications (OOPSLA'01), 2001.

[Hierons+08] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, Using Formal Specifications to

Support Testing, Research Papers, Information Systems and Computing School of Information Systems, Computing and Mathematics, Brunel University, 2008.

[Hong+03] H.S. Hong, S.D. Cha, 1. Lee, O. Sokolsky, and H. Ural, Data Flow Testing as Model Checking, in Proceedings of the 25th IEEE International Conference on Software Engineering (ICSE'03), 2003, pp. 232 – 242.

[Huang+95] Chung-Ming Huang, Yuan-Chuen Lin, Ming-Yuhe Jang, An Executable Protocol Test Sequence Generation Method for EFSM - Specified protocols, International Workshop on Protocol Test Systems (IWPTS), Evry, September1995.

[IEEE 610] ANSI/IEEE standard 610.12-1990: Glossary of Software Engineering Terminology. New York: The Institute of Electrical and Electronic engineers, 1987.

[Johnson 06] David S. Johnson, The NP-completeness column: The many limits on approximation, ACM Transactions on Algorithms (TALG), Volume 2 , Issue 3 (July 2006), 473 – 489, 2006.

[Jones 97] Capers Jones, Software quality: analysis and guidelines for success, International Thompson Computer Press, 1997.

[Juristo+06] Natalia Juristo, Ana M. Moreno, Sira Vegasm, and Martín Solari, In Search of What We Experimentally Know about Unit Testing, IEEE Software, v.23(6), pp. 72-80, November 2006.

[Karp 72] Richard M. Karp, Reducibility Among Combinatorial Problems, in R. E. Miller and J. W. Thatcher (editors): Complexity of Computer Computations. New York: Plenum, 85–103, 1972.

[Korel+98] Bogdan Korel, A. Al-Yami, Automated Regression Test Generation, in Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis, pp. 143 – 152, 1998.

[Korel+02] Bogdan Korel, Luay H. Tahat, Boris Vaysburg, Model Based Regression Test Reduction Using Dependence Analysis, in Proceedings of the International Conference on Software Maintenance (ICSM'02), 2002, pp. 214-223.

[Kung+95] David C. Kung, Jerry Gao, and Pei Hsia, Class Firewall, Test Order, and Regression Testing of Object-Oriented Programs, Journal of Object-Oriented Programming 8(2), May 1995, pp. 51 - 65.

[Leung+91] Harenton K.N. Leung and Lee J. White, A Cost Model to Compare Regression Test Strategies, in Proceedings of the Conference on Software Maintenance, 1991, pp. 201-208.

[Lund+94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems, Journal of the ACM (JACM), v.41 n.5, p.960-981, Sept. 1994

[Mayrhauser+99] Anneliese von Mayrhauser, NING ZHANG, Automated Regression Testing using DBT and Sleuth, Journal of Software Maintenance: Research and Practice, Volume 11, Issue 2, pp. 93 – 116, 1999.

[McGregor+01] John D. McGregor, and David A. Sykes, A Practical Guide to Testing Object-Oriented Software, Addison Wesley Inc., 2001.

[Memon+05] Atif Memon , Adithya Nagarajan , Qing Xie, Automating regression testing for evolving GUI software, Journal of Software Maintenance and Revolution: Research and Practice, v.17 n.1, pp.27-64, January 2005.

[Muccini+05] Henry Muccini, Marcio S. Dias, Debra J. Richardson, Towards software architecture-based regression testing, ACM SIGSOFT Software Engineering Notes, v.30 n.4, 2005.

[Myers 79] Glenford L. Myers, The Art of Software Testing, Wiley-Interscience, 1979.

[Naslavsky+07] Leila Naslavsky, Hadar Ziv, and Debra J. Richardson, Towards Leveraging Model Transformaation to Support Model-based Testing, in Proceedings of 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), pp. 509-512, November 2007.

[Nguyen 01] Hung Q. Nguyen, Testing Applications on the Web, John Wiley & Sons Inc., 2001.

[Probert 05] Robert L. Probert, Software Quality Engineering course notes, University of Ottawa, 2005.

[Rampone 01] S. Rampone, Probability-driven Greedy Algorithms for Set Cover, in Proceedings of VIII SIGEF Congress "New Logics for the New Economy", Naples, Italy, pp. 215-220, September 2001.

[Rapps+85] Sandra Rapps, and Elaine J. Weyuker, Selecting software test data using data flow information. IEEE Transactions on Software Engzneenng, SE-11(4):367-375, April 1985.

[Raz+97] R. Raz and M. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, in Proceedings of STOC 1997, pp. 475-484, 1997.

[Rothermel+96] Gregg Rothermel and Mary Jean Harrold, Analyzing Regression Test Selection Techniques, IEEE Transactions on Software Engineering, V.22, no. 8, August 1996, pp. 529 – 551.

[Rothermel+00] Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia, Regression Test Selection for C++ Software, Journal of Software Testing, Verification, and Reliability, V.10, no. 2, June 2000.

[Sarikaya+86] B. Sarikaya, G. V. Bochmann, Obtaining Normal Form Specifications for Protocols, In Computer Network Usage: Recent Experiences, Elsevier Science Publishers, 1986.

[SIAM 96] Inquiry Board Traces Ariane 5 Failure to Overflow Error, SIAM News, Society for Industrial and Applied Mathematics, Vol. 29, November 1996.

[Stocks+96] Phil Stocks, and David Carrington, A Framework for Specification-Based Testing, IEEE Transactions on Software Engineering, Vol. 22, No. 11, November 1996, pp: 777 - 793.

[Tahat+01] Luay H. Tahat, Atef Bader, Boris Vaysburg, and Bogdan Korel, Requirement-Based Automated Black box Test Generation, in Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, pp.489-495, October 08-12, 2001.

[Tahat 07] Luay H. Tahat, PH.D thesis, Automatic regression test suite prioritization using system models, Illinois Institute of Technology (IIT), 2007.

[Telelogic] Telelogic TAU, http://www.telelogic.com.

[Ural+91] Hasan Ural, and Bo Yang, A Test Sequence Selection Method for Protocol Testing, IEEE Transactions on Communications, Vol.39, No.4, pp.514-523, 1991.

[Vaysburg+02] Boris Vaysburg, Luay H. Tahat, and Bogdan Korel, Dependence Analysis in Reduction of Requirement Based Test Suites, in Proceedings of IEEE International Symposium on Software Testing and Analysis (ISSTA), pp. 107-111, July, 2002.

[Vieira+00] Marlon E. Vieira, Marcio S. Dias, and Debra J. Richardson, Object-Oriented Specification-Based Testing Using UML Statechart Diagrams, ICSE2000 -Workshop on Automated Program Analysis, Testing, and Verification, June 2000.

[White 91] L. J. White, Software Testing and Verification, Elsevier Science B.V., 1991.

[White 96] L. White, Regression testing of GUI event interactions, in Proceedings of the International Conference on Software maintenance, pp. 350-358, 1996.

[Wong+95] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, Effect of Test Set Minimization on Fault Detection Effectiveness, in Proceedings of 17th International Conference on Software Engineering, April 1995, pp. 41 -50.

[Wu+03] Ye Wu, and Jeff Offutt, Maintaining Evolving Component based software with UML, in Proceedings of 7th European on software maintenance and reengineering, IEEE, pp. 133-142, 2003.

[Xie 05] Bo Xie, Master's thesis, Requirement-based Regression Test Suite Reduction Use Dependence Analysis, University of Ottawa, 2005.

[Xie+05] Tao Xie, David Notkin. Checking Inside the Black Box: Regression Testing by Comparing Value Spectra, IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 869 - 883, October, 2005.

[Xu+04] Lihua Xu, Marcio Dias, Debra Richardson. Generating Regression Tests via Model Checking, 28th Annual International Computer Software and Applications Conference (COMPSAC'04), pp. 336-341, 2004.

[Weisstein] Eric W. Weisstein, Bellman-Ford Algorithm, MathWorld. http://mathworld.wolfram.com/Bellman-FordAlgorithm.html

Appendices

Appendix A: Definitions of new dependence per modification (NDPM)

Table APP-1. Definitions of new dependence per modification (NDPM)

NDPMs	History	Definition
Effects of the model on the modification		
affecting data dependence (Affecting DD)	Originally defined in [Xie 05], modified by us	Let t_i be an added or a changed transition, and v be a variable in the modified EFSM model, then there exists an *affecting data dependence* (*Affecting DD*) from t_j to t_i w.r.t. v in the modified EFSM model iff there is a DD(t_j, t_i, v) in the modified EFSM model, and if t_i is a changed transition, changes in t_i include addition of a use of v (before v is possibly defined at t_i), or deletion of a def of v before uses of v at t_i
affecting control dependence (Affecting CD)	Defined in [Xie 05]	Let t_i be an added transition, then there exists an *affecting control dependence* (*Affecting CD*) from t_j to t_i in the modified EFSM model iff there is a CD (t_j, t_i) in the modified EFSM model.
affecting ghost data dependence (Affecting GDD)	Originally defined in [Korel+02] as affecting ghost dependence, used by [Xie 05] as defined	Let t_i be a deleted or a changed transition, and v be a variable in the original EFSM model, then there exists an *affecting ghost data dependence* (*Affecting GDD*) from t_j to t_i w.r.t. v in the modified EFSM model, iff: (1) there is a DD(t_j, t_i, v) in the original EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i w.r.t. v in the

	by [Korel+02], and modified by us	modified EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i in the modified EFSM model, and (4) if t_i is a changed transition, t_i does not have a use of v (before v is possibly defined at t_i) in the modified EFSM model.
affecting ghost control dependence (Affecting GCD)	Introduced by us	Let t_i be a deleted transition, then there exists an *affecting ghost control dependence* (*Affecting GCD*) from t_j to t_i in the modified EFSM model iff: (1) there is a CD(t_j, t_i) in the original EFSM model, (2) the starting state of t_i does not post-dominate the starting state of t_j, and the starting state of t_i post-dominates t_j in the modified EFSM model, and (3) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i in the modified EFSM model.
Effects of the modification on the model		
affected data dependence (Affected DD)	Originally defined in [Xie 05], modified by us	Let t_i be an added or a changed transition, and v be a variable in the modified EFSM model, then there exists an *affected data dependence* (*Affected DD*) from t_i to t_j w.r.t. v in the modified EFSM model iff (1) there is a DD(t_i, t_j, v) in the modified EFSM model, and (2) if t_i is a changed transition, changes in t_i include addition or revision of a statement in which the last def of v appears.
affected control dependence (Affected CD)	Defined in [Xie 05]	Let t_i be an added transition, then there exists an *affected control dependence* (*Affected CD*) from t_i to t_j in the modified EFSM model iff there is a CD(t_i, t_j) in the modified EFSM model.

affected ghost data dependence (Affected GDD)	Originally defined in [Korel+02] as affected ghost dependence, used by Xie as defined by [Korel+02], and modified by us	Let t_i be a deleted or a changed transition, v be a variable, and s be the terminating state of t_i in the original EFSM model, then there exists an *affected ghost data dependence* (*Affected GDD*) from t_i to t_j w.r.t. v in the modified EFSM model iff: (1) there exists a DD(t_i, t_j, v) in the original EFSM model, (2) input interaction $i(t_i)$ occurs and enabling condition $c(t_i)$ evaluates to *TRUE* at the starting state of t_i, (3) there exists a definition-clear path from s to t_j w.r.t. v in the modified EFSM model, and (4) if t_i is a changed transition, t_j has a def of v in the original EFSM model, but does not have a def of v in the modified EFSM model.
affected ghost control dependence (Affected GCD)	Introduced by us	Let t_i be a deleted transition, and s be the terminating state of t_i in the original EFSM model, then there exists an *affected ghost control dependence* (*Affected GCD*) from t_i to t_j in the modified EFSM model iff: (1) there exists a CD(t_i, t_j) in the original EFSM model, (2) input interaction $i(t_i)$ occurs and enabling condition evaluates to *TRUE* at the starting state of t_i, and (3) the starting state of t_j does not post-dominate the starting state of t_i and the starting state of t_j post-dominates s in the modified EFSM model.
Side-effects of the modification		
activation data dependence (Activation DD)	Originally defined in [Korel+02] as activation dependence, named in [Tahat 07] as	Let t_i be an added or a changed transition, and v be a variable in the modified EFSM model, there exists an *activation data dependence* (*Activation DD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff (1) there does not exist a DD(t_j, t_k, v) in the original EFSM model, (2) there exists a definition-clear path from t_j to t_i w.r.t. v in the modified EFSM

	data activation dependence, used by [Xie 05] as defined by [Korel+02]; re-named and modified by us	model, (3) there exists a definition-clear path from t_i to t_k w.r.t. v in the modified EFSM model, (4) t_i does not have a def of v in the modified EFSM model, and (5) if t_i is a changed transition, t_i has a def of v in the original EFSM model.
activation control dependence (Activation CD)	Originally defined in [Tahat 07] as control activation dependence of type A, re-named and modified by us.	Let t_i be an added or a deleted transition in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then there exists an *activation control dependence* (*Activation CD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there does not exists a CD(t_j, t_k) in the original EFSM model, (2) there exists a CD(t_j, t_k) in the modified EFSM model, (3) if t_i is an added transition, i) t_i is in path(s) from s_1 to *Exit* without traversing s_2 in the modified EFSM model, when s_2 post-dominates s_1 in the original EFSM model; or ii) t_i is in path(s) from t_j to s_2 in the modified EFSM model, when there is no path from t_j to s_2 in the original EFSM model, and (4) if t_i is a deleted transition, t_i was in path(s) from t_j to *Exit* without traversing s_2 in the original EFSM model.
activation ghost data dependence (Activation GDD)	Originally defined in [Korel+02] as ghost activation dependence, named in [Tahat 07] as	Let t_i be a deleted or a changed transition, v be a variable in the original EFSM model, then there exists an *activation ghost data dependence* (*Activation GDD*) from t_i to the DD(t_j, t_k, v) in the modified EFSM model iff: (1) there exists a DD(t_j, t_k, v) in the original EFSM model which ceases to exist in the modified EFSM model, (2) there exists a definition-clear path from t_j to the starting state of t_i

	data activation dependence, used by [Xie 05] as defined by [Korel+02]; re-named and modified by us.	w.r.t. v in the original EFSM model, (3) input interaction $i(t_i)$ occurs and enabling condition evaluates to TRUE at the starting state of t_i in the modified EFSM model, (4) there exists a definition-clear path from the terminating state of t_i to t_k w.r.t. v in the original EFSM model, and (5) if t_i is a changed transition, t_i does not have a def of v in the original EFSM model, but has a def of v in the modified EFSM model
activation ghost control dependence (Activation GCD)	Originally defined in [Tahat 07] as control activation dependence of type D, re-named and modified by us.	Let t_i be an added or deleted transition in the modified EFSM model, s_1 be the starting state of t_j, and s_2 be the starting state of t_k, then there exists a *activation ghost control dependence* (*Activation GCD*) from t_i to the CD(t_j, t_k) in the modified EFSM model iff (1) there exists a CD(t_j, t_k) in the original EFSM model which ceases to exist in the modified EFSM model, (2) if t_i is an added transition, t_i is in path(s) from t_j to *Exit* without traversing s_2 in the modified EFSM model, and (3) if t_i is a deleted transition, t_i was in path(s) from s_1 to *Exit* without traversing s_2 in the original EFSM model.

Appendix B: Algorithm to Generate SDG for Modified EFSM Model

Chemli proposed an algorithm for constructing SDG S from a given EFSM E in [Chemli 06], which utilized the following internal data structures for S and E:

An EFSM E is denoted by (*name*, *Start*, *Exit*, {*transition*}) where

name is the name of E,

Start is the start state of E,

Exit is the exit state of E,

transition is (*Label*, s_b, s_e, *VOlist*, *ListOfActions*) where

Label(t) denotes its label,

$s_b(t)$ denotes its beginning state by the index of the state,

$s_e(t)$ denotes its ending state by the index of the state,

Volist(t) denotes {(*OType*, v, *Label*, *OOrder*) | variable v occurs in the transition as *OType* in order *OOrder*} where

OType denotes the type of occurrence which is either a definition denoted *def*, or a c-use denoted *c-use*, or a p-use denoted *p-use*, and *OOrder* denotes the order of occurrence of variable v, which is numbered from top to bottom and from right to left in the textual representation of the transition. Elements in *VOList* set are referenced by an index between 0 and size of *VOList* – 1, corresponding to their order in the set. For example, in transition t_1 of the EFSM shown in Figure 2-1, the variable *b* occurs as a definition in the order of occurrence of 1, the variable *pin* occurs as a definition in the order of occurrence of 2, and the variable *attempts* occurs as a definition in the order of occurrence of 3.

ListOfActions(t) denotes the list of the actions in a transition.

An SDG S for an EFSM E is an adjacency matrix $SDG[t_i, t_k]$, which is either an empty set or a set containing one or more (v, *dependenceType*, v_{ik}) where

t_i, t_j are transitions in E

v denotes variable in V that occurs in t_i as in *Volist*(t_i) and in t_k as in *Volist*(t_k),

dependenceType denotes the type of dependence which is one of the following:

1) a DD: *dd*

2) a CD: *cd*

3) a du-pair (a definition of v, d^v, and a use of v, c^v, form a du-pair if there is a definition-clear path between d^v and c^v) based on a c-use: denoted as *dcu-pair*

4) a du-pair (a definition of v, d^v, and a use of v, p^v, form a du-pair if there is a definition-clear path between d^v and p^v) based on a p-use: denoted as *dpu-pair*

v_{ik} denotes (*oo_def, oo_use*) where

variable v in V occurs in t_i in E as a definition denoted *def* in the order of occurrence *oo_def*, and in t_k in E as a use denoted *use* in the order of occurrence *oo_use*.

Based on Chemli's work, Bo Xie presented an algorithm for generating SDG for the modified EFSM model and algorithms for the dependencies introduced by EMs and used in her thesis [Xie 05]. We slightly change Xie's algorithm. Our modifications to the algorithms are in bold. Texts which are struck through with lines were in Xie's algorithms, but are not in our algorithms. We did not include the parts of the algorithms which are unchanged. Interested reader may refer to [Xie 05] for more details about these algorithms.

Let

R_O be the original EFSM model

S_O be the SDG of R_O

M be $\{m \mid m \text{ is an EM}\}$

m be (*MType*, *Trans*) where

MType denotes the modification type of a transition, which is ~~either~~ an addition, ~~or~~ a deletion, **or a change**

Trans denotes the added **or changed** transition (*Label, Source, Dest, ActionList, VOList*) or denotes the transition number of the deleted transition where

Label denotes the label of the transition,

Source denotes source state by the index of the state,

Dest denotes destination state by the index of the state,

ActionList denotes the list of the actions in this transition,

VOList denotes the variable and occurrence list in this transition.

R_M be the modified EFSM model

Then,

Let R_M be R_O

for each $m \in M$

if m is an addition of a transition t_i

add t_i to R_M

end if

if m is a deletion of a transition t_i

delete t_i from R_M and add a dummy self-loop transition t_{i_dummy} to R_M at the starting state of t_i

end if

if *m* is a change on a transition t_i

replace the *VOlist* and *ListOfActions of* t_i in R_M by the *ActionList and VOList* of *m*

end if

end for

Algorithm SDG_S_M

Input: R_M, S_o, M

Output: S_M

Data Structure:

Let

Def(v) = { t | t is a transition in R_M which defines v},

Use(v) = { t | t is a transition in R_M which uses v},

.....

S_M be a matrix where each entity $S_M[t_i, t_j]$ is in the form of (f_1, f_2, f_3) where

t_i, t_j are transition numbers in R_M

f_1, f_2 denote status information given below, for the diagonal elements of $S_M[t_i, t_j]$

if t_i is an added transition, put *added* in f_1, *empty* in f_2 of $S_M[t_i, t_j]$

if t_i is a deleted transition, put *deleted* in f_1, T_{i_dummy} in f_2 of $S_M[t_i, t_j]$

if t_i is a changed transition, put *changed* in f_1

if t_i is t_{i_dummy}, put *replacing* in f_1, t_i in f_2 in $S_M[t_{i\_dummy}, t_{j\_dummy}]$

all other elements of $S_M[t_i, t_j]$ will have both of these two fields empty i.e., ε

.....

Algorithm Steps:

.....

2. for each $m \in M$

if m is an addition of a transition t_i

...

if m is a deletion of a transition t_i

...

if m is a change of a transition t_i

mark t_i as *changed* in f_1 of $S_M[t_i, t_j]$ and put *empty* in f_2 of $S_M[t_i, t_j]$

.....

4. for each $m \in M$

/* Determine new dependencies introduced by each EM*/

if m is an addition of a transition of t_i

determine Affecting DD(s) from t_j to t_i **and mark as NDPM**

determine Affecting CD(s) **from t_j to t_i and mark as NDPM**

determine Affected DD(s) from t_i to t_j **and mark as NDPM**

determine Affected CD(s) **from t_i to t_j and mark as NDPM**

~~**determine activation dependence(s)**~~

determine Activation DD(s) from t_m to t_n and mark as NDPM

determine Activation CD(s) from t_j to t_i and mark as NDPM

determine Activation GCD(s) from t_j to t_i and mark as NDPM

~~**else /* m is a deletion of a transition T_i */**~~

if m is a deletion of a transition of t_i

search column t_i to identify ~~**and mark**~~ Affecting GDD(s) **from t_j to t_i and mark as NDPM**

search column t_i to identify Affecting GCD(s) from t_j to t_i and mark as NDPM

search row t_i to identify and mark Affected GDD(s) **from t_i to t_j**

search row t_i to identify Affected GCD(s) from t_i to t_j and mark as NDPM

determine Activation CD(s) from t_m to t_n and mark as NDPM

~~**determine ghost activation dependence(s)**~~

determine Activation GDD(s) from t_m to t_n and mark as NDPM

determine Activation GCD(s) from t_m to t_n and mark as NDPM

if m is a change of a transition of t_i

determine Affecting DD(s) from t_j to t_i and mark as NDPM

search column t_i to identify Affecting GDD(s) from t_j to t_i and mark as NDPM

determine Affected DD(s) from t_i to t_j and mark as NDPM

search row t_i to identify Affected GDD(s) from t_i to t_j and mark as NDPM

determine Activation DD(s) from t_m to t_n and mark as NDPM

determine Activation GDD(s) from t_m to t_n and mark as NDPM

5. for each pair of transitions t_j and $t_k \in R_M$, if neither t_j nor t_k is an added transition

for each row t_j in S_M where the first field of $S_M[t_j, t_j]$ is not marked as added or deleted or replacing

for each column t_k in S_M where the first field of $S_M[t_k, t_k]$ is not marked as added or deleted or replacing

Determine Activation DD (s) from t_j to t_k with associated Contributing Modifications, and mark as NDPM

Determine Activation GDD(s) from t_j to t_k with associated Contributing Modifications, and mark as NDPM

Determine Activation CD(s) from t_j to t_k with associated Contributing Modifications, and mark as NDPM

Determine Activation GCD(s) from t_j to t_k with associated Contributing Modifications, and mark as NDPM

Appendix C: Regression Test Suite Generation Algorithm

Test case selection algorithm to be used in Step (4) of RTSG method is as follows: Let *UM* be the set of NDPMs / EMs. Relationships between test cases and NDPMs / EMs are tracked in *TB*.

While *TB* is not Empty:

1) For each test case *ts* (column) in *TB*, calculate $P(ts \mid UM)$ using formula (ii) (refer to section 2.9.2).
2) Select test case with the highest $P(ts \mid UM)$. If there are multiple choices, select the first test case among the choices.
3) Remove rows for all NDPMs / EMs covered by the selected test case, and remove the column for the selected test case from *TB*.

Our regression test suite generation algorithm for a given EFSM *OM* and a set of elementary modifications *M* in transitions in *OM*:

Let

OM be the original EFSM model

M be {*m* | *m* is an EM} on *OM* /*refer to Appendix A for definition of *m* */

Algorithm RTSG

Input: *OM*, *M*.

Output: *RTS*, a set of complete test cases.

Data Structure:

Let

TB be a table where each entity $TB[U_i, A_j]$ has Value 0 or 1.

Algorithm Steps:

1. Generate *MM*, the modified EFSM model after M is applied, from *OM* and *M*
2. Use SDG_S_M algorithm to generated SDG for *MM* (refer to Appendix A). During generation of SDG_M, NDPMs associated to all EMs in *M* are identified. Put all NDPMs into a set UNTESTED.

3. Initialize table *TB* which is used to keep track of relationships between dependencies (all NDPMs, and CDs for Activation_CD or Activation_GCD) and EMs to be covered and generated test cases.

   /*Make sure only one test case will be generated for one group of Activation_CD or Activation_GCD to cover the control dependence.*/

   for each group of Activation_CD or Activation_GCD $\in$ UNTESTED that are associated to the same CD

     Append the CD to UNTESTED

   for each dependence $d \in$ UNTESTED

     Add a row in *TB* to represent *d*

   for each $m \in M$

     Add a row in *TB* to represent *m*

4. Generates a set of complete test cases (sequences of transitions which start at *Start*, and end at *Exit* of the EFSM) to cover each dependence in UNTESTED and each EM in *M* at least once.

   Generate the shortest executable preamble $PR(t_i)$ for each transition $t_i \in MM$

   Generate the shortest executable postamble $PT(t_i)$ for each transition $t_i \in MM$

   for each $d \in$ UNTESTED

     /* Instruction to get activating path is proposed in Section 4.1.1.*/

     Generate a shortest executable activating path *ap* for *d* according to the type of *d*

     $TC = PR(t_i) + ap + PT(t_j)$

     /* Instruction to determine if a test case is expected to be failed is proposed in Section 4.1.2.*/

     if we do not expect the *TC* to be failed

       Check *TC*'s executability

       if *TC* is not executable

         /*Try to find an executable test case. If an executable test case is not found, return NULL. */

         Make_TC_executable(*TC*)

     if a *TC* is not NULL

       $TestSuite \leftarrow TestSuite \cup \{TC\}$

Add a column *c* to *TB* to represent *TC*

/*Track relationship between *TC* with dependencies or EMs in table TB. */

for each row $r \in TB$

if the dependence or EM reprensted by *r* is covered by *TC*

Assign Value 1 to cell (*r*, *c*)

Remove the dependence represented by *r* from set UNTESTED

Marked associated EM of the dependence represented by *r* or the EM represted by *r* as COVERED

else

Assign Value 0 to cell (*r*, *c*)

else

/*Not executable test case can be found, *d* is not testable.*/

Move *d* to UNTESTABLE

for each $m \in M$ which is not marked as TESTED or UNTESTABLE

Generate a path from *Start* to *Exit* that traverse *m* as a complete test case *TC*

Check *TC*'s executability

While *TC* is not executable

Find other path from *Start* to *Exit* that traverse *m* as a complete test case *TC*

Check *TC*'s executability

if an executable *TC* exist

$TestSuite \leftarrow TestSuite \cup \{TC\}$

Add a column *c* to *TB* to represent *TC*

for each row $r \in TB$ that represents an EM

if the EM reprensted by *r* is covered by *TC*

Assign Value 1 to cell (*r*, *c*), and marked the EM as COVERED

else

Assign Value 0 to cell (*r*, *c*), and marked the EM as UNTESTABLE

5. Call the probability-driven greedy algorithm SCP_Greedy (to select a minimum set of complete test cases from the generated test cases that cover all identified NDPMs and EMs in *M*.

FinalRTS= SCP_Greedy (*TB*)

Return *FinalRTS*

Algorithm Make_TC_executable

Input: *DUT*, dependence under test (t_i, t_j) or (t_i, t_j, v);

EMUT, elementary modification under test

Output: if there exists an executable test case *TC* covers *DUT*, return *TC*; otherwise return NULL

Algorithm steps:

1. Look for paths that does not contain any predicate or the shortest path whose predicate will not create a contradiction with the predicates of the other two paths first

Mark TC as not executable

sub_path1← the shortest executable preamble of t_i $PR(t_j)$

sub_path2 ← the shortest executable activating path for DUT

sub_path3 ← the shortest executable postamble of t_j $PT(t_j)$

if sub_path1 contains predicates that may create a contradiction with predicates of the other two paths

Mark sub_path1 as CHANGABLE

if sub_path2 contains predicates that may create a contradiction with predicates of the other two paths

Mark sub_path2 as CHANGABLE

if sub_path3 contains predicates that may create a contradiction with predicates of the other two paths

Mark sub_path3 as CHANGABLE

2. Try different combination of the three sub paths to construct an executable complete test case

if sub_path1 is CHANGEABLE

While *TC* is not executable and exist another executable $PR(t_j)$

sub_path1 ← another executable $PR(t_j)$

TC = sub_path1 + sub_path2 + sub_path3

Check *TC*'s executability

```
    if TC is not executable
      if sub_path2 is CHANGEABLE
        While TC is not executable and exists another activating path for DUT
          sub_path2 ← another executable activating path for DUT
          TC = sub_path1 + sub_path2 + sub_path3
          Check TC's executability
          While TC is not executable and sub_path3 is CHANGEABLE and exist
            another executable PT(t_j)
            sub_path3 ← another executable PT(t_j)
            TC = sub_path1 + sub_path2 + sub_path3
            Check TC's executability
      else
        While TC is not executable and sub_path3 is CHANGEABLE and exist
          another executable PT(t_j)
          sub_path3 ← another executable PT(t_j)
          TC = sub_path1 + sub_path2 + sub_path3
          Check TC's executability
  else
    if sub_path2 is CHANGEABLE
      While TC is not executable and exists another activating path for DUT
        sub_path2 ← another executable activating path for DUT
        TC = sub_path1 + sub_path2 + sub_path3
        Check TC's executability
        While TC is not executable and sub_path3 is CHANGEABLE and exist another
          executable PT(t_j)
          sub_path3 ← another executable PT(t_j)
          TC = sub_path1 + sub_path2 + sub_path3
          Check TC's executability
    else
      While TC is not executable and sub_path3 is CHANGEABLE and exist another
        executable PT(t_j)
```

sub_path3 ← another executable $PT(t_j)$

TC = sub_path1 + sub_path2 + sub_path3

Check TC's executability

3. Return an executable test case or NULL

if TC is executable

Return TC

else

Return NULL

Algorithm SCP_Greedy

Input: TB, a table represents relationship between universal U and a family A of subsets of U. Each row in TB represents a $U_i \in U$, and each column represents an $A_j \in A$. A cell $[U_i, A_j]$ has value 1, if A_j includes (covers) U_i; otherwise, the cell has value 0.

Output: A set of selected $A_i \in A$ whose union is U (A set of column ids of TB is returned)

Algorithem steps:

1. Calculate Possibility of each unselected column in TB. Keep tracking of the column which has the maximum probability.

$p_{max} = 0$ //A parameter to keep the maximum probability for all unselected column.

/*Calculate Possibility P_c for each unselected column w.r.t. rows in TB*/

for each column $c \in TB$

row_counter = 0

SUMPc = 0

/*Calculate probability of the column*/

for each row $r \in TB$

r_counter++

If cell $(r, c) == 1$

Count the number of columns in TB that cover the row r as Nr

$Pc\_r = 1 / Nr$

else

$Pc\_r = 0$

$SUMPc = SUMPc + Pc\_r$

$Pc = SUM\ Pc\ /\ r\_counter$

/*Determine if Pc is the maximum value so far. Keep tracking the maximum probability and the column. If there are multiple choice, the first column with the maximum value will be tracked*/

if $Pc > p_{max}$

$p_{max} = Pc$

$select\_column = c$

2. Select the column that has the maximum probability (referred by *selected_column*), and remove all covered rows from *TB*.

SelectedColumn ← *SelectedColumn* ∪ {*select_column*}

Remove *select_column* from *TB*

for *row* $r \in TB$

if cell (r, *select_column*) == 1

Remove r from *TB*

3. Repeat Step 1 and 2 untill *TB* is Empty
4. Return all selected columns.

Complexity of Algorithm RTSG:

Let n_t be the number of transitions in *OM*, and n_m be the number of EMs in *M*. Then the number of transitions in *MM* is not greater than $n_t + n_m$. Generally, $n_m << n_t$.

Step 1: The complexity of this step is: n_m

Step 2: Let u be the maximum number of variable usages appear in one transition in *MM*. Then there are up to $u*(n_t + n_m)^2$ DDs, and up to $(n_t + n_m)\ (n_t + n_m - 1)$ CDs in SDG_M. Generally, u is proportional to n_t. The complexity of this step is:

$u*(n_t + n_m)^2 + (n_t + n_m)\ (n_t + n_m - 1) \Rightarrow O(n_t^3)$

Step 3: The complexity of this step is:

$(n_t + n_m - 1)\ (n_t + n_m - 2) + u*(n_t + n_m)^2 + (n_t + n_m)\ (n_t + n_m - 1) + n_m \Rightarrow O(n_t^3)$

Step 4: The complexity of this step is:

$(n_t + n_m) + (n_t + n_m) + (u*(n_t + n_m)^2 + (n_t + n_m)\ (n_t + n_m - 1))\ *\ (6 + 1.5\ *\ (\ u*(n_t + n_m)^2 + (n_t + n_m)\ (n_t + n_m - 1) + n_m)) + n_m\ *\ (5 + n_m) \Rightarrow O(n_t^6)$

Step 5: Let c be the number of columns, and r be the number of rows in table TB. Then both c and r are not greater than $u^*(n_t + n_m)^2 + (n_t + n_m)(n_t + n_m - 1) + n_m$. In RTSG, generally, $c << r$. The complexity of this step is:

$$\sum_{i=1}^{r} (3 + c(4 + 3.5\, i)) \Rightarrow O(r^2) \Rightarrow O(n_t^6)$$

Therefore, in the worst-case, the complexity of RTSG can be $O(n_t^6)$.

Appendix D: Generating IPs for a Given Test Case

We need to modify Xie's algorithm in [Xie 05] for generating interaction patterns for a given test case due to our modifications on their definitions of the new dependencies and due to the definitions we introduce. We only present changed portion of this algorithm. Please refer to [Xie 05] for more details. Our modifications to the algorithms are in bold. Texts which are struck through with lines were in Xie's algorithms, but are not in our algorithms.

Data Structure:

$G = (N, E)$ where

$N = \{n_t \mid n_t$ is a node representing transition t in *rts*, n_0 denotes the entry node of G, n_{end} denotes the exit node of G, and n_{tut} denotes the transition t_{tut} node in $G\}$ and

$E = \{e \mid e = (n_s, n_d, type), n_s, n_d \in N\}$ where

e is an edge representing a dependence between two transitions

n_s and n_d are the source node and Param4 node of e respectively, and

type denotes the type of dependence which is one of the following:

1) *data dependence, affecting data dependence, or affected data dependence: dd*
2) *control dependence, affecting control dependence, or affected control dependence: cd*
3) *affecting ghost data dependence: affecting_gdd*
4) ***affecting ghost control dependence: affecting_gcd***
5) *affected ghost data dependence: affected_gdd*
6) ***affected ghost control dependence: affected_gcd***
7) *activation **data** dependence: ~~**ad**~~ **a_dd***
8) ***activation control dependence: a_cd***
9) *ghost activation **data** dependence: ~~**gad**~~ **ga_dd***
10) ***ghost activation control dependence: ga_cd***

Input: S_M, *tut*(transition under test), *rts* (a regression test sequence)

……

Algorithm Steps:

1.During traversal of a test case $rts \in RTS_{tut}$, construct sub-SDG G as a graph induced by a set E of dependence edges, i.e., $G[E]$, E is a subset of the set of dependence edges in S_M as follows:

.....

Loop from the last transition (say t_k) of *rts* to the second transition of *rts*

Loop from the second last transition of *rts* (say t_j) to the first transition of *rts*

if there is one or more *dd* w.r.t. v in $S_M[t_j, t_k]$ and there is no *dd*/~~***ad***~~ ***a_dd*** edge w.r.t. v incoming to t_k

add a ——▸ edge (t_j, t_k) to E

if there is one or more *cd* in $S_M[t_j, t_k]$ and there is no *cd*/***a_cd*** edge incoming to t_k

add a ------▸ edge (t_j, t_k) to E

if there is one or more *affecting_gdd* w.r.t. v in $S_M[t_j, t_k]$ and t_k is the deleted **or changed** transition related to t_{tut} and there is no *affecting_gdd* edge w.r.t. v incoming to t_k

add a —~~gg~~→ edge (t_j, t_k) to E

if there is one or more *affecting_gcd* w.r.t. v in $S_M[t_j, t_k]$ and t_k is the deleted or changed transition related to t_{tut} and there is no *affecting_gcd* edge w.r.t. v incoming to t_k

add a ---gg-▸ edge (t_j, t_k) to E

if there is one or more *affected_gdd* w.r.t. v in $S_M[t_j, t_k]$ and t_j is the deleted **or changed** transition related to t_{tut} and there is no *affected_gdd* edge w.r.t. v incoming to t_k

add a —~~dg~~→ edge (t_j, t_k) to E

if there is one or more *affected_gcd* w.r.t. v in $S_M[t_j, t_k]$ and t_j is the deleted or changed transition related to t_{tut} and there is no *affected_gcd* edge w.r.t. v incoming to t_k

add a ---dg-▸ edge (t_j, t_k) to E

if there is one or more ~~***ad***~~ ***a_dd*** w.r.t. v from t to (t_j, t_k) in $S_M[t_j, t_k]$ and t is the added **or changed** transition related to t_{tut} and there is no ~~***ad***~~ ***a_dd***/*dd* edge w.r.t. v incoming to t_k

add a ——▸ edge (t_j, t_k) a ——◆ edge from t_i to edge (t_j, t_k) to E

if there is one or more *a_cd* w.r.t. *v* from *t* to (t_j, t_k) in $S_M[t_j, t_k]$ and *t* is the added or changed transition related to t_{tut} and there is no *a_cd/cd* edge w.r.t. *v* incoming to t_k

add a ------► edge (t_j, t_k) a ———◆ edge from t_i to edge (t_j, t_k) to *E*

if there is one or more ~~***gad***~~ ***ga_dd/ga_cd*** w.r.t. *v* from t_i to (t_j, t_k) in $S_M[t_j, t_k]$ and t_i is the deleted **or changed** transition related to t_{tut} and there is no ~~***gad***~~ ***ga_dd/ga_cd*** edge w.r.t. *v* incoming to t_k

add a —g—◆ edge from t_i to edge (t_j, t_k) to *E*

The resulting graph is *G*[*E*]

2. In *G*[*E*], identify dependencies that "affect" t_{tut} to obtain an Affecting Interaction Pattern:

~~**if t_{tut} is an added transition**~~

traverse backwards from t_{tut}, and mark all traversed *dd*(s), *cd*(s), ***affecting_gdd*(s),** ***affecting_gcd*(s),** in *G*[*E*]

remove all unmarked dependencies from *G*[*E*] to obtain an Affecting Interaction Pattern

~~**else /* t_{tut} is a dummy transition t_{i_dummy}, corresponding to deleted transition t_i */**~~

~~**traverse backwards from t_i through the affecting_gdd(s), and mark all traversed dd(s) and cd(s) in G[E]**~~

~~**remove all unmarked dependencies from G[E] to obtain an Affecting Interaction Pattern**~~

3. In *G*[*E*], identify dependencies that are "affected" by t_{tut} to obtain an Affected Interaction Pattern:

~~**if t_{tut} is an added transition**~~

traverse forward from t_{tut}, and mark all traversed *dd*(s), *cd*(s), ***affected_gdd*(s),** ***affected_gcd*(s),** in *G*[*E*]

remove all unmarked dependencies from *G*[*E*] to obtain an Affected Interaction Pattern

~~**else /* t_{tut} is a dummy transition t_{i_dummy}, corresponding to deleted transition t_i */**~~

~~**traverse forward from t_i through the affected_gdd(s), mark all traversed dd(s) and cd(s) in G[E]**~~

~~remove all unmarked dependencies from G[E] to obtain an Affected Interaction Pattern~~

4. In $G[E]$, identify dependencies that are "affected" by t_{tut} to obtain a Side-Effect Interaction Pattern:

~~if t_{tut} is an added transition~~

traverse forward from t_{tut} through the ~~activation dependencies~~ **Activation_dd, Activation_cd, Ghost_Activation_dd, Ghost_Activation_cd**, and mark all traversed *dd*(s) and *cd*(s) in $G[E]$

remove all unmarked dependencies from $G[E]$ to obtain a Side-Effect Interaction Pattern

~~else /* t_{tut} is a dummy transition t_{i_dummy}, corresponding to deleted transition t_i */~~

~~traverse forward from t_i through the ghost activation dependencies, and mark all traversed *dd*(s) and *cd*(s) in $G[E]$~~

~~remove all unmarked dependencies from $G[E]$ to obtain a Side-Effect Interaction Pattern~~

Algorithm Interaction_Patterns

Let *uniqueT* = $\{t \mid t$ is a transition in *rts*$\}$

node(*i*) denote a node whose index is equal to *i* in *G*,

finish[] denote an array of integers.

Algorithm Steps:

uniqueT ← ∅, sub-SDG *G* ← (∅,∅)

1. for each transition $t \in rts$ from the first transition to the last transition

/* if $t \in$ *rts* is a dummy transition due to a deleted transition t_i then use t_i instead of t */

if $t \notin$ *uniqueT*

uniqueT ← *uniqueT* ∪ $\{t\}$

sort the elements of *uniqueT* in ascending order of transitions' labels.

/* Each element of sorted *uniqueT* can be referred by its index e.g., *uniqueT*[0] denotes the first element of *uniqueT*. */

/* *construct* sub-SDG *G* */

/* *First, create nodes in* sub-SDG *G* */

.....

/* *Secondly, create edges in* sub-SDG *G* */

Loop from the last transition to the second transition of *rts*

rts[*k*] ← the current transition in *rts*

n_k ← the node representing the transition *rts*[*k*] in *N*

Loop from the second last transition of *rts* to the first transition of *rts*

rts[*j*] ← the current transition in *rts*

n_j ← the node representing the transition *rts*[*j*] in *N*

if ((there is a *dd* w.r.t. *v* from *rts*[*j*] to *rts*[*k*] in S_M) and (there is no ***dd*/~~*ad*~~ *a_dd*** edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, dd)\}$

if ((there is a *cd* from *rts*[*j*] to *rts*[*k*] in S_M) and (there is no ***cd*/*a_cd*** edge incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, cd)\}$

if ((there is an *affecting_gdd* w.r.t. *v* from *rts*[*j*] to *rts*[*k*] in S_M) and (*rts*[*k*] is the deleted **or changed** transition related to T_{tut}) and (there is no *affecting_gdd* edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, affecting\_gdd)\}$

if ((there is an *affecting_gcd* w.r.t. *v* from *rts*[*j*] to *rts*[*k*] in S_M) and (*rts*[*k*] is the deleted or changed transition related to T_{tut}) and (there is no *affecting_gcd* edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, affecting\_gcd)\}$

if ((there is an *affected_gdd* w.r.t. *v* from *rts*[*j*] to *rts*[*k*] in S_M) and (*rts*[*j*] is the deleted **or changed** transition related to T_{tut}) and (there is no *affected_gdd* edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, affected\_gdd)\}$

if ((there is an *affected_gcd* w.r.t. *v* from *rts*[*j*] to *rts*[*k*] in S_M) and (*rts*[*j*] is the deleted or changed transition related to T_{tut}) and (there is no *affected_gcd* edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k, affected\_gcd)\}$

if ((there is an ***ad a_dd*** w.r.t. v from t to (rts[j], $rts[k]$) in S_M) and (t is the added **or changed** transition related to t_{tut}) and (there is no ***ad a_dd/dd*** edge w.r.t. v incoming to n_k in G))

$E \leftarrow E \cup \{(n_j, n_k,$ ***ad a_dd***$)\}$

/* Since this ***ad a_dd*** is related to t_{tut}, we don't need to add an addition edge from n_i to (n_j, n_k) here. We know this ***ad a_dd*** is activated by t_{tut} */

if ((there is an *a_cd* w.r.t. *v* from *t* to (rts[*j*], *rts*[*k*]) in S_M) and (*t* is the added or changed transition related to t_{tut}) and (there is no *a_cd/cd* edge w.r.t. *v* incoming to n_k in *G*))

$E \leftarrow E \cup \{(n_j, n_k,$ ***a_cd***$)\}$

/* Since this *a_cd* is related to t_{tut}, we don't need to add an addition edge from n_i to (n_j, n_k) here. We know this *a_cd* is activated by t_{tut} */

if ((there is a ***gad ga_dd/ga_cd*** w.r.t. v from t_i to ($rts[j]$ to $rts[k]$) in S_M) and (t_i is the deleted transition **or changed** related to t_{tut}) and (there is no ***gad ga_dd/ga_cd*** edge w.r.t. v incoming to n_k in G))

$E \leftarrow E \cup \{(n_j, n_k,$ ***gad ga_dd/ga_cd***$)\}$

/* Since this ***gad ga_dd/ga_cd*** is related to t_{tut}, we don't need to add an addition edge from n_i to (n_j, n_k) here. We know this ***gad ga_dd/ga_cd*** is activated by t_{tut} */

2. .....

3. .....

4. In the sub-SDG G, identify dependencies that are "affected" by t_{tut} to obtain a Side-Effect Interaction Pattern:

~~**if t_{tut} is an added transition**~~

/* *traverse forward* from n_{tut} through the marked activation **data/control** dependencies **or ghost activation data/control dependencies** to n_{end} in G, mark traversed nodes and edges. After the traversal, remove all unmarked nodes and edges in G */

mark n_{tut} in N

for each ~~ad~~ **Activation_dd /Activation_cd /Ghost_Activation_dd / Ghost_Activation_cd** edge e in E

mark the ~~ad~~ edge e in E and mark n_s and n_d in N

$i \leftarrow$ the index of n_d

$j \leftarrow$ the index of n_{end} *in* G

/ initialize the value of each element in finish[] equal to 0 */*

l = the index of the last node in N

for $i = 0$ to l

finish[i] $\leftarrow 0$

call traverse_forward(i, *finish*[], j)

remove all unmarked nodes and unmarked edges in G

The resulting graph G is a Side-Effect Interaction Pattern.

~~**else /* t_{tut} is a dummy transition t_{i_dummy}, corresponding to deleted transition t_i, traverse forward from n_{tut} through the marked ghost activation dependencies to n_{end} in G_s, mark traversed nodes and edges. After the traversal, remove all unmarked nodes and edges in G */**~~

~~**mark n_{tut} in N**~~

~~**for each gad edge e in E**~~

~~**mark the gad edge e in E and mark n_s and n_d in N**~~

~~**i ← the index of n_d**~~

~~**j ← the index of n_{end} in G**~~

~~**/* initialize the value of each element in finish[] equal to 0 */**~~

~~**l = the index of the last node in N**~~

~~**for i = 0 to l do**~~

~~**finish[i] ← 0**~~

~~**call traverse_forward(i, finish[], j)**~~

~~**remove all unmarked nodes and unmarked edges in G**~~

~~**The resulting graph G is a Side-Effect Interaction Pattern.**~~

Appendix E: Regression Test Suite Reduction Algorithm

Test case selection algorithm to be used in Step (3) of RTSR method is as follows: Let *UIP* be the set of IPs. Relationships between test cases and IPs are tracked in *TB*.

While *TB* is not empty:

1) For each test case *ts* in *TB*, calculate *P(ts | UIP)* using formula (ii) (refer to section 2.9.2).
2) Select test case with the highest *P(ts | UIP)*. If there are multiple choices, select the first test case among the choices.
3) Remove rows for all IPs covered by the selected test case, and remove the column for the selected test case from *TB*.

Let

TUT denotes a set of transitions correponding to all EMs in *M*

RTS denotes a set of regression test cases

OM denotes the original EFSM representing the SUT.

Algorithm RTSR

Input: *OM, RTS, TUT*

Output: *RRTS, PatternSet*

Data Structure:

Let

tut denotes a transition under test in *TUT* related to an EM in *M*

rts denotes a regression test case in *RTS*

ip denotes an interaction pattern

Pattern1Set be {*Pattern1* | *Pattern1* is an Affecting IP}

Pattern2Set be {*Pattern2* | *Pattern2* is an Affected IP}

Pattern3Set be {*Pattern3* | *Pattern3* is a Side-effect IP}

WorkingSet be {Pattern | Pattern is an IP}

TB be a table where each entity $TB[U_i, A_j]$ has Value 0 or 1.

Algorithm Steps:

1. Generate *MM*, the modified EFSM model after *M* is applied, from *OM* and *M*
2. Use *SDG_S*$_M$ algorithm to generated SDG for *MM* (refer to Appendix A). During generation of *SDG*$_M$, NDPMs associated to all EMs in *M* are identified. .
3. Initialize table *TB*, which is used to keep track of relationships between calculated IPs and EMs with test cases in *RTS*.

   for each *tut* ∈ *TUT*

     Add a row to *TB* to represent *tut*

   for each *rts* ∈ *RTS*

     Add a column to *TB* to represent *rts*
4. Calculated IPs for each test case in *RTS* w.r.t. each EM in *M*

   for each *tut* ∈ *TUT*

     for each *rts* ∈ *RTS*

       if *rts* traverses *tut*

         /*The *tut* is traversed. Track the coverage in *TB* */

         Locate row *rtut* in *TB* that represents *tut*

         Assign Value 1 to cell (*rtut*, *rts*)

         /*Obtain Affecting IPs and Affected IPs*/

         Using Algorithm Interaction_Patterns to obtain an Affecting IP and an Affected IP, namely *IP*1, *IP*2

         /*Track relationship between *IP*1, *IP*2 and *rts* in *SCPTB*/

         if *IP*1 is not in *Pattern*1*Set*

           *Pattern*1*Set* ← *Pattern*1*Set* ∪ {*IP*1}

           Add a row *r* to *TB* to represent *IP*1

           Assign Value 1 to cell (*r*, *rts*)

         else

           Locate row *r* in *TB* that represents *IP*1

           Assign Value 1 to cell (*r*, *rts*)

         if *IP*2 is not in *Pattern*2*Set*

           *Pattern*1*Set* ← *Pattern*2*Set* ∪ {*IP*2}

           Add a row *r* to *TB* to represent *IP*2

           Assign Value 1 to cell (*r*, *rts*)

else

Locate row *r* in *TB* that represents *IP*2

Assign Value 1 to cell (*r*, *rts*)

/*Obtain Side-effect Interaction Patterns. */

Using Algorithm Interaction_Patterns to obtain side-effect interaction pattern, namely *IP*3

/*Track relationship between *IP*3 and *rts* in *TB*/

if *IP*3 is not in *Pattern3Set*

Pattern3Set ← *Pattern3Set* ∪ {*IP*3}

Add a row *r* to *TB* to represent *IP*3

Assign Value 1 to cell (*r*, *rts*)

else

Locate row *r* in *TB* that represents *IP*3

Assign Value 1 to cell (*r*, *rts*)

5. Call the probability-driven greedy algorithm SCP_Greedy (refer to Appendix B for SCP_Greedy) to select a minimum set of test cases from RTS that covers all calculated IPs

FinalRTS = SCP_Greedy (*TB*)

Return *FinalRTS*

Complexity of Algorithm RTSR:

Let n_t be the number of transitions in *OM*, n_m be the number of EMs in *M*. Then the number of transitions in *MM* is not greater than $n_t + n_m$. Generally, $n_m << n_t$. Let n_{tc} be the number of test cases in the given RTS. Generally, $n_m << n_{tc}$.

Step 1: The complexity of this step is: n_m

Step 2: Let u be the maximum number of variable usages appear in one transition in *MM*. Then there are up to $u*(n_t + n_m)^2$ DDs, and up to $(n_t + n_m)(n_t + n_m - 1)$ CDs in SDG_M. Generally, u is proportional to n_t. The complexity of this step is:

$u*(n_t + n_m)^2 + (n_t + n_m)(n_t + n_m - 1) \Rightarrow O(n_t^3)$

Step 3: The complexity of this step is:

$n_m + n_{tc} \Rightarrow O(n_{tc})$

Step 4: The complexity of this step is:

$3^* n_m^* n_{tc} * (4 + 4.5) \Rightarrow O(n_{tc})$

Step 5: Let c be the number of columns, and r be the number of rows in table TB. Then $c = n_{tc}$, $r \leq 3^* n_m^* n_{tc} + n_m$. In the worst-case, each selected column covers one row in table TB. The complexity of this step is:

$$\sum_{i=1}^{r} (3 + (c - i)(4 + 3.5\, i)) \Rightarrow O(c * r^2) \Rightarrow O(n_{tc}^3)$$

Therefore, in the worst-case, the complexity of RTSR is the greater of $O(n_t^3)$ and $O(n_{tc}^3)$.

Appendix F: IBM NTC Model

To avoid exposing IBM confidential data, names of features, parameters, and constant values have been replaced by substitution in Appendix F.

Table APP-2. Set of EMs applied to NTCv1

EM #	Details
m_1	Add t87: <s20, s1, FEAT4(Param1), ε , ε, EAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param1); FEAT6(Param9)>
m_2	Add t88: <s4, s21, FEAT5(Param14, Param15), Param16 == Value4, ε, ε>
m_3	Add t89: <s21, s4, BACK(), ε, ε, ε>
m_4	Add t90: <s21, s1, CANCEL(), ε, ε, ε>
m_5	Add t91: <s21, s1, FEAT4(Param13), ε, ε, FEAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param13); FEAT6(Param9)>
m_6	Add t92: <s20, s6, FEAT5(Param1), ε, FEAT2(Param16), ε>
m_7	Add t93: <s6, s20, BACK(), Param16 == Value8, ε, ε>
m_8	Add t94:<s21, s5, FEAT5(Param13), ε, ε, ε>
m_9	Add t95: <s5, s21,BACK(), Param16 == Value4, ε, ε>
m_{10}	Add t96:<s4, s20, FEAT5(Param14, Param15), Param16 == Value8, ε, ε>
m_{11}	Add t97: <s20, s4, BACK(), ε, ε, ε>
m_{12}	Add t98: <s20, s1, CANCEL(), ε, ε, ε>
m_{13}	Change t12 from: <s5, s6, FEAT5(Param10), (Param16 == Value6 or Param16 == Value5), FEAT2(Param16), ε> to: <s5, s6, FEAT5(Param10), (Param16 == Value6 or Param16 == Value7 or Param16 == Value5 or Param16 == Value4), FEAT2(Param16), ε> {*Note: FEAT5(Param10) is modified*}

m_{14}	Change t15 from:<s4, s5, FEAT5(Param14, Param15), (Param16 == Value6) or (Param16 == Value1 and Param11 == true) or (Param16 == Value2) or (Param16 == Value3), FEAT3(Param16, Param11), ε> to: <s4, s5, FEAT5(Param14, Param15), (Param16 == Value6) or (Param16 == Value7) or (Param16 == Value1 and Param11 == true) or (Param16 == Value2) or (Param16 == Value3), FEAT3(Param16, Param11), ε>
m_{15}	Change t16 from: <s5, s9, FEAT5(Param10), Param16 == Value1 and Param11 == true) or (Param16 == Value2), ε, ε> to: <s5, s9, FEAT5(Param10), Param16 == Value1 and Param11 == true) or (Param16 == Value2), ε, ε> *{Note: FEAT5(Param10) is modified}*
m_{16}	Change t31 from: <s6, s1, FEAT4(Param3), ε, ε, FEAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param10, Param17, Paman2, Param7, Param6, Param12, Param5, Param8, Param18, Param3); FEAT6(Param9)> to: <s6, s1, FEAT4(Param3), ε, ε, FEAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param10, Param17, Paman2, Param7, Param6, Param12, Param5, Param8, Param18, Param1, Param13, Param3); FEAT6(Param9)>
m_{17}	Change t39 from:<s17, s5, FEAT5(Param18), ε, FEAT3(Param16), ε> to: <s17, s5, FEAT5(Param18), ε, FEAT3(Param16), ε> *{Note: FEAT3(Param16) is modified}*
m_{18}	Change t44 from: <s5, s10, FEAT5(Param10), Param16 == Value3, ε, ε> to: <s5, s10, FEAT5(Param10), Param16 == Value3, ε, ε>

	{Note: FEAT5(Param10) is modified}
m_{19}	Change t78 from: <s5, s4, BACK(), (Param16 == Value6) or (Param16 == Value1 and Param11 == true) or (Param16 == Value2) or (Param16 == Value3), ε, ε> to: <s5, s4, BACK(), (Param16 == Value6) or (Param16 == Value7) or (Param16 == Value1 and Param11 == true) or (Param16 == Value2) or (Param16 == Value3), ε, ε>
m_{20}	Change t81 from: <s6, s5, BACK(), Param16 == Value5 or Param16 == Value6, ε, ε> to: <s6, s5, BACK(), Param16 == Value5 or Param16 == Value6 or Param16 == Value7 or Param16 == Value4, ε, ε>
m_{21}	Change t84 from: <s5, s1, FEAT4(Param10), ε, ε, FEAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param10, Param18); FEAT6(Param9)> to: <s5, s1, FEAT4(Param10), ε, ε, FEAT1(Param9, Param16, Param11, Param4, Param14, Param15, Param10, Param18, Paman13); FEAT6(Param9)>

All system test cases in the full test suite TS_{ibm} are listed in Table APP-3, where the last column indicates whether a system test case in the full test suite also appears in the RTS. Two test cases can traverse the same test sequence carrying different test data.

Table APP-3. Full test suite TS_{ibm} for IBM NTCv2

Test #	Test Sequences	Belong to RTS_{ibm}
1	t1, t2, t15, t16, t37, t31	√
2	t1, t2, t15, t16, t37, t31	√
3	t1, t2, t13, t31	√
4	t1, t2, t15, t16, t37, t31	√

5	t1, t2, t15, t16, t37, t31	√
6	t1, t2, t15, t16, t37, t31	√
7	t1, t2, t15, t16, t37, t31/ t1, t2, t13, t31	√
8	t1, t2, t15, t16, t37, t31/ t1, t2, t15, t16, t19, t24, t27, t28, t37, t31	√
9	t1, t2, t15, t16, t37, t31/ t1, t2, t86	√
10	t1, t2, t15, t16, t37, t31	√
11	t1, t2, t15, t16, t37, t31	√
12	t1, t2, t15, t16, t37, t31	√
13	t1, t2, t15, t16, t37, t31	√
14	t1, t2, t15, t16, t37, t31	
15	t1, t2, t86	
16	t1, t2, t15, t16, t37, t31	
17	t1, t2, t43, t37, t31	√
18	t1, t2, t43, t37, t31	
19	t1, t2, t43, t37, t31	√
20	t1, t2, t43, t37, t31	√
21	t1, t2, t86	
22	t1, t2, t11, t32, t56, t64, t19, t24, t27, t28, t37, t31	
23	t1, t2, t86	√
24	t1, t2, t11, t32, t56, t64, t37, t31	√
25	t1, t2, t11, t32, t56, t64, t37, t31/ t1, t2, t15, t44, t45, t37, t31	
26	t1, t2, t15, t44, t45, t37, t31	√
27	t1, t2, t15, t44, t45, t37, t31	
28	t1, t2, t15, t44, t45, t37, t31	
29	t1, t2, t15, t44, t45, t37, t31	
30	t1, t2, t15, t44, t45, t37, t31	
31	t1, t2, t49, t50, t52, t53, t37, t31	
32	t1, t2, t46, t47, t45, t54, t51, t31/ t1, t2, t46, t47, t45, t27, t28, t54, t51, t31/ t1, t2, t46, t47, t45, t37, t31	
33	t1, t2, t15, t44, t45, t37, t31	

34	t1, t2, t15, t44, t45, t19, t24, t27, t28, t31	√
35	t1, t2, t46, t47, t45, t37, t31/ t1, t2, t15, t44, t45, t37, t31	√
36	t1, t2, t49, t50, t51, t31	√
37	t1, t2, t49, t50, t51, t31	√
38	t1, t2, t49, t50, t52, t53, t37, t31	√
39	t1, t2, t46, t47, t45, t54, t51, t31/ t1, t2, t49, t50, t52, t53, t37, t31/ t1, t2, t15, t44, t45, t37, t31	√
40	t1, t2, t46, t47, t45, t54, t51, t31/ t1, t2, t49, t50, t51, t31/ t1, t2, t46, t47, t45, t27, t28, t54, t51, t31	√
41	t1, t2, t15, t44, t45, t37, t31	√
42	t1, t2, t15, t44, t19, t24, t27, t31	
43	t1, t2, t86	
44	t1, t2, t96, t92, t31	√
45	t1, t2, t96, t92, t31	√
46	t1, t2, t96, t92, t31	√
47	t1, t2, t96, t92, t31	√
48	t1, t2, t96, t87/ t1, t2, t86	√
49	t1, t2, t86/ t1, t2, t88, t94, t12, t31	√
50	t1, t2, t88, t94, t12, t31/ t1, t2, t86	√
51	t1, t2, t15, t44, t45, t37, t31	
52	t1, t2, t31	√
53	t1, t2, t43, t37, t31/ t1, t2, t15, t12, t31	
54	t1, t2, t15, t44, t45, t37, t31	
55	t1, t2, t15, t44, t45, t37, t31	
56	t1, t2, t15, t44, t45, t37, t31	
57	t1, t2, t15, t44, t45, t37, t31	√
58	t1, t2, t15, t44, t45, t37, t31	√
59	t1, t2, t86	√
60	t1, t2, t15, t44, t45, t19, t24, t37, t31	
61	t1, t2, t38, t39, t12, t31	√

62	t1, t2, t38, t39, t12, t31	
63	t1, t2, t38, t39, t12, t31	√
64	t1, t2, t38, t39, t12, t31	√
65	t1, t2, t37	√
66	t1, t2, t15, t12, t31	√
67	t1, t2, t15, t12, t31	√
68	t1, t2, t15, t84	√
69	t1, t2, t37	
70	t1, t2, t15, t84	
71	t1, t2, t37	√
72	t1, t2, t37	√
73	t1, t2, t15, t84	√
74	t1, t2, t37	√

Table APP-4. NDPM identified w.r.t. EMs in IBM NTCv2

EMs	NDPMs
m_1 (Add t_{87})	Affecting DD (t2, t87, Param9)
	Affecting DD (t2, t87, Param9)
	Affecting DD (t2, t87, Param16)
	Affecting DD (t2, t87, Param11)
	Affecting DD (t2, t87, Param4)
	Affecting DD (t11, t87, Param14)*
	Affecting DD (t11, t87, Param15)*
	Affecting DD (t13, t87, Param14)*
	Affecting DD (t13, t87, Param15)*
	Affecting DD (t15, t87, Param14)*
	Affecting DD (t15, t87, Param15)*
	Affecting DD (t38, t87, Param14)*
	Affecting DD (t38, t87, Param15)*
	Affecting DD (t43, t87, Param14)*
	Affecting DD (t43, t87, Param15)*
	Affecting DD (t46, t87, Param14)*
	Affecting DD (t46, t87, Param15)*
	Affecting DD (t49, t87, Param14)*
	Affecting DD (t49, t87, Param15)*
	Affecting DD (t88, t87, Param14)*
	Affecting DD (t88, t87, Param15)*
	Affecting DD (t96, t87, Param14)
	Affecting DD (t96, t87, Param15)
	Affecting CD (t93, t87)
	Affecting CD (t96, t87)

m_2 (Add t_{88})	Affecting DD (t2, t88, Param16) Affecting CD (t2, t88) Affected DD (t88, t31, Param14)* Affected DD (t88, t31, Param15)* Affected DD (t88, t41, Param14)* Affected DD (t88, t41, Param15)* Affected DD (t88, t59, Param14)* Affected DD (t88, t59, Param15)* Affected DD (t88, t60, Param14)* Affected DD (t88, t60, Param15)* Affected DD (t88, t61, Param14)* Affected DD (t88, t61, Param15)* Affected DD (t88, t67, Param14)* Affected DD (t88, t67, Param15)* Affected DD (t88, t84, Param14)* Affected DD (t88, t84, Param15)* Affected DD (t88, t87, Param14)* Affected DD (t88, t87, Param15)* Affected DD (t88, t91, Param14) Affected DD (t88, t91, Param15) Affected CD (t88, t89) Affected CD (t88, t90) Affected CD (t88, t91) Affected CD (t88, t94)

m_3 (Add t_{89})	Affecting CD (t88, t89)
	Affecting CD (t95, t89)
	Affected CD (t89, t8)
	Affected CD (t89, t9)
	Affected CD (t89, t11)
	Affected CD (t89, t13)
	Affected CD (t89, t15)
	Affected CD (t89, t38)
	Affected CD (t89, t43)
	Affected CD (t89, t46)
	Affected CD (t89, t49)
	Affected CD (t89, t86)
	Affected CD (t89, t88)
	Affected CD (t89, t96)
m_4 (Add t_{90})	Affecting CD (t88, t90)
m_5 (Add t_{91})	Affecting DD (t2, t91, Param9)
	Affecting DD (t2, t91, Param9)
	Affecting DD (t2, t91, Param16)
	Affecting DD (t2, t91, Param11)
	Affecting DD (t2, t91, Param4)
	Affecting DD (t11, t91, Param14)*
	Affecting DD (t11, t91, Param15)*
	Affecting DD (t13, t91, Param14)*
	Affecting DD (t13, t91, Param15)*
	Affecting DD (t15, t91, Param14)*
	Affecting DD (t15, t91, Param15)*
	Affecting DD (t38, t91, Param14)*
	Affecting DD (t38, t91, Param15)*
	Affecting DD (t43, t91, Param14)*
	Affecting DD (t43, t91, Param15)*
	Affecting DD (t46, t91, Param14)*

	Affecting DD (t46, t91, Param15)* Affecting DD (t49, t91, Param14)* Affecting DD (t49, t91, Param15)* Affecting DD (t88, t91, Param14) Affecting DD (t88, t91, Param15) Affecting DD (t96, t91, Param14)* Affecting DD (t96, t91, Param15)* Affecting CD (t88, t91)
m_6 (Add t_{92})	Affecting DD (t2, t92, Param16) Affecting CD (t93, t92) Affecting CD (t96, t92) Affected DD (t92, t31, Param1) Affected CD (t92, t14)* Affected CD (t92, t30) Affected CD (t92, t31) Affected CD (t92, t73)* Affected CD (t92, t81)* Affected CD (t92, t82)* Affected CD (t92, t93)
m_7 (Add t_{93})	Affecting DD (t2, t93, Param16) Affecting CD (t12, t93)* Affecting CD (t13, t93)* Affecting CD (t37, t93)* Affecting CD (t51, t93)* Affecting CD (t92, t93) Affected CD (t93, t87) Affected CD (t93, t92) Affected CD (t93, t97) Affected CD (t93, t98)

m_8 (Add t_{94})	Affecting CD (t88, t94) Affecting CD (t95, t94) Affected DD (t94, t84, Param13) Affected CD (t94, t12) Affected CD (t94, t16)* Affected CD (t94, t44)* Affected CD (t94, t78)* Affected CD (t94, t79)* Affected CD (t94, t84) Affected CD (t94, t85) Affected CD (t94, t95)
m_9 (Add t_{95})	Affecting DD (t2, t95, Param16) Affecting CD (t15, t95)* Affecting CD (t39, t95)* Affecting CD (t76, t95)* Affecting CD (t80, t95)* Affecting CD (t81, t95)* Affecting CD (t94, t95) Affected CD (t95, t89) Affected CD (t95, t90) Affected CD (t95, t91) Affected CD (t95, t94)

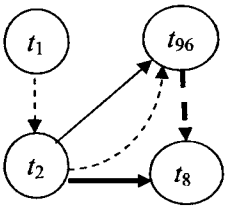
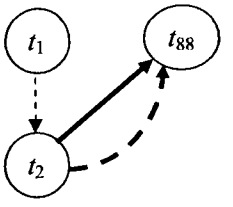
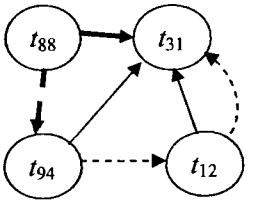
m_{10} (Add t_{96})	Affecting DD (t2, t96, Param16)
	Affecting CD (t2, t96)
	Affecting CD (t14, t96)*
	Affecting CD (t40, t96)*
	Affecting CD (t48, t96)*
	Affecting CD (t75, t96)*
	Affecting CD (t77, t96)*
	Affecting CD (t78, t96)*
	Affecting CD (t83, t96)*
	Affected DD (t96, t31, Param14)
	Affected DD (t96, t31, Param15)
	Affected DD (t96, t31, Param14)*
	Affected DD (t96, t31, Param15)*
	Affected DD (t96, t41, Param14)*
	Affected DD (t96, t41, Param15)*
	Affected DD (t96, t59, Param14)*
	Affected DD (t96, t59, Param15)*
	Affected DD (t96, t60, Param14)*
	Affected DD (t96, t60, Param15)*
	Affected DD (t96, t61, Param14)*
	Affected DD (t96, t61, Param15)*
	Affected DD (t96, t67, Param14)*
	Affected DD (t96, t67, Param15)*
	Affected DD (t96, t84, Param14)*
	Affected DD (t96, t84, Param15)*
	Affected DD (t96, t87, Param14)
	Affected DD (t96, t87, Param15)
	Affected DD (t96, t91, Param14)*
	Affected DD (t96, t91, Param15)*
	Affected CD (t96, t87)
	Affected CD (t96, t92)

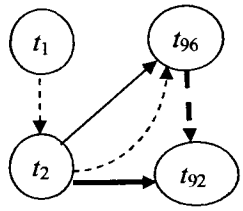
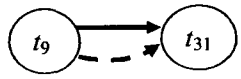
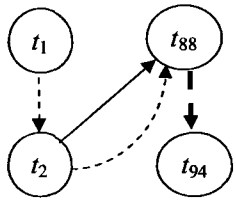
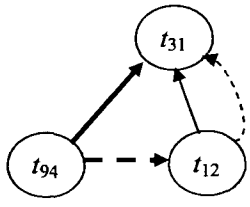
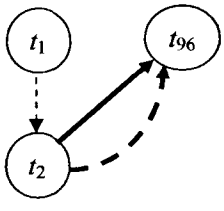
	Affected CD (t96, t97)
	Affected CD (t96, t98)
m_{11} (Add t_{97})	Affecting CD (t93, t97)
	Affecting CD (t96, t97)
	Affected CD (t97, t8)
	Affected CD (t97, t9)
	Affected CD (t97, t11)
	Affected CD (t97, t13)
	Affected CD (t97, t15)
	Affected CD (t97, t38)
	Affected CD (t97, t43)
	Affected CD (t97, t46)
	Affected CD (t97, t49)
	Affected CD (t97, t86)
	Affected CD (t97, t88)
	Affected CD (t97, t96)
m_{12} (Add t_{98})	Affecting CD (t93, t98)
	Affecting CD (t96, t98)
m_{13} (change t_{12})	Affecting DD (t2, t12, Param16)
	Affecting DD (t2, t12, Param16)
	Affected DD (t12, t31, Param10)
	Affected DD (t12, t67, Param10)*
m_{14} (change t_{15})	Affecting DD (t2, t15, Param16)
m_{15} (change t_{16})	Affected DD (t16, t31, Param10)
	Affected DD (t16, t67, Param10)
m_{16} (change t_{31})	Affecting DD (t92, t31, Param1)
	Affecting DD (t94, t31, Param13)
m_{17} (change t_{39})	NONE
m_{18} (change t_{44})	Affected DD (t44, t31, Param10)
	Affected DD (t44, t67, Param10)
m_{19} (change t_{78})	Affecting DD (t2, t78, Param16)

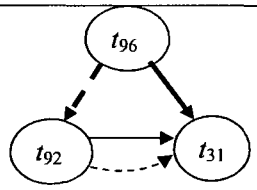
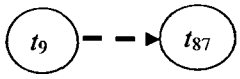
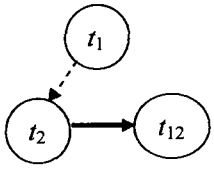
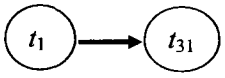
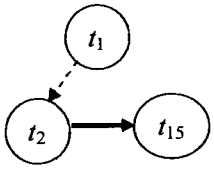
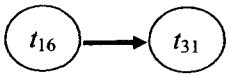
m_{20} (change t_{81})	Affecting DD (t2, t81, Param16) Affecting DD (t2, t81, Param16)
m_{21} (change t_{84})	Affecting DD (t94, t84, Param13)

* means the NDPM is not applicable.

Table APP-5, IPs and equivalent system test cases in RTS_{ibm} w.r.t. IPs

IP #	IP	Equivalent System Test Cases w.r.t. IP (specified by test ids)
m_1 (Add t_{87})		
1	Affecting IP 	48
m_2 (Add t_{88})		
2	Affecting IP 	49, 50
3	Affected IP 	49, 50

	m_6 (Add t_{92})	
4	Affecting IP 	44, 45, 46, 47
5	Affected IP 	44, 45, 46, 47
	m_8 (Add t_{94})	
6	Affecting IP 	49, 50
7	Affected IP 	49, 50
	m_{10}(Add t_{96})	
8	Affecting IP 	44, 45, 46, 47, 48
9	Affected IP	44, 45, 46, 47

		
10	Affected IP 	48
	m_{13}(Change t_{12})	
11	Affecting IP 	49, 50, 61, 63, 64, 66, 67
12	Affected IP 	49, 50, 61, 63, 64, 66, 67
	m_{14}(Change t_{15})	
13	Affecting IP 	1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 26, 34, 35, 39, 41, 57, 58, 66, 67, 68, 73
	m_{15}(Change t_{16})	
14	Affected IP 	1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
	m_{16}(Change t_{31})	
15	Affecting IP	44, 45, 46, 47

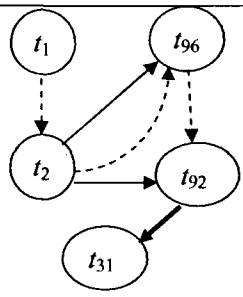
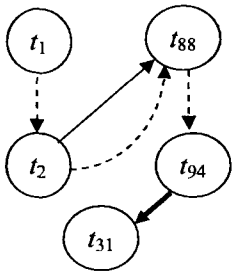
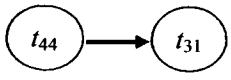
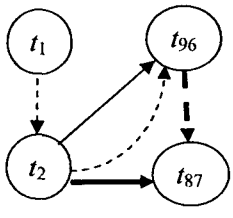
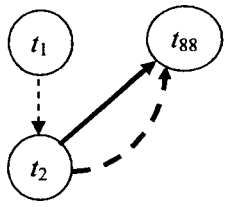
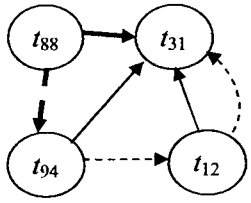
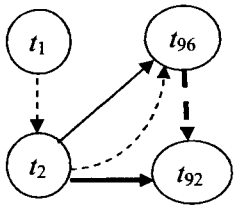
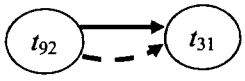
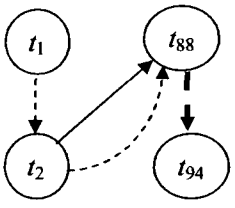
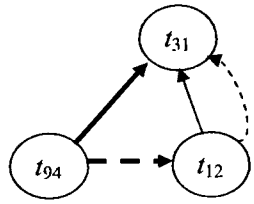
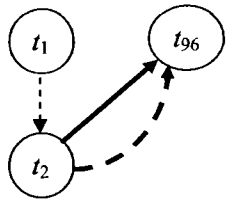
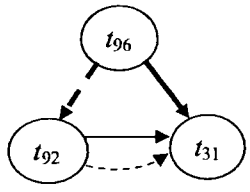
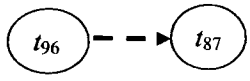
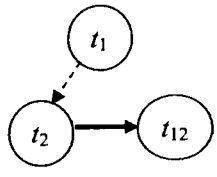
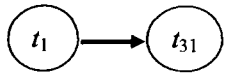
		
16	Affecting IP 	49, 50
	m_{18} (Change t_{44})	
17	Affected IP 	26, 34, 35, 39, 41, 57, 58

Table APP-6. IPs and equivalent system test cases in TS_{ibm} w.r.t. IPs

IP #	**IP**	**Equivalent System Test Cases w.r.t. IP (specified by test ids)**
	m_1	
1	Affecting IP 	48
	m_2	

2	Affecting IP 	49, 50
3	Affected IP 	49, 50
	m_6	
4	Affecting IP 	44, 45, 46, 47
5	Affected IP 	44, 45, 46, 47
	m_8	
6	Affecting IP 	49, 50

7	Affected IP 	49, 50
	m_{10}	
8	Affecting IP 	44, 45, 46, 47, 48
9	Affected IP 	44, 45, 46, 47
10	Affected IP 	48
	m_{13}	
11	Affecting IP 	49, 50, 53, 61, 62, 63, 64, 66, 67
12	Affected IP 	49, 50, 53, 61, 62, 63, 64, 66, 67

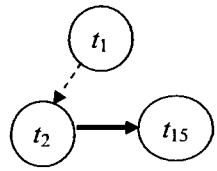
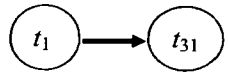
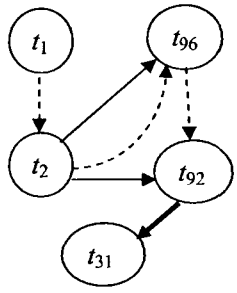
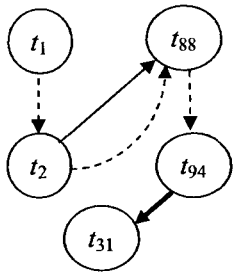
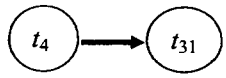
m_{14}		
13	Affecting IP 	1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 25, 26, 27, 28, 29, 30, 33, 34, 35, 39, 41, 42, 51, 53, 54, 55, 56, 57, 58, 60, 66, 67, 68, 70, 73
m_{15}		
14	Affected IP 	1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16
m_{16}		
15	Affecting IP 	44, 45, 46, 47
16	Affecting IP 	49, 50
m_{18}		
17	Affected IP 	25, 26, 27, 28, 29, 30, 33, 34, 35, 39, 41, 42, 51, 54, 55, 56, 57, 58, 60

Table APP-7. Generate RTS RTS_g for IBM NTCv2

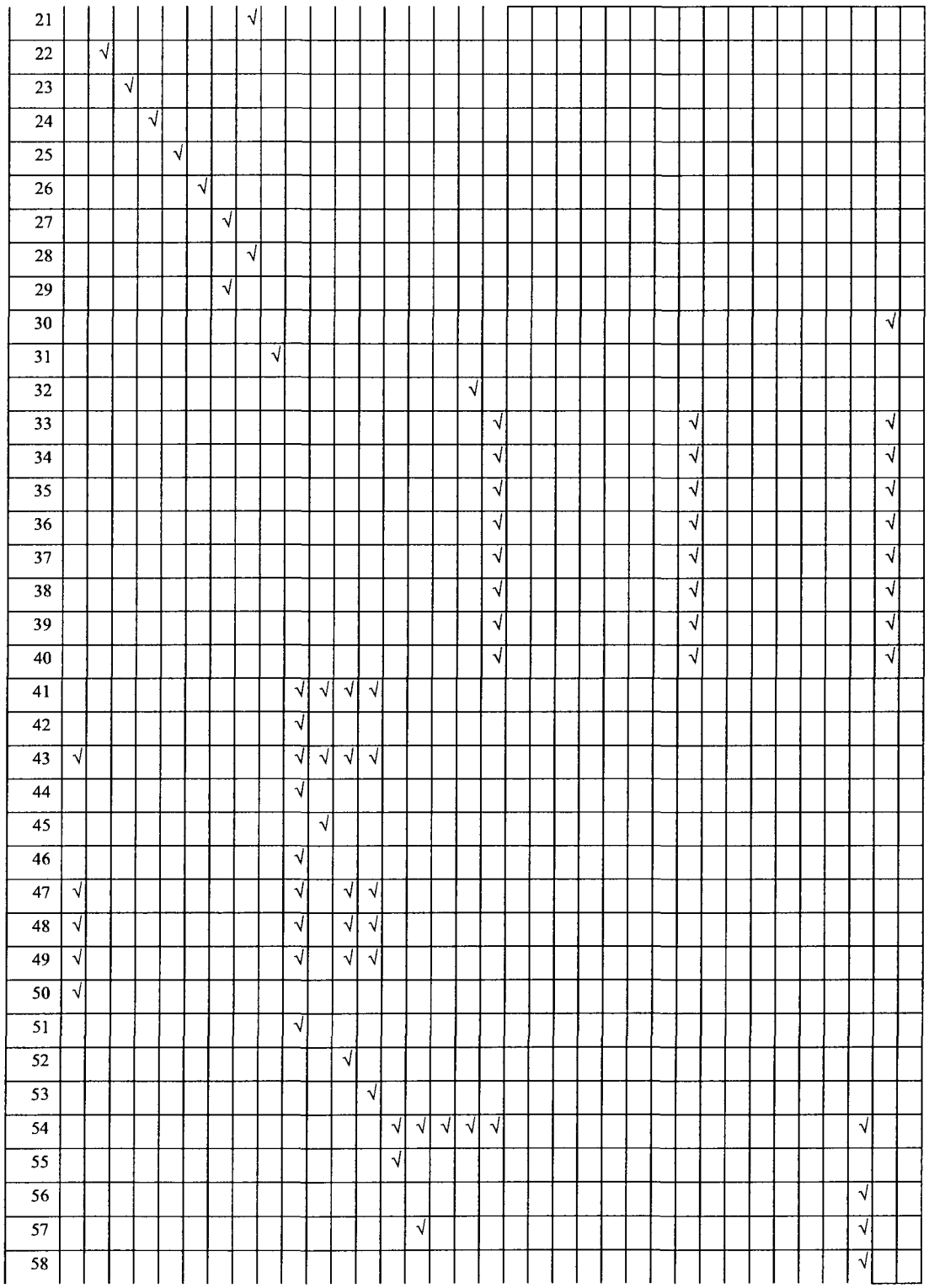
Test #	Test Sequences
1	t1, t2, t96, t92, t93, t87
2	t1, t2, t88, t89, t8, t2, t11, t59
3	t1, t2, t88, t89, t8, t2, t13, t31
4	t1, t2, t88, t89, t8, t2, t15, t84
5	t1, t2, t88, t89, t8, t2, t38, t41
6	t1, t2, t88, t89, t8, t2, t43, t67
7	t1, t2, t88, t89, t8, t2, t46, t48, t86
8	t1, t2, t88, t89, t8, t2, t49, t83, t9
9	t1, t2, t88, t89, t8, t2, t96, t87
10	t1, t2, t96, t92, t93, t92, t31
11	t1, t2, t96, t92, t30
12	t1, t2, t96, t92, t93, t97, t9
13	t1, t2, t96, t92, t93, t98
14	t1, t2, t88, t94, t95, t94, t85
15	t1, t2, t88, t94, t12, t31
16	t1, t2, t88, t94, t95, t89, t86
17	t1, t2, t88, t94, t95, t90
18	t1, t2, t88, t94, t95, t91
19	t1, t2, t96, t97, t8, t2, t11, t59
20	t1, t2, t96, t97, t8, t2, t13, t31
21	t1, t2, t96, t97, t8, t2, t15, t84
22	t1, t2, t96, t97, t8, t2, t38, t41
23	t1, t2, t96, t97, t8, t2, t43, t67
24	t1, t2, t96, t97, t8, t2, t46, t48, t86
25	t1, t2, t96, t97, t8, t2, t49, t83, t9
26	t1, t2, t96, t97, t8, t2, t88, t91
27	t1, t2, t96, t97, t96, t87
28	t1, t2, t15, t16, t37, t31

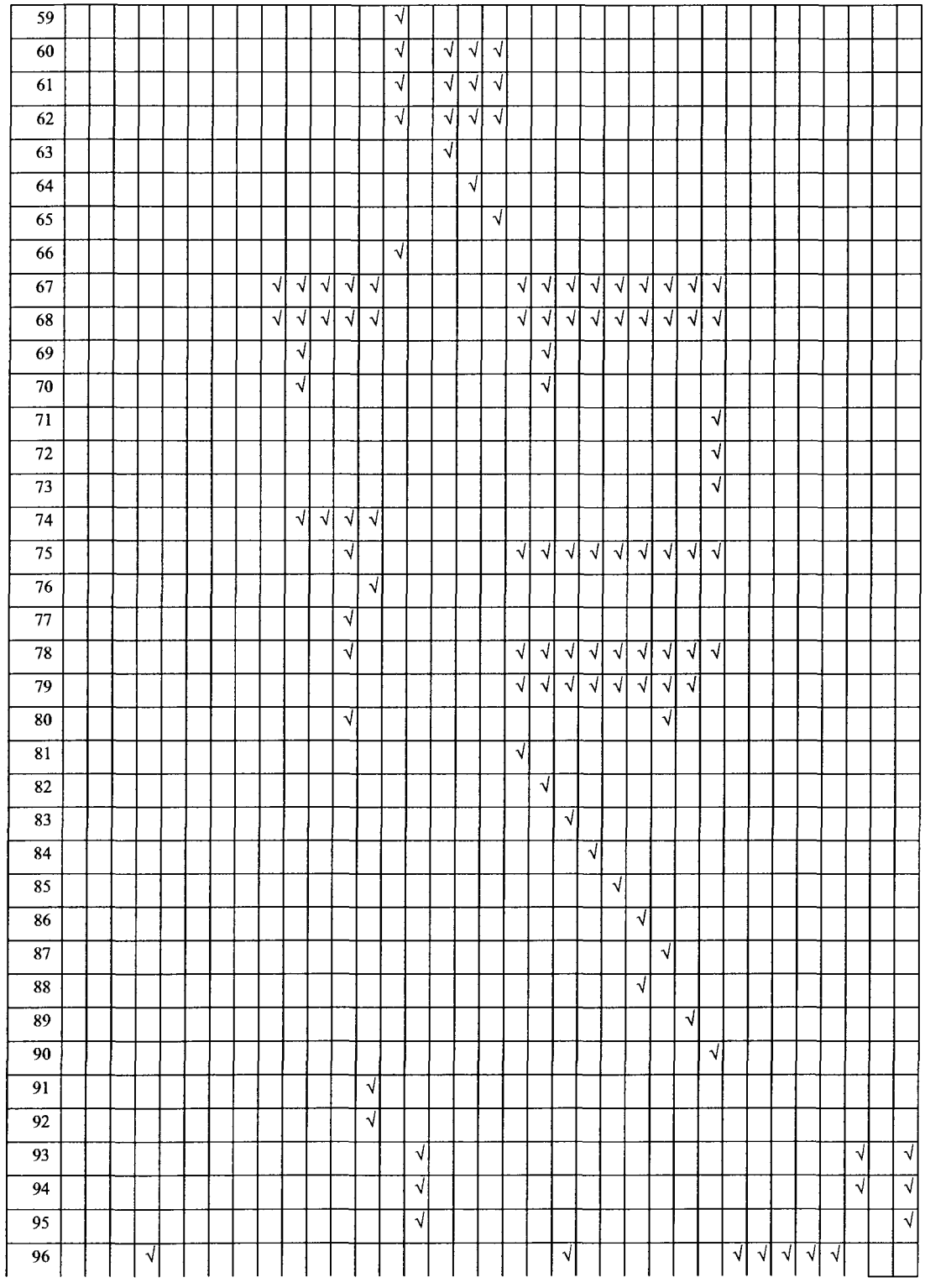
29	t1, t2, t15, t16, t67
30	t1, t2, t15, t44, t45, t37, t31
31	t1, t2, t15, t44, t45, t67
32	t1, t2, t15, t78, t86
33	t1, t2, t88, t94, t12, t81, t84
34	t1, t2, t88, t89, t88, t91
35	t1, t2, t38, t39, t12, t31

Table APP-8 lists relationships between generated regression test cases in RTS_g and NDPMs / EMs that are covered by each of the test cases. Because of the size of the table, we have to use small fonts to make it fit.

Table APP-8. Generated regression test cases in RTS_g vs. covered NDPMs and EMs

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	√																									√									
2	√																									√									
3	√																									√									
4	√																									√									
5	√																									√									
6	√																									√									
7	√																									√									
8	√																																		
9	√																									√									
10		√	√	√	√	√	√	√	√				√	√	√	√	√								√						√	√			
11		√	√	√	√	√	√	√	√				√	√	√	√	√								√						√	√			
12																√									√							√			
13																√									√							√			
14		√	√	√	√	√	√	√	√					√																		√			
15															√																					
16									√							√									√							√			
17													√	√	√	√	√																√			
18		√	√	√	√	√	√	√	√						√																				
19															√																					
20		√	√	√	√	√	√	√	√																										





97																										√						
98																											√					
99										√																					
100														√																	
101																												√			
102																													√		
103																														√	
104																															√
105																															√
106																															√
m1	√																							√							
m2		√	√	√	√	√	√	√	√				√	√	√	√	√						√						√	√	
m3		√	√	√	√	√	√	√	√						√															√	
m4																√															
m5																	√						√							√	
m6	√									√	√	√	√																		
m7	√									√		√	√																		
m8														√	√	√	√	√												√	
m9														√		√	√	√													
m10									√	√	√	√	√						√	√	√	√	√	√	√	√	√					
m11												√							√	√	√	√	√	√	√	√	√					
m12													√																		
m13														√																√		√
m14				√																√					√	√	√	√	√	√			
m15																										√	√					
m16										√				√																		
m17																																√
m18																												√	√			
m19																														√		
m20																															√	
m21																															√	

Appendix G: Simplified ATM Model

Table APP-9. Xie's RTS for the simplified ATM system

Test #	Test Sequences
1	T1 T4 T5 T7 T8
2	T1 T4 T5 T7 T9 T7 T8
3	T1 T4 T5 T7 T6dummy T9 T7 T8
4	T1 T4 T6dummy T5 T7 T8
5	T1 T4 T6dummy T5 T7 T9 T7 T8
6	T1 T4 T6dummy T5 T7 T6dummy T9 T7 T8
7	T1 T4 T6dummy T9 T7 T5 T7 T8
8	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T8
9	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
10	T1 T4 T9 T7 T5 T7 T8
11	T1 T4 T9 T7 T5 T7 T9 T7 T8
12	T1 T4 T9 T7 T5 T7 T6dummy T9 T7 T8
13	T1 T4 T9 T7 T6dummy T5 T7 T8
14	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T
15	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
16	T1 T4 T5 T7 T5 T7 T8
17	T1 T4 T5 T7 T5 T7 T9 T7 T8
18	T1 T4 T5 T7 T5 T7 T6dummy T9 T7 T8
19	T1 T4 T5 T7 T6dummy T5 T7 T8
20	T1 T4 T5 T7 T6dummy T5 T7 T9 T7 T8
21	T1 T4 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
22	T1 T4 T5 T7 T9 T7 T5 T7 T8
23	T1 T4 T5 T7 T9 T7 T5 T7 T9 T7 T8
24	T1 T4 T5 T7 T9 T7 T5 T7 T6dummy T9 T7 T8
25	T1 T4 T5 T7 T6dummy T9 T7 T5 T7 T8
26	T1 T4 T5 T7 T6dummy T9 T7 T5 T7 T9 T7 T8

27	T1 T4 T5 T7 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
28	T1 T4 T5 T7 T9 T7 T6dummy T5 T7 T8
29	T1 T4 T5 T7 T9 T7 T6dummy T5 T7 T9 T7 T8
30	T1 T4 T5 T7 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
31	T1 T4 T6dummy T5 T7 T5 T7 T8
32	T1 T4 T6dummy T5 T7 T5 T7 T9 T7 T8
33	T1 T4 T6dummy T5 T7 T5 T7 T6dummy T9 T7 T8
34	T1 T4 T6dummy T5 T7 T6dummy T5 T7 T8
35	T1 T4 T6dummy T5 T7 T6dummy T5 T7 T9 T7 T8
36	T1 T4 T6dummy T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
37	T1 T4 T6dummy T5 T7 T9 T7 T5 T7 T8
38	T1 T4 T6dummy T5 T7 T9 T7 T5 T7 T9 T7 T8
39	T1 T4 T6dummy T5 T7 T9 T7 T5 T7 T6dummy T9 T7 T8
40	T1 T4 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T8
41	T1 T4 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T9 T7 T8
42	T1 T4 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
43	T1 T4 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T8
44	T1 T4 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T9 T7 T8
45	T1 T4 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
46	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T8
47	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T9 T7 T8
48	T1 T4 T6dummy T9 T7 T5 T7 T5 T7 T6dummy T9 T7 T8
49	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T5 T7 T8
50	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T5 T7 T9 T7 T8
51	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
52	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T5 T7 T8
53	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T5 T7 T9 T7 T8
54	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T5 T7 T6dummy T9 T7 T8
55	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T8
56	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T9 T7 T8

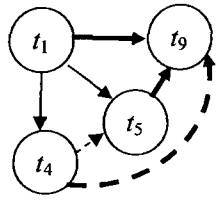
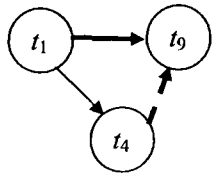
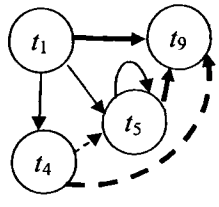
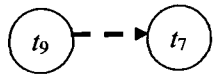
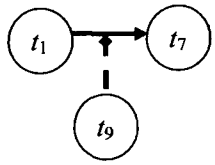
57	T1 T4 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
58	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T8
59	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T9 T7 T8
60	T1 T4 T6dummy T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
61	T1 T4 T9 T7 T5 T7 T5 T7 T8
62	T1 T4 T9 T7 T5 T7 T5 T7 T9 T7 T8
63	T1 T4 T9 T7 T5 T7 T5 T7 T6dummy T9 T7 T8
64	T1 T4 T9 T7 T5 T7 T6dummy T5 T7 T8
65	T1 T4 T9 T7 T5 T7 T6dummy T5 T7 T9 T7 T8
66	T1 T4 T9 T7 T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
67	T1 T4 T9 T7 T5 T7 T9 T7 T5 T7 T8
68	T1 T4 T9 T7 T5 T7 T9 T7 T5 T7 T9 T7 T8
69	T1 T4 T9 T7 T5 T7 T9 T7 T5 T7 T6dummy T9 T7 T8
70	T1 T4 T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T8
71	T1 T4 T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T9 T7 T8
72	T1 T4 T9 T7 T5 T7 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
73	T1 T4 T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T8
74	T1 T4 T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T9 T7 T8
75	T1 T4 T9 T7 T5 T7 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
76	T1 T4 T9 T7 T6dummy T5 T7 T5 T7 T8
77	T1 T4 T9 T7 T6dummy T5 T7 T5 T7 T9 T7 T8
78	T1 T4 T9 T7 T6dummy T5 T7 T5 T7 T6dummy T9 T7 T8
79	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T5 T7 T8
80	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T5 T7 T9 T7 T8
81	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T5 T7 T6dummy T9 T7 T8
82	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T5 T7 T8
83	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T5 T7 T9 T7 T8
84	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T5 T7 T6dummy T9 T7 T8
85	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T8
86	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T9 T7 T8

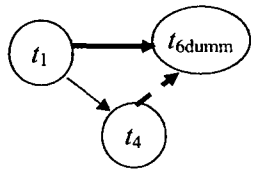
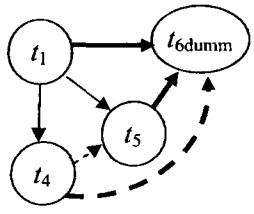
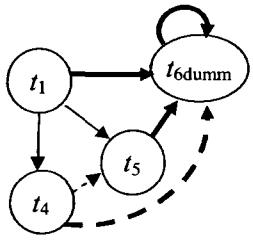
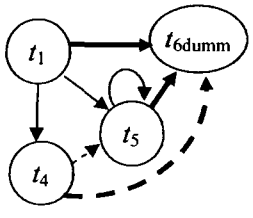
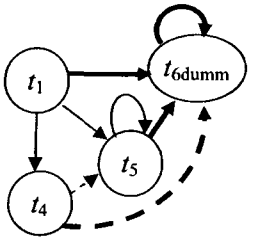
87	T1 T4 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T5 T7 T6dummy T9 T7 T8
88	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T8
89	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T9 T7 T8
90	T1 T4 T9 T7 T6dummy T5 T7 T9 T7 T6dummy T5 T7 T6dummy T9 T7 T8
91	T1 T4 T9 T7 T8
92	T1 T4 T6dummy T9 T7 T8
93	T1 T2 T2 T2 T3

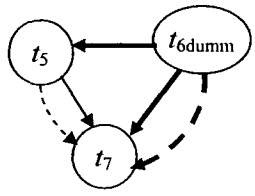
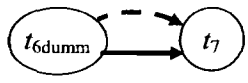
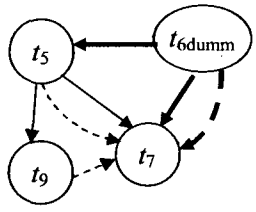
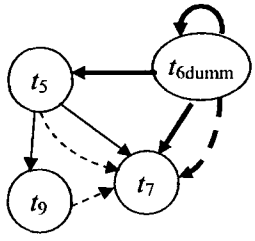
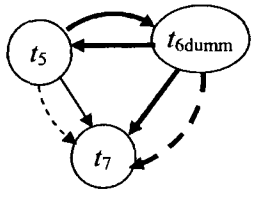
Table APP-10. NDPMs identified in Study 1 on the simplified ATM system

EMs	NDPMs
m_1: ADD $t_{9:}$ <S_2, S_3, Balance, ε, Display(b), ε>	Affecting CD (t4, t9) Affecting DD (t1, t9, b) Affecting DD (t5, t9, b) Affected CD (t9, t7) Activation DD (t1, t7, b)
m_2: DELETE t_6	Affecting GDD (t1, t6, b) Affecting GDD (t5, t6, b) Affecting GDD (t6, t6, b) Affecting GCD (t4, t6) Affected GDD (t6, t6, b) Affected GDD (t6, t5, b) Affected GDD (t6, t7, b) Affected GCD (t6, t7)

Table APP-11. IPs and equivalent test cases w.r.t. IPs in Study 1

IP #	IP	Equivalent Test Cases w.r.t. IP (specified by test#)
	m_1 (Add t_9)	
1	Affecting IP 	2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 22, 25, 28, 37, 40, 43, 52, 55, 58, 67, 70, 73, 82, 85, 88
2	Affecting IP 	7, 10, 13, 46, 49, 61, 64, 76, 79, 91, 92
3	Affecting IP 	17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32, 33, 35, 36, 38, 39, 41, 42, 44, 45, 47, 48, 50, 51, 53, 54, 56, 57, 59, 60, 62, 63, 65, 66, 68 69, 71, 72, 74, 75, 77, 78, 80, 81, 83, 84, 86, 87 89, 90
4	Affected IP 	2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
5	Side-effect IP 	2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92

m_2 (Delete t_6)		
6	Affecting IP 	4, 5, 7, 8, 13, 14, 31, 32, 37, 38, 46, 47, 52, 53, 76, 77, 82, 83, 92
7	Affecting IP 	3, 12, 19, 20, 25, 26, 28, 29, 64, 65, 70, 71, 73, 74
8	Affecting IP 	6, 9, 15, 34, 35, 40, 41, 43, 44, 49, 50, 55, 56, 58, 59, 79, 80, 85, 86, 88, 89
9	Affecting IP 	18, 24, 63, 69
10	Affecting IP 	21, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 66, 72, 75, 78, 81, 84, 87, 90

11	Affected IP 	4, 7, 13, 19, 25, 28, 64, 70, 73
12	Affected IP 	3, 12, 18, 24, 63, 69, 92
13	Affected IP 	5, 8, 14, 20, 26, 29, 65, 71, 74
14	Affected IP 	6, 9, 15, 21, 27, 30, 66, 72, 75
15	Affected IP 	31, 46, 76

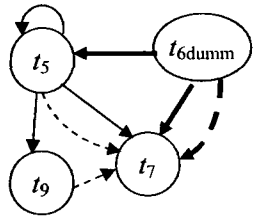
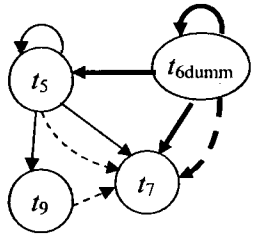
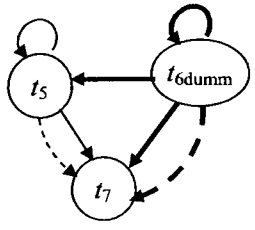
16	Affected IP 	32, 37, 38, 47, 52, 53, 77, 82, 83
17	Affected IP 	33, 35, 36, 39, 40, 41, 42, 43, 44, 45, 48, 50, 51, 54, 55, 56, 57, 58, 59, 60, 78, 80, 81, 84, 85, 86, 87, 88, 89, 90
18	Affected IP 	34, 49, 79

Table APP-12. Test cases in our reduced RTS vs. covered IPs in Study 1

Test # / IP #		5	18	21	25	32	33	34	46
m_1	1	√			√				
	2								√
	3		√	√		√	√		
	4	√	√	√	√	√	√		√
	5	√	√	√	√	√	√		√
m_2	6	√				√			√
	7				√				
	8							√	

	9		√						
	10			√			√		
	11				√				
	12		√						
	13	√							
	14			√					
	15								√
	16					√			
	17						√		
	18							√	

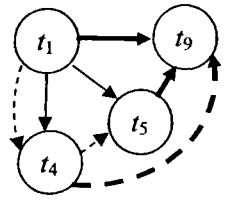
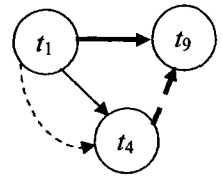
Table APP-13. NDPMs identified in Study 2

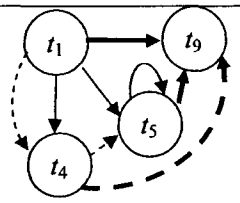
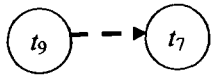
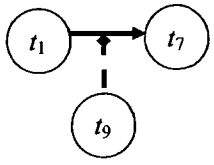
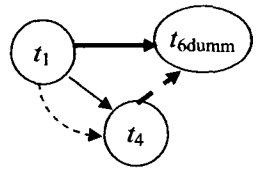
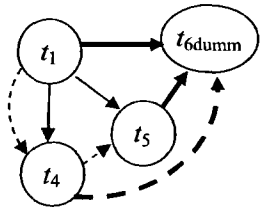
EMs	NDPMs
m_1: ADD t_9: <S_2, S_3, Balance, ε, Display(b), ε>	Affecting CD (t4, t9) Affecting DD (t1, t9, b) Affecting DD (t5, t9, b) Affected CD (t9, t7) Activation DD (t1, t7, b)
m_2: DELETE t_6	Affecting GDD (t1, t6, b) Affecting GDD (t5, t6, b) Affecting GDD (t6, t6, b) Affecting GCD (t4, t6) Affected GDD (t6, t6, b) Affected GDD (t6, t5, b) Affected GDD (t6, t7, b) Affected GCD (t6, t7)
m_3: CHANGE t_5 to: <S_2, S_3, Withdrawal(w), w < b, , b = b - w>	Affecting DD (t1, t5, b) Affecting DD (t5, t5, b)

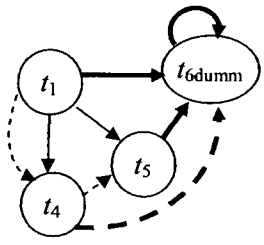
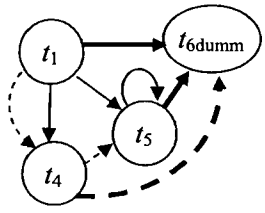
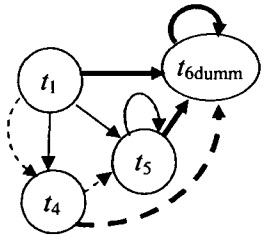
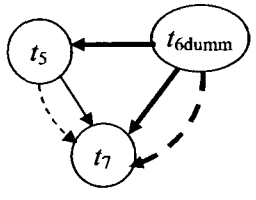
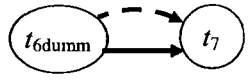
m_4: ADD t10: <Start, Exit, CardError, Display error, Eject card>.	Activation CD (t1, t2) Activation CD (t1, t4) Activation CD (t1, t5) Activation CD (t1, t8)
m_5: DELETE t3	Affecting GDD (t1, t3, attempts) * Affecting GDD (t1, t3, pin) Affecting GDD (t2, t3, attempts) Activation CD (t1, t5) Activation CD (t1, t8) Activation GCD (t4, t5) Activation GCD (t4, t8)

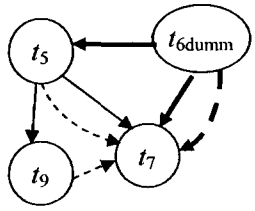
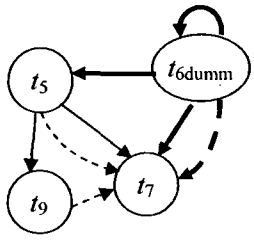
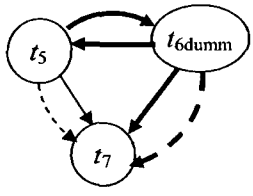
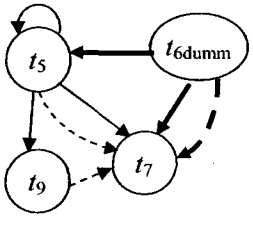
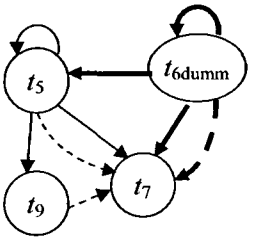
* Affecting GDD (t1, t3, attempts) is not applicable.

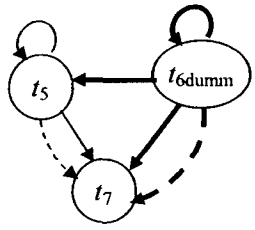
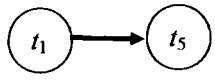
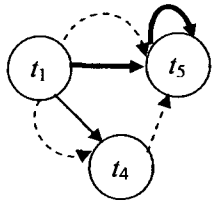
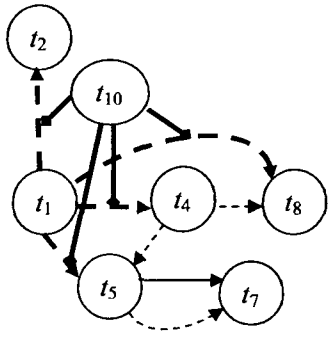
Table APP-14. IPs and the equivalent test cases w.r.t. IPs in Study 2

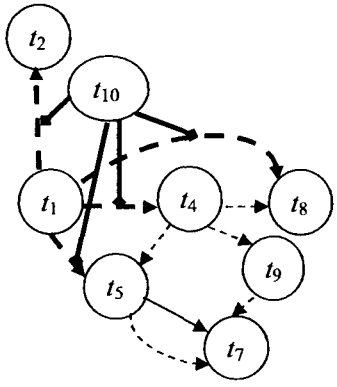
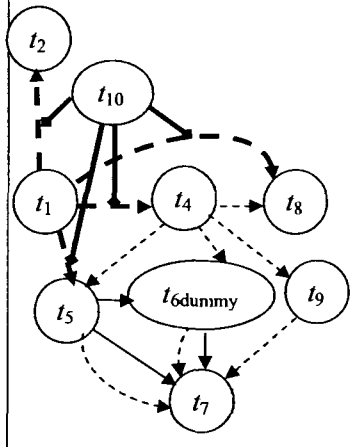
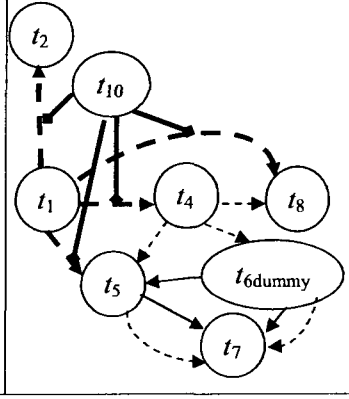
IP #	IP	Equivalent Test Cases w.r.t. IP (specified by test #)
	m_1 (Add t_9)	
1	Affecting IP 	2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 22, 25, 28, 37, 40, 43, 53, 55, 58, 67, 70, 73, 82, 85, 88
2	Affecting IP 	7, 10, 13, 46, 49, 61, 64, 76, 79, 91, 92
3	Affecting IP	17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32, 33, 35, 36, 38, 39, 41, 42, 44, 45, 47, 48, 50, 51, 53, 54, 56, 57, 59, 60, 62, 63, 65, 66, 68, 69, 71, 72, 74, 75, 77, 78, 80, 81, 83, 84, 86, 87, 89, 90

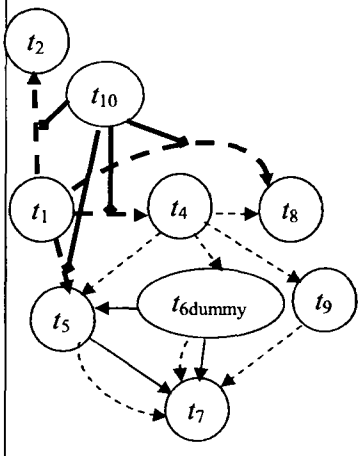
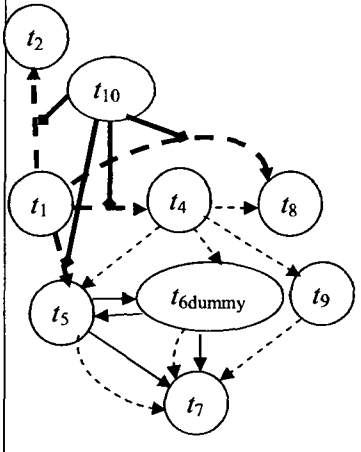
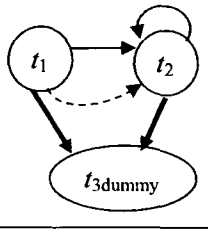
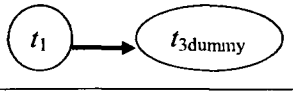
		
4	Affected IP 	2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
5	Side-effect IP 	2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92
	m_2 (Delete t_6)	
6	Affecting IP 	4, 5, 7, 8, 13, 14, 31, 32, 37, 38, 46, 47, 52, 53, 76, 77, 82, 83, 92
7	Affecting IP 	3, 12, 19, 20, 25, 26, 28, 29, 64, 65, 70, 71, 73, 74

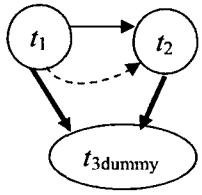
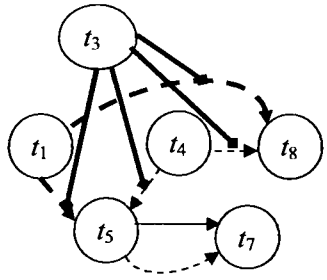
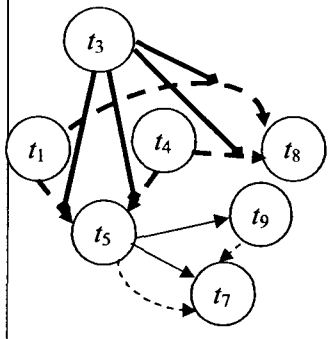
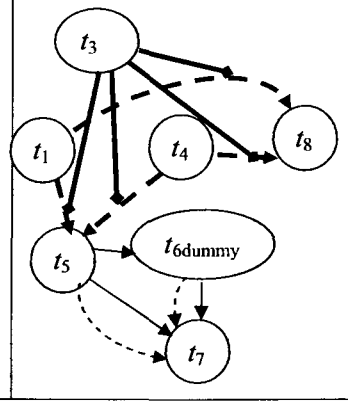
8	Affecting IP 	6, 9, 15, 34, 35, 40, 41, 43, 44, 49, 50, 55, 56, 58, 59, 79, 80, 85, 86, 88, 89
9	Affecting IP 	18, 24, 63, 69
10	Affecting IP 	21, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 66, 72, 75, 78, 81, 84, 87, 90
11	Affected IP 	4, 7, 13, 19, 25, 28, 64, 70, 73
12	Affected IP 	3, 12, 18, 24, 63, 69, 92

13	Affected IP 	5, 8, 14, 20, 26, 29, 65, 71, 84
14	Affected IP 	6, 9, 15, 21, 27, 30, 66, 72, 75
15	Affected IP 	31, 46, 76
16	Affected IP 	32, 37, 38, 47, 52, 53, 77, 82, 83
17	Affected IP 	33, 35, 36, 39, 40, 41, 42, 43, 44, 45, 48, 50, 51, 54, 55, 56, 57, 58, 59, 60, 78, 80, 81, 84, 85, 86, 87, 88, 89, 90

18	Affected IP 	34, 49, 79
	m_3 (Change t_5)	
19	Affecting IP 	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
20	Affecting IP 	16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90
	m_4 (Add t_{10})	
21	Side-effect IP 	1

22	Side-effect IP 	2, 10, 11
23	Side-effect IP 	3, 12
24	Side-effect IP 	4, 16

25	Side-effect IP 	5, 7, 8
26	Side-effect IP 	6, 9, 15, 19, 20, 21, 25, 26, 27, 28, 29, 30, 35, 36, 40, 41, 42, 43, 44, 45, 49, 50, 51, 55, 56, 57, 58, 59, 60, 63, 64, 65, 66, 70, 71, 72, 73, 75, 79, 80, 81, 85, 86, 87, 88, 89, 90
	m_5 (Delete t_3)	
27	Affecting IP 	93, 96
28	Affecting IP 	94

29	Affecting IP 	95
30	Side-effect IP 	1, 4, 5, 7, 8, 13, 14
31	Side-effect IP 	2, 10, 11
32	Side-effect IP 	3, 12

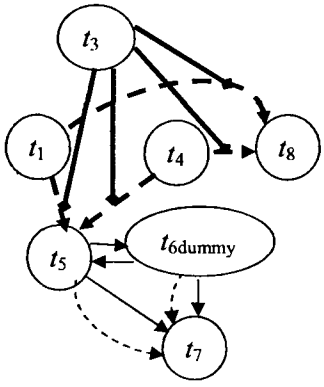
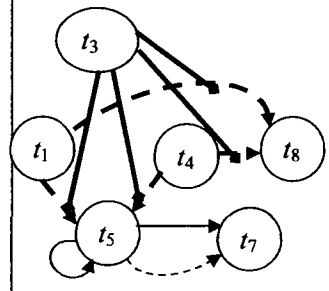
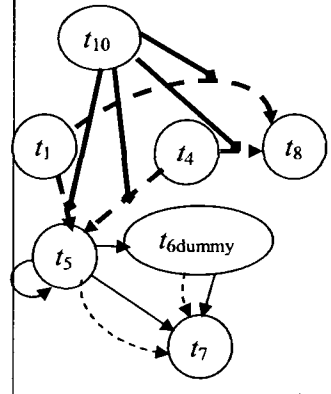
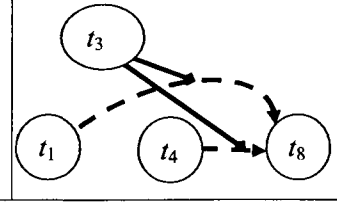
33	Side-effect IP 	6, 9, 15, 19, 20, 21, 25, 26, 27, 28, 29, 30, 34, 35, 36, 40, 41, 42, 43, 44, 45, 49, 50, 51, 55, 56, 57, 58, 59, 60, 64, 65, 66, 70, 71, 72, 73, 74, 75, 79, 80, 81, 85, 86, 87, 88, 89, 90
34	Side-effect IP 	16, 17, 22, 23, 31, 32, 37, 38, 46, 47, 52, 53, 61, 62, 67, 68, 76, 77, 82, 83
35	Side-effect IP 	18, 24, 33, 39, 48, 54, 63, 69, 78, 84
36	Side-effect IP 	91, 92

Table APP-15. RTS reduction results vs. EMs in Study 2

EM Under Test	Number of regression test cases	Number of reduced regression test cases (selected test #)	List of eliminated test cases (specified by test #)
Add t_9	86	9 (3, 5, 10, 18, 21, 32, 33, 49, 92,)	2, 6, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91
Delete t_6	79	10 (3, 4, 5, 18, 21, 31, 32, 33, 49, 92)	6, 7, 8, 9, 12, 13, 14, 15, 19, 20, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 64, 65, 66, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90
Change t_5	90	10 (1, 3, 4, 5, 18, 21, 31, 32, 33, 49)	2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
Add t_{10}	93	12 (1, 3, 4, 5, 10, 18, 21, 31, 32, 33, 49, 92)	2, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52,

			53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 97
Delete t_3	96	15 (1, 3, 4, 5, 10, 18, 21, 31, 32, 33, 49, 92, 93, 94, 95)	2, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 96

Table APP-16. Test cases in reduced RTS vs. covered IPs in Study 2

Test# / IP #		1	3	4	5	10	18	21	31	32	33	49	92	93	94	95
m_1	1		√		√											
	2					√						√	√			
	3						√	√		√	√					
	4		√		√	√	√	√		√	√	√	√			
	5		√		√	√	√	√		√	√	√	√			
m_2	6			√	√				√	√			√			
	7		√													
	8											√				
	9						√									
	10							√			√					
	11			√												
	12		√				√						√			
	13				√											

	14							1								
	15								√							
	16									√						
	17										√					
	18											√				
m_3	19	√	√	√	√	√										
	20						√	√	√	√	√	√				
m_4	21	√														
	22					√										
	23		√													
	24			√												
	25				√											
	26							√				√				
m_5	27													√		
	28														√	
	29															√
	30	√		√	√											
	31					√										
	32		√													
	33							√				√				
	34								√	√						
	35						√				√					
	36												√			

Since one of the twenty-six identified NDPM in Table APP-13 is not applicable, we only listed the twenty-five applicable NDPM in Table APP-17.

Table APP-17. Generated regression test cases vs. covered NDPMs in Study 3

NDPM \ Test #		1	2	3	4	5	6	7	8
1	Affecting CD (t_4, t_9)	√	√						
2	Affecting DD (t_1, t_9, b)	√							
3	Affecting DD (t_5, t_9, b)		√						
4	Affected CD (t_9, t_7)	√	√						
5	Activation DD (t_1, t_7, b)	√							
6	Affecting GDD (t_1, t_6, b)				√	√			
7	Affecting GDD (t_5, t_6, b)			√					
8	Affecting GDD (t_6, t_6, b)				√				
9	Affecting GCD (t_4, t_6)			√	√	√			
10	Affected GDD (t_6, t_6, b)				√				
11	Affected GDD (t_6, t_5, b)					√			
12	Affected GDD (t_6, t_7, b)			√	√	√			
13	Affected GCD (t_6, t_7)			√	√	√			
14	Affecting DD (t_1, t_5, b)			√			√		
15	Affecting DD (t_5, t_5, b)						√		
16	Activation CD (t_1, t_2)								√
17	Activation CD (t_1, t_4)	√	√	√	√	√	√		√
18	Activation CD (t_1, t_5)		√	√		√	√		√
19	Activation CD (t_1, t_8)	√	√	√	√	√	√		√
20	Affecting GDD (t_1, t_3, *pin*)							√	
21	Affecting GDD (t_2, t_3, *attempts*)							√	
22	Activation CD (t_1, t_5)		√	√		√	√	√	
23	Activation CD (t_1, t_8)	√	√	√	√	√	√	√	
24	Activation GCD (t_4, t_5)		√	√		√	√	√	
25	Activation GCD (t_4, t_8)	√	√	√	√	√	√	√	

Appendix H: The Five EFSM Models Provided By Tahat

1. ATM_Tahat model

The original EFSM model of the ATM system provided by Tahat:

T1: <0, 1, Card(pin, sb, cb, ca), , Write("Promote for PIN"), attempts := 0>

T2: <1, 1, PIN(p), (p != pin) and (attempts < 3), Write("Wrong PIN, re-enter PIN"), attempts := attempts + 1 >

T4: <1, 7, PIN(p), (p != pin) and (attempts = 3), Write("Wrong PIN, eject card"), Eject Card>

T5: <1, 2, PIN(p), p = pin, Write("Checking/Saving"), >

T6: <2, 3, Checking(), , , >

T7: <3, 2, Done(), , , >

T8: <2, 5, Savings(), , , >

T9: <5, 2, Done(), , , >

T10: <3, 4, Balance(), , Write(cb), >

T11: <3, 4, Deposit(d), , , cb := cb + d>

T12: <3, 4, Withdrawal(w), w > cb, Write("Insufficient Fund"), >

T13: <3, 4, Withdrawal(w), w < cb, , cb =cb – w>

T14: <4, 3, Receipt(), , Write(cb);Write("Menu Selection")>

T15: <5, 6, Withdrawal(w), w < sb, , sb := sb – w >

T16: <5, 6, Withdrawal(w), w > sb, Write("Insufficient Fund"), >

T17: <5, 6, Deposit(d), , , sb := sb + d>

T18: <5, 6, Balance(), , Write(sb), >

T19: <6, 5, Receipt(), , Write(sb); Write("Menu Selection"), >

T28: <2, 7, Exit(), , Write("Eject Card"), Eject Card>

Table APP-18. M_O and identified NDPMs for ATM_Tahat model

EMs	NDPMs
ADD T20: <3, 4, Withdrawal(w), (ca >= w) and (w <= cb) and (w >= 1) and (w <= 400), , cb := cb -w; ca := ca -2>	Affecting DD (T1, T20, ca) Affecting DD (T1, T20, ca) Affecting DD (T1, T20, cb) Affecting DD (T1, T20, cb) Affecting DD (T11, T20, cb) Affecting DD (T12, T20, cb) Affecting DD (T20, T20, ca) Affecting DD (T20, T20, ca) Affecting DD (T20, T20, cb) Affecting DD (T20, T20, cb) Affecting CD (T6, T20) Affected DD (T20, T10, cb) Affected DD (T20, T11, cb) Affected DD (T20, T12, cb) Affected DD (T20, T20, ca) Affected DD (T20, T20, ca) Affected DD (T20, T20, cb) Affected DD (T20, T20, cb) Affected DD (T20, T21, ca) Affected DD (T20, T21, ca) Affected CD (T20, T14)
ADD T21: <3, 4, Withdrawal(w), (w > ca), Display("No money in ATM machine. Available Cash in ATM = "); Display(ca), >	Affecting DD (T1, T21, ca) Affecting DD (T1, T21, ca) Affecting DD (T20, T21, ca) Affecting DD (T20, T21, ca) Affecting CD (T6, T21) Affected CD (T21, T14)
ADD T22: <3, 4, Withdrawal(w), (w > 400), Display("Withdraw must be less than 400 dollars."), >	Affecting CD (T6, T22) Affected CD (T22, T14)

ADD T23: <3, 4, Withdrawal(w), (w < 1), Display("Withdraw must be greater than 1 dollar."), >	Affecting CD (T6, T23) Affected CD (T23, T14)
DELETE T13	Affecting GDD (T1, T13, cb) Affecting GDD (T1, T13, cb) Affecting GDD (T11, T13, cb) Affecting GDD (T12, T13, cb) Affecting GDD (T13, T13, cb) Affecting GDD (T13, T13, cb) Affecting GCD (T6, T13) Affected GDD (T13, T10, cb) Affected GDD (T13, T11, cb) Affected GDD (T13, T12, cb) Affected GDD (T13, T13, cb) Affected GDD (T13, T13, cb) Affected GCD (T13, T14)

Table APP-19. Generated RTS for ATM_Tahat model using M_O

Test #	Test Sequences	Expect Result
1	T1 T5 T6 T11 T14 T20 T14 T7 T28	
2	T1 T5 T6 T12 T14 T20 T14 T7 T28	
3	T1 T5 T6 T20 T14 T20 T14 T7 T28	
4	T1 T5 T6 T20 T14 T10 T14 T7 T28	
5	T1 T5 T6 T20 T14 T11 T14 T7 T28	
6	T1 T5 T6 T20 T14 T12 T14 T7 T28	
7	T1 T5 T6 T20 T14 T21 T14 T7 T28	
8	T1 T5 T6 T21 T14 T7 T28	
9	T1 T5 T6 T22 T14 T7 T28	
10	T1 T5 T6 T23 T14 T7 T28	
11	T1 T5 T6 T13dummy T14 T7 T28	We expect this test case to fail

12	T1 T5 T6 T11 T14 T13dummy T14 T7 T28	We expect this test case to fail
13	T1 T5 T6 T12 T14 T13dummy T14 T7 T28	We expect this test case to fail
14	T1 T5 T6 T13dummy T14 T13dummy T14 T7 T28	We expect this test case to fail
15	T1 T5 T6 T13dummy T14 T10 T14 T7 T28	We expect this test case to fail
16	T1 T5 T6 T13dummy T14 T11 T14 T7 T28	We expect this test case to fail
17	T1 T5 T6 T13dummy T14 T12 T14 T7 T28	We expect this test case to fail

Table APP-20. M_C and identified NDPMs for ATM_Tahat model

EMs	NDPMs
CHANGE T13 from: <3, 4, Withdrawal(w), (w < cb), , cb :=cb – w> to: <3, 4, Withdrawal(w), (ca >= w) and (w <= cb) and (w >= 1) and (w <= 400), , cb := cb -w; ca := ca -2>	Affecting DD (T1, T13, ca) Affecting DD (T1, T13, ca) Affecting DD (T13, T13, ca) Affecting DD (T13, T13, ca) Affected DD (T13, T13, ca) Affected DD (T13, T13, ca) Affected DD (T13, T21, ca) Affected DD (T13, T21, ca)
ADD T21: <3, 4, Withdrawal(w), (w > ca), Display("No money in ATM machine. Available Cash in ATM = "); Display(ca), >	Affecting DD (T1, T21, ca) Affecting DD (T1, T21, ca) Affecting DD (T13, T21, ca) Affecting DD (T13, T21, ca) Affecting CD (T6, T21) Affected CD (T21, T14)
ADD T22: <3, 4, Withdrawal(w), (w > 400), Display("Withdraw must be less than 400 dollars."), >	Affecting CD (T6, T22) Affected CD (T22, T14)
ADD T23: <3, 4, Withdrawal(w), (w < 1), Display("Withdraw must be greater than 1 dollar."), >	Affecting CD (T6, T23) Affected CD (T23, T14)

Table APP-21. Generated RTS for ATM_Tahat model using M_C

Test #	Test Sequences
1	T1 T5 T6 T13 T14 T13 T14 T7 T28
2	T1 T5 T6 T13 T14 T21 T14 T7 T28
3	T1 T5 T6 T21 T14 T7 T28
4	T1 T5 T6 T22 T14 T7 T28
5	T1 T5 T6 T23 T14 T7 T28

Note: None of the test case in this table is expected to fail

2. Cruise Control System model

The original EFSM model of the Cruise Control system:

T1: <0, 1, EngineOn(), , , CCS := Off>

T2: <1, 2, SetSpeed(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed >= MinSpeed) and (CurrentSpeed <= MaxSpeed) and (BrakeR == True) and (AccelR == True), , CruiseSpeed := CurrentSpeed; CCS := Activated; write(CruiseSpeed)>

T3: <2, 2, SensorSpeed(CurrentSpeed), CurrentSpeed == CruiseSpeed, Write(CurrentSpeed), >

T4: <2, 2, SensorSpeed(CurrentSpeed), CurrentSpeed > CruiseSpeed, Write("Your Speed is over cruising speed, Decreasing Speed"), decr := CurrentSpeed - CruiseSpeed; CurrentSpeed := CurrentSpeed - decr; Write(CurrentSpeed)>

T5: <2, 2, SensorSpeed(CurrentSpeed), (CurrentSpeed < CruiseSpeed) and (CurrentSpeed >= DeactivatingSpeed), Write("Your Speed is lower than cruising speed, Increasing Speed to crusie speed"), inc := CruiseSpeed-CurrentSpeed; CurrentSpeed := CurrentSpeed - CruiseSpeed; write(CurrentSpeed)>

T6: <2, 2, Accel(CurrentSpeed), CurrentSpeed < MaxSpeed, Write("Your Speed is lower than Max speed, Increasing Speed by 1 mile each Accel"), CruiseSpeed := CruiseSpeed + 1>

T7: <2, 2, Coast(CurrentSpeed), CurrentSpeed >= MinSpeed, Write("Your Speed is greater than Min Speed limit, decreasing Speed"), CruiseSpeed := CruiseSpeed - StepSpeed;write(CruiseSpeed)>

T8: <2, 3, SensorSpeed(CurrentSpeed), CurrentSpeed < DeactivatingSpeed, Write("Current Speed less than 35 mile, Cruise Control Speed: CCS Suspended"), CCS := Suspended; Write(CurrentSpeed); CurrentSpeed := CruiseSpeed - CurrentSpeed>

T9: <2, 3, Brake(BrakeR), , , CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>

T10: <2, 3, Accelerator(AccelR), , , CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>

T11: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed < MaxSpeed), , CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed >

T13: <2, 4, Off(), , , CCS := Off>

T14: <1, 4, Off(), , , CCS := Off>

T20: <3, 4, Off(), , , CCS := Off>

Table APP-22. M_O and identified NDPMs for Cruise Control System model

EMs	NDPMs
ADD T15: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed > MinSpeed) and (CurrentSpeed <= MaxSpeed) and (BrakeR == True) and (AccelR == True), Write("CCS Activated"), CCS := Activated; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T2, T15, CruiseSpeed) Affecting DD (T6, T15, CruiseSpeed) Affecting DD (T7, T15, CruiseSpeed) Affecting DD (T8, T15, CurrentSpeed) Affecting DD (T8, T15, CurrentSpeed) Affecting DD (T8, T15, CurrentSpeed) Affecting DD (T8, T15, CurrentSpeed) Affecting DD (T9, T15, CurrentSpeed) Affecting DD (T9, T15, CurrentSpeed) Affecting DD (T9, T15, CurrentSpeed) Affecting DD (T9, T15, CurrentSpeed) Affecting DD (T9, T15, BrakeR) Affecting DD (T10, T15, AccelR) Affecting DD (T10, T15, CurrentSpeed) Affecting DD (T10, T15, CurrentSpeed)

	Affecting DD (T10, T15, CurrentSpeed)
	Affecting DD (T10, T15, CurrentSpeed)
	Affecting CD (T8, T15)
	Affecting CD (T9, T15)
	Affecting CD (T10, T15)
	Affected DD (T15, T3, CurrentSpeed)
	Affected DD (T15, T3, CurrentSpeed)
	Affected DD (T15, T4, CurrentSpeed)
	Affected DD (T15, T4, CurrentSpeed)
	Affected DD (T15, T4, CurrentSpeed)
	Affected DD (T15, T5, CurrentSpeed)
	Affected DD (T15, T5, CurrentSpeed)
	Affected DD (T15, T5, CurrentSpeed)
	Affected DD (T15, T5, CurrentSpeed)
	Affected DD (T15, T6, CurrentSpeed)
	Affected DD (T15, T7, CurrentSpeed)
	Affected DD (T15, T8, CurrentSpeed)
	Affected DD (T15, T8, CurrentSpeed)
	Affected DD (T15, T8, CurrentSpeed)
	Affected DD (T15, T9, CurrentSpeed)
	Affected DD (T15, T10, CurrentSpeed)
	Affected CD (T15, T3)
	Affected CD (T15, T4)
	Affected CD (T15, T5)
	Affected CD (T15, T6)
	Affected CD (T15, T7)
	Affected CD (T15, T8)
	Affected CD (T15, T9)
	Affected CD (T15, T10)

	Affecting DD (T2, T16, CruiseSpeed)
	Affecting DD (T6, T16, CruiseSpeed)
	Affecting DD (T7, T16, CruiseSpeed)
	Affecting DD (T8, T16, CurrentSpeed)
	Affecting DD (T8, T16, CurrentSpeed)
	Affecting DD (T8, T16, CurrentSpeed)
	Affecting DD (T9, T16, CurrentSpeed)
	Affecting DD (T9, T16, CurrentSpeed)
	Affecting DD (T9, T16, CurrentSpeed)
	Affecting DD (T10, T16, CurrentSpeed)
	Affecting DD (T10, T16, CurrentSpeed)
	Affecting DD (T10, T16, CurrentSpeed)
	Affecting CD (T8, T16)
	Affecting CD (T9, T16)
	Affecting CD (T10, T16)
	Affected DD (T16, T3, CurrentSpeed)
	Affected DD (T16, T3, CurrentSpeed)
	Affected DD (T16, T4, CurrentSpeed)
	Affected DD (T16, T4, CurrentSpeed)
	Affected DD (T16, T4, CurrentSpeed)
	Affected DD (T16, T5, CurrentSpeed)
	Affected DD (T16, T5, CurrentSpeed)
	Affected DD (T16, T5, CurrentSpeed)
	Affected DD (T16, T5, CurrentSpeed)
	Affected DD (T16, T6, CurrentSpeed)
	Affected DD (T16, T7, CurrentSpeed)
	Affected DD (T16, T8, CurrentSpeed)
	Affected DD (T16, T8, CurrentSpeed)
	Affected DD (T16, T8, CurrentSpeed)
	Affected DD (T16, T9, CurrentSpeed)
	Affected DD (T16, T10, CurrentSpeed)

	Affected CD (T16, T3) Affected CD (T16, T4) Affected CD (T16, T5) Affected CD (T16, T6) Affected CD (T16, T7) Affected CD (T16, T8) Affected CD (T16, T9) Affected CD (T16, T10)
ADD T17: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed > MaxSpeed), Write("T17: CCS Suspended"), CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T2, T17, CruiseSpeed) Affecting DD (T6, T17, CruiseSpeed) Affecting DD (T7, T17, CruiseSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting CD (T8, T17) Affecting CD (T9, T17) Affecting CD (T10, T17) Affected DD (T17, T3, CurrentSpeed) Affected DD (T17, T3, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed)

	Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T6, CurrentSpeed) Affected DD (T17, T7, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T9, CurrentSpeed) Affected DD (T17, T10, CurrentSpeed) Affected CD (T17, T3) Affected CD (T17, T4) Affected CD (T17, T5) Affected CD (T17, T6) Affected CD (T17, T7) Affected CD (T17, T8) Affected CD (T17, T9) Affected CD (T17, T10)
ADD T18: <3, 2, Resume(CurrentSpeed, 0, 0), (CurrentSpeed >= MinSpeed) and (CurrentSpeed <= MaxSpeed) and (BrakeR == False) and (AccelR == False), Write("T18: CCS Suspended"), CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T2, T18, CruiseSpeed) Affecting DD (T6, T18, CruiseSpeed) Affecting DD (T7, T18, CruiseSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, BrakeR) Affecting DD (T10, T18, AccelR) Affecting DD (T10, T18, CurrentSpeed) Affecting DD (T10, T18, CurrentSpeed)

	Affecting DD (T10, T18, CurrentSpeed) Affecting DD (T10, T18, CurrentSpeed) Affecting CD (T8, T18) Affecting CD (T9, T18) Affecting CD (T10, T18) Affected DD (T18, T3, CurrentSpeed) Affected DD (T18, T3, CurrentSpeed) Affected DD (T18, T4, CurrentSpeed) Affected DD (T18, T4, CurrentSpeed) Affected DD (T18, T4, CurrentSpeed) Affected DD (T18, T5, CurrentSpeed) Affected DD (T18, T5, CurrentSpeed) Affected DD (T18, T5, CurrentSpeed) Affected DD (T18, T5, CurrentSpeed) Affected DD (T18, T6, CurrentSpeed) Affected DD (T18, T7, CurrentSpeed) Affected DD (T18, T8, CurrentSpeed) Affected DD (T18, T8, CurrentSpeed) Affected DD (T18, T8, CurrentSpeed) Affected DD (T18, T9, CurrentSpeed) Affected DD (T18, T10, CurrentSpeed) Affected CD (T18, T3) Affected CD (T18, T4) Affected CD (T18, T5) Affected CD (T18, T6) Affected CD (T18, T7) Affected CD (T18, T8) Affected CD (T18, T9) Affected CD (T18, T10)
DELETE T11	Affecting GDD (T2, T11, CruiseSpeed) Affecting GDD (T6, T11, CruiseSpeed)

	Affecting GDD (T7, T11, CruiseSpeed)
	Affecting GDD (T8, T11, CurrentSpeed)
	Affecting GDD (T8, T11, CurrentSpeed)
	Affecting GDD (T8, T11, CurrentSpeed)
	Affecting GDD (T9, T11, CurrentSpeed)
	Affecting GDD (T9, T11, CurrentSpeed)
	Affecting GDD (T9, T11, CurrentSpeed)
	Affecting GDD (T10, T11, CurrentSpeed)
	Affecting GDD (T10, T11, CurrentSpeed)
	Affecting GDD (T10, T11, CurrentSpeed)
	Affecting GCD (T8, T11)
	Affecting GCD (T9, T11)
	Affecting GCD (T10, T11)
	Affected GDD (T11, T3, CurrentSpeed)
	Affected GDD (T11, T3, CurrentSpeed)
	Affected GDD (T11, T4, CurrentSpeed)
	Affected GDD (T11, T4, CurrentSpeed)
	Affected GDD (T11, T4, CurrentSpeed)
	Affected GDD (T11, T5, CurrentSpeed)
	Affected GDD (T11, T5, CurrentSpeed)
	Affected GDD (T11, T5, CurrentSpeed)
	Affected GDD (T11, T5, CurrentSpeed)
	Affected GDD (T11, T6, CurrentSpeed)
	Affected GDD (T11, T7, CurrentSpeed)
	Affected GDD (T11, T8, CurrentSpeed)
	Affected GDD (T11, T8, CurrentSpeed)
	Affected GDD (T11, T8, CurrentSpeed)
	Affected GDD (T11, T9, CurrentSpeed)
	Affected GDD (T11, T10, CurrentSpeed)
	Affected GCD (T11, T3)
	Affected GCD (T11, T4)

	Affected GCD (T11, T5) Affected GCD (T11, T6) Affected GCD (T11, T7) Affected GCD (T11, T8) Affected GCD (T11, T9) Affected GCD (T11, T10)

Table APP-23. Generated RTS for Cruise Control System model using M_O

Test #	Test Sequences	Expect Result
1	T1 T2 T6 T8 T15 T13	
2	T1 T2 T7 T8 T15 T13	
3	T1 T2 T3 T9 T15 T13	
4	T1 T2 T3 T10 T15 T13	
5	T1 T2 T3 T8 T15 T3 T13	
6	T1 T2 T3 T8 T15 T4 T13	
7	T1 T2 T3 T8 T15 T5 T13	
8	T1 T2 T3 T8 T15 T6 T13	
9	T1 T2 T3 T8 T15 T7 T13	
10	T1 T2 T3 T8 T15 T8 T13	
11	T1 T2 T3 T8 T15 T9 T13	
12	T1 T2 T3 T8 T15 T10 T13	
13	T1 T2 T6 T8 T16 T13	
14	T1 T2 T7 T8 T16 T13	
15	T1 T2 T3 T9 T16 T13	
16	T1 T2 T3 T10 T16 T13	
17	T1 T2 T3 T8 T16 T3 T13	
18	T1 T2 T3 T8 T16 T4 T13	
19	T1 T2 T3 T8 T16 T5 T13	
20	T1 T2 T3 T8 T16 T6 T13	
21	T1 T2 T3 T8 T16 T7 T13	

22	T1 T2 T3 T8 T16 T8 T13	
23	T1 T2 T3 T8 T16 T9 T13	
34	T1 T2 T3 T8 T16 T10 T13	
25	T1 T2 T6 T8 T17 T13	
26	T1 T2 T7 T8 T17 T13	
27	T1 T2 T3 T9 T17 T13	
28	T1 T2 T3 T10 T17 T13	
29	T1 T2 T3 T8 T17 T3 T13	
30	T1 T2 T3 T8 T17 T4 T13	
31	T1 T2 T3 T8 T17 T5 T13	
32	T1 T2 T3 T8 T17 T6 T13	
33	T1 T2 T3 T8 T17 T7 T13	
34	T1 T2 T3 T8 T17 T8 T13	
35	T1 T2 T3 T8 T17 T9 T13	
36	T1 T2 T3 T8 T17 T10 T13	
37	T1 T2 T6 T8 T18 T13	
38	T1 T2 T7 T8 T18 T13	
39	T1 T2 T3 T9 T18 T13	
40	T1 T2 T3 T10 T18 T13	
41	T1 T2 T3 T8 T18 T3 T13	
42	T1 T2 T3 T8 T18 T4 T13	
43	T1 T2 T3 T8 T18 T5 T13	
44	T1 T2 T3 T8 T18 T6 T13	
45	T1 T2 T3 T8 T18 T7 T13	
46	T1 T2 T3 T8 T18 T8 T13	
47	T1 T2 T3 T8 T18 T9 T13	
48	T1 T2 T3 T8 T18 T10 T13	
49	T1 T2 T6 T8 T11dummy T13	We expect this test case to fail
50	T1 T2 T7 T8 T11dummy T13	We expect this test case to fail
51	T1 T2 T3 T9 T11dummy T13	We expect this test case to fail

52	T1 T2 T3 T10 T11dummy T13	We expect this test case to fail
53	T1 T2 T3 T8 T11dummy T3 T13	We expect this test case to fail
54	T1 T2 T3 T8 T11dummy T4 T13	We expect this test case to fail
55	T1 T2 T3 T8 T11dummy T5 T13	We expect this test case to fail
56	T1 T2 T3 T8 T11dummy T6 T13	We expect this test case to fail
57	T1 T2 T3 T8 T11dummy T7 T13	We expect this test case to fail
58	T1 T2 T3 T8 T11dummy T8 T13	We expect this test case to fail
59	T1 T2 T3 T8 T11dummy T9 T13	We expect this test case to fail
60	T1 T2 T3 T8 T11dummy T10 T13	We expect this test case to fail

Table APP-24. M_C and identified NDPMs for Cruise Control System model

EMs	NDPMs
Change T11 from:<3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed < MaxSpeed), , CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed > to: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed > MinSpeed) and (CurrentSpeed <= MaxSpeed) and (BrakeR == True) and (AccelR == True), Write("CCS Activated"), CCS := Activated; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T8, T11, CurrentSpeed) Affecting DD (T9, T11, CurrentSpeed) Affecting DD (T9, T11, BrakeR) Affecting DD (T10, T11, AccelR) Affecting DD (T10, T11, CurrentSpeed)

ADD T16: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed <= MinSpeed), Write("T16: CCS Suspended"), CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed >	Affecting DD (T2, T16, CruiseSpeed) Affecting DD (T6, T16, CruiseSpeed) Affecting DD (T7, T16, CruiseSpeed) Affecting DD (T8, T16, CurrentSpeed) Affecting DD (T8, T16, CurrentSpeed) Affecting DD (T8, T16, CurrentSpeed) Affecting DD (T9, T16, CurrentSpeed) Affecting DD (T9, T16, CurrentSpeed) Affecting DD (T9, T16, CurrentSpeed) Affecting DD (T10, T16, CurrentSpeed) Affecting DD (T10, T16, CurrentSpeed) Affecting DD (T10, T16, CurrentSpeed) Affecting CD (T8, T16) Affecting CD (T9, T16) Affecting CD (T10, T16) Affected DD (T16, T3, CurrentSpeed) Affected DD (T16, T3, CurrentSpeed) Affected DD (T16, T4, CurrentSpeed) Affected DD (T16, T4, CurrentSpeed) Affected DD (T16, T4, CurrentSpeed) Affected DD (T16, T5, CurrentSpeed) Affected DD (T16, T5, CurrentSpeed) Affected DD (T16, T5, CurrentSpeed) Affected DD (T16, T5, CurrentSpeed) Affected DD (T16, T6, CurrentSpeed) Affected DD (T16, T7, CurrentSpeed) Affected DD (T16, T8, CurrentSpeed) Affected DD (T16, T8, CurrentSpeed) Affected DD (T16, T8, CurrentSpeed) Affected DD (T16, T9, CurrentSpeed) Affected DD (T16, T10, CurrentSpeed)

	Affected CD (T16, T3) Affected CD (T16, T4) Affected CD (T16, T5) Affected CD (T16, T6) Affected CD (T16, T7) Affected CD (T16, T8) Affected CD (T16, T9) Affected CD (T16, T10)
ADD T17: <3, 2, Resume(CurrentSpeed, BrakeR, AccelR), (CurrentSpeed > MaxSpeed), Write("T17: CCS Suspended"), CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T2, T17, CruiseSpeed) Affecting DD (T6, T17, CruiseSpeed) Affecting DD (T7, T17, CruiseSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T8, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T9, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting DD (T10, T17, CurrentSpeed) Affecting CD (T8, T17) Affecting CD (T9, T17) Affecting CD (T10, T17) Affected DD (T17, T3, CurrentSpeed) Affected DD (T17, T3, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T4, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T5, CurrentSpeed)

	Affected DD (T17, T5, CurrentSpeed) Affected DD (T17, T6, CurrentSpeed) Affected DD (T17, T7, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T8, CurrentSpeed) Affected DD (T17, T9, CurrentSpeed) Affected DD (T17, T10, CurrentSpeed) Affected CD (T17, T3) Affected CD (T17, T4) Affected CD (T17, T5) Affected CD (T17, T6) Affected CD (T17, T7) Affected CD (T17, T8) Affected CD (T17, T9) Affected CD (T17, T10)
ADD T18: <3, 2, Resume(CurrentSpeed, 0, 0), (CurrentSpeed >= MinSpeed) and (CurrentSpeed <= MaxSpeed) and (BrakeR == False) and (AccelR == False), Write("T18: CCS Suspended"), CCS := Suspended; CurrentSpeed := CruiseSpeed - CurrentSpeed>	Affecting DD (T2, T18, CruiseSpeed) Affecting DD (T6, T18, CruiseSpeed) Affecting DD (T7, T18, CruiseSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T8, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, CurrentSpeed) Affecting DD (T9, T18, BrakeR) Affecting DD (T10, T18, AccelR) Affecting DD (T10, T18, CurrentSpeed) Affecting DD (T10, T18, CurrentSpeed)

	Affecting DD (T10, T18, CurrentSpeed)
	Affecting DD (T10, T18, CurrentSpeed)
	Affecting CD (T8, T18)
	Affecting CD (T9, T18)
	Affecting CD (T10, T18)
	Affected DD (T18, T3, CurrentSpeed)
	Affected DD (T18, T3, CurrentSpeed)
	Affected DD (T18, T4, CurrentSpeed)
	Affected DD (T18, T4, CurrentSpeed)
	Affected DD (T18, T4, CurrentSpeed)
	Affected DD (T18, T5, CurrentSpeed)
	Affected DD (T18, T5, CurrentSpeed)
	Affected DD (T18, T5, CurrentSpeed)
	Affected DD (T18, T5, CurrentSpeed)
	Affected DD (T18, T6, CurrentSpeed)
	Affected DD (T18, T7, CurrentSpeed)
	Affected DD (T18, T8, CurrentSpeed)
	Affected DD (T18, T8, CurrentSpeed)
	Affected DD (T18, T8, CurrentSpeed)
	Affected DD (T18, T9, CurrentSpeed)
	Affected DD (T18, T10, CurrentSpeed)
	Affected CD (T18, T3)
	Affected CD (T18, T4)
	Affected CD (T18, T5)
	Affected CD (T18, T6)
	Affected CD (T18, T7)
	Affected CD (T18, T8)
	Affected CD (T18, T9)
	Affected CD (T18, T10)

Table APP-25. Generated RTS for Cruise Control System model using M_C

Test #	Test Sequences
1	T1 T2 T8 T11 T13
2	T1 T2 T9 T11 T13
3	T1 T2 T10 T11 T13
4	T1 T2 T6 T8 T16 T13
5	T1 T2 T7 T8 T16 T13
6	T1 T2 T3 T9 T16 T13
7	T1 T2 T3 T10 T16 T13
8	T1 T2 T3 T8 T16 T3 T13
9	T1 T2 T3 T8 T16 T4 T13
10	T1 T2 T3 T8 T16 T5 T13
11	T1 T2 T3 T8 T16 T6 T13
12	T1 T2 T3 T8 T16 T7 T13
13	T1 T2 T3 T8 T16 T8 T13
14	T1 T2 T3 T8 T16 T9 T13
15	T1 T2 T3 T8 T16 T10 T13
16	T1 T2 T6 T8 T17 T13
17	T1 T2 T7 T8 T17 T13
18	T1 T2 T3 T9 T17 T13
19	T1 T2 T3 T10 T17 T13
20	T1 T2 T3 T8 T17 T3 T13
21	T1 T2 T3 T8 T17 T4 T13
22	T1 T2 T3 T8 T17 T5 T13
23	T1 T2 T3 T8 T17 T6 T13
24	T1 T2 T3 T8 T17 T7 T13
25	T1 T2 T3 T8 T17 T8 T13
26	T1 T2 T3 T8 T17 T9 T13
27	T1 T2 T3 T8 T17 T10 T13
28	T1 T2 T6 T8 T18 T13

29	T1 T2 T7 T8 T18 T13
30	T1 T2 T3 T9 T18 T13
31	T1 T2 T3 T10 T18 T13
32	T1 T2 T3 T8 T18 T3 T13
33	T1 T2 T3 T8 T18 T4 T13
34	T1 T2 T3 T8 T18 T5 T13
35	T1 T2 T3 T8 T18 T6 T13
36	T1 T2 T3 T8 T18 T7 T13
37	T1 T2 T3 T8 T18 T8 T13
38	T1 T2 T3 T8 T18 T9 T13
39	T1 T2 T3 T8 T18 T10 T13

Note: None of the test case in this table is expected to fail

3. Fuel Pump model

The original EFSM model of the Fuel Pump system:

T1: <0, 1, Activate(Rprice, Mprice, Sprice), , , Regular := 1; Medium := 1; Super := 1; total := 0>

T2: <1, 2, Cash(), , Write("Insert Credit Card"), >

T3: <1, 3, Credit(), , Write("Please Insert Your Credit Card"), >

T4: <3, 1, Reject(), , Write("Invalid Credit Card"), Reject Card>

T5: <2, 4, Accept(), , Write("Select Fuel Menu"), >

T6: <3, 4, Accept(), , Write("Select Fuel Menu"), >

T7: <4, 5, Regular(), , Write("Pull Handle to pump Gas"); Write("Push Start"), Regular := 1>

T8: <4, 5, Medium(), , Write("Pull Handle to pump Gas"); Write("Push Start"), Medium := 1>

T9: <4, 5, Super(), , Write("Pull Handle to pump Gas"); Write("Push Start"), Super := 1>

T10: <5, 6, Handle(), , Write("Start Filling! push the lever"), >

T11: <6, 7, Lever(), , Write("Pumping in Progress"), >

T12: <7, 7, Lever(), , Write("Pumping in Progress"), >

T13: <7, 8, Flowoff(), , Write("Stop Pumping"), >

T14: <8, 7, Lever(), , Write("Start Pumping"), >

T15: <8, 8, Handeloff(), , Write("Pumping Stop"), >

T16: <8, 9, Handelon(), , Write("Do You Want Car Wash"), >

T17: <9, 10, Carwash(carwash, amount), carwash == False, , Charge := amount * Rprice; total := total + Charge; Write(Charge)>

T18: <9, 10, Carwash(carwash, amount), carwash == False, , Charge := amount * Mprice; total := total + Charge; Write(Charge)>

T19: <9, 10, Carwash(carwash, amount), carwash == False, , Charge := amount * Sprice; total := total + Charge; Write(Charge)>

T20: <9, 10, Carwash(carwash, amount), (carwash == True) and (Medium == True), , Charge := amount * Rprice + 5; total := total + Charge; Write(Charge)>

T25: <10, 11, Receipt(), , Write(Charge); Write("Thank You for Shoping"); Write(" Come back again"), >

T28: <11, 12, Exit(), , Write("Thank You for Shopping"), >

Table APP-26. M_O and identified NDPMs for Fuel Pump model

EMs	NDPMs
ADD T21: <9, 10, Carwash(carwash, amount), (carwash == True) and (Regular == True) and (amount <= 40), , Charge := amount * Rprice + 5; total := total + Charge; Write(Charge)>	Affecting DD (T1, T21, Rprice) Affecting DD (T1, T21, Regular) Affecting DD (T1, T21, total) Affecting DD (T7, T21, Regular) Affected DD (T21, T25, Charge)
ADD T22: <9, 10, Carwash(carwash, amount), (carwash == True) and (Medium == True) and (amount > 40) and (amount <= 60), , Charge := amount * Mprice + 4.75; total := total + Charge; Write(Charge)>	Affecting DD (T1, T22, Mprice) Affecting DD (T1, T22, Medium) Affecting DD (T1, T22, total) Affecting DD (T8, T22, Medium) Affected DD (T22, T25, Charge)
ADD T23: <9, 10, Carwash(carwash, amount), (carwash == True) and (Super == True) and (amount > 60) and (amount <= 100), ,	Affecting DD (T1, T23, Sprice) Affecting DD (T1, T23, Super) Affecting DD (T1, T23, total)

Charge := amount * Sprice + 5; total := total + Charge; Write(Charge)>	Affecting DD (T9, T23, Super) Affected DD (T23, T25, Charge)
ADD T24: <9, 10, Carwash(carwash, amount), (carwash == True) and (Super == True) and (amount > 100), , Charge := amount * Sprice + 5; total := total + Charge; Write(Charge) >	Affecting DD (T1, T24, Sprice) Affecting DD (T1, T24, Super) Affecting DD (T1, T24, total) Affecting DD (T9, T24, Super) Affected DD (T24, T25, Charge)
DELETE T20	Affecting GDD (T1, T20, Rprice) Affecting GDD (T1, T20, Medium) Affecting GDD (T1, T20, total) Affecting GDD (T8, T20, Medium) Affected GDD (T20, T25, Charge)

Table APP-27. Generated RTS for Fuel Pump model using M_O

Test #	Test Sequences	Expect Result
1	T1 T3 T6 T7 T11 T13 T16 T21 T25 T28	
2	T1 T3 T6 T8 T11 T13 T16 T22 T25 T28	
3	T1 T3 T6 T9 T11 T13 T16 T23 T25 T28	
4	T1 T3 T6 T9 T11 T13 T16 T24 T25 T28	
5	T1 T3 T6 T8 T11 T13 T16 T20dummy T25 T28	We expect this test case to fail

Table APP-28. M_C and identified NDPMs for Fuel Pump model

EMs	NDPMs
Change T20 from: <9, 10, Carwash(carwash, amount), (carwash == True) and (Medium == True), , Charge := amount * Rprice + 5; total := total + Charge; Write(Charge)> to: <9, 10, Carwash(carwash, amount),	Affecting DD (T1, T20, Regular) Affecting DD (T7, T20, Regular) Affecting GDD (T1, T20, Medium) Affecting GDD (T8, T20, Medium)

(carwash == True) and (Regular == True) and (amount <= 40), , Charge := amount * Rprice + 5; total := total + Charge; Write(Charge)>	
ADD T22: <9, 10, Carwash(carwash, amount), (carwash == True) and (Medium == True) and (amount > 40) and (amount <= 60), , Charge := amount * Mprice + 4.75; total := total + Charge; Write(Charge)>	Affecting DD (T1, T22, Mprice) Affecting DD (T1, T22, Medium) Affecting DD (T1, T22, total) Affecting DD (T8, T22, Medium) Affected DD (T22, T25, Charge)
ADD T23: <9, 10, Carwash(carwash, amount), (carwash == True) and (Super == True) and (amount > 60) and (amount <= 100), , Charge := amount * Sprice + 5; total := total + Charge; Write(Charge)>	Affecting DD (T1, T23, Sprice) Affecting DD (T1, T23, Super) Affecting DD (T1, T23, total) Affecting DD (T9, T23, Super) Affected DD (T23, T25, Charge)
ADD T24: <9, 10, Carwash(carwash, amount), (carwash == True) and (Super == True) and (amount > 100), , Charge := amount * Sprice + 5; total := total + Charge; Write(Charge) >	Affecting DD (T1, T24, Sprice) Affecting DD (T1, T24, Super) Affecting DD (T1, T24, total) Affecting DD (T9, T24, Super) Affected DD (T24, T25, Charge)

Table APP-29. Generated RTS for Fuel Pump model using M_C

Test #	Test Sequences
1	T1 T3 T6 T8 T11 T13 T16 T20 T25 T28
2	T1 T3 T6 T7 T11 T13 T16 T20 T25 T28
3	T1 T3 T6 T8 T11 T13 T16 T22 T25 T28
4	T1 T3 T6 T9 T11 T13 T16 T23 T25 T28
5	T1 T3 T6 T9 T11 T13 T16 T24 T25 T28

Note: None of the test case in this table is expected to fail

4. ISDN model

The original EFSM model of the ISDN system:

T1: <0, 1, SETUP(), , , lineNum := 0>

T2: <1, 2, SETUP(), , , lineNum := 0>

T3: <2, 3, MOREINFOREQUEST(), , , >

T4: <2, 4, PROCEEDINGREQUEST(), , , >

T6: <3, 4, CALLPROCEEDINGREQUEST(), , , >

T7: <4, 4, PROGRESSREQUEST(), , , >

T8: <2, 1, REJECTREQUEST(), , , >

T9: <3, 3, INFOREQUEST(), , , >

T10: <4, 10, SETUPRESPONSE(), , , >

T11: <4, 5, ALERTINGREQUEST(), , , >

T12: <5, 10, SETUPRESPONSE(), , , >

T13: <1, 6, SETUPREQUEST(), , , >

T14: <6, 12, RELEASECOMPLETE(), , , >

T15: <6, 9, CALLPROCEEDING(), , , >

T16: <6, 7, ALERTING(), , , >

T17: <6, 8, CONNECTLINE(lineNum), lineNum > 0, Write("T17 executed: move to state 8"), >

T18: <9, 7, ALERTING(), , , >

T19: <9, 8, CONNECT(), , , >

T20: <7, 8, CONNECT(), , , >

T21: <8, 10, SETUPCOMPLETEREQUEST(), , , >

T22: <11, 15, RELEASEREQUEST(), , , >

T25: <15, 17, RELEASECOMPLETE(), , , >

T26: <1, 14, RESUME(), , , >

T27: <14, 1, RESUMEREJECT(), , , >

T28: <14, 10, RESUMEACK(), , , >

T29: <10, 13, SUSPEND(), , , >

T30: <13, 10, NOTIFYREQUEST(), , , >

T31: <13, 1, SUSPENDRESPONSE(), , , >

T32: <1, 6, SETUPREQUEST(), , , >

T33: <6, 16, PROCEEDING(), , , >

T35: <2, 3, MOREINFOREQUEST(), , , >

T37: <3, 3, PROGRESSREQUEST(), , , >

T38: <5, 5, PROGRESSREQUEST(), , , >

T39: <10, 10, CONNECTACK(), , , >

T40: <10, 10, NOTIFY(), , , >

T41: <12, 17, RELEASE(), , , >

T42: <12, 17, DISCONNECT(), , , >

T43: <13, 10, SUSPENDREJECTREQUEST(), , , >

T44: <14, 10, RESUMERESPONSE(), , , >

T45: <14, 1, RESUMEREJECTREQUEST(), , , >

T46: <15, 17, RELEASE(), , , >

T47: <16, 16, MOREINFOREQUEST(), , , >

T48: <16, 7, ALERTING(), , , >

T49: <16, 8, CONNECT(), , , >

T50: <16, 9, CALLPROCEEDING(), , , >

T51: <16, 16, PROGRESS(), , , >

T53: <10, 14, RESUME(), , , >

T56: <15, 17, DISCONNECT(), , , >

T57: <9, 9, PROGRESS(), , , >

T58: <16, 9, SETUPACK(), , , >

T59: <1, 11, DISCONNECT(), , , >

T60: <2, 11, DISCONNECT(), , , >

T61: <3, 11, DISCONNECT(), , , >

T62: <4, 11, DISCONNECT(), , , >

T63: <5, 11, DISCONNECT(), , , >

T64: <6, 11, DISCONNECT(), , , >

T65: <7, 11, DISCONNECT(), , , >

T66: <8, 11, DISCONNECT(), , , >

T67: <9, 11, DISCONNECT(), , , >

T68: <10, 11, DISCONNECT(), , , >

T69: <13, 11, DISCONNECT(), , , >

T70: <14, 11, DISCONNECT(), , , >

T71: <16, 11, DISCONNECT(), , , >

T72: <1, 12, DISCONNECTREQUEST(), , , >

T73: <2, 12, DISCONNECTREQUEST(), , , >

T74: <3, 12, DISCONNECTREQUEST(), , , >

T75: <4, 12, DISCONNECTREQUEST(), , , >

T76: <5, 12, DISCONNECTREQUEST(), , , >

T77: <6, 12, DISCONNECTREQUEST(), , , >

T78: <7, 12, DISCONNECTREQUEST(), , , >

T79: <8, 12, DISCONNECTREQUEST(), , , >

T80: <9, 12, DISCONNECTREQUEST(), , , >

T81: <10, 12, DISCONNECTREQUEST(), , , >

T82: <11, 12, DISCONNECTREQUEST(), , , >

T84: <13, 12, DISCONNECTREQUEST(), , , >

T85: <14, 12, DISCONNECTREQUEST(), , , >

T86: <15, 12, DISCONNECTREQUEST(), , , >

T91: <16, 12, DISCONNECTREQUEST(), , , >

T92: <12, 17, DISCONNECTREQUEST(), , , >

Table APP-30. M_O and identified NDPMs for ISDN model

EMs	NDPMs
ADD T87: <6, 8, CONNECTLINE(lineNum), (lineNum >= 0 and lineNum <= 50), Write("Call function1 to connect line"); Write("T87 executed: move to state 8"), >	Affecting CD (T13, T87) Affecting CD (T32, T87) Affected CD (T87, T21) Affected CD (T87, T66) Affected CD (T87, T79)
ADD T88: <6, 8, CONNECTLINE(lineNum), (lineNum > 50 and lineNum <= 100), Write("Call function2 to connect line"); Write("T87 executed: move to state 8"), >	Affecting CD (T13, T88) Affecting CD (T32, T88) Affected CD (T88, T21)

	Affected CD (T88, T66) Affected CD (T88, T79)
ADD T89: <6, 8, CONNECTLINE(lineNum), lineNum >100, Write("Call function3 to connect line"); Write("T87 executed: move to state 8"), >	Affecting CD (T13, T89) Affecting CD (T32, T89) Affected CD (T89, T21) Affected CD (T89, T66) Affected CD (T89, T79)
DELETE T17	Affecting GCD (T13, T17) Affecting GCD (T32, T17) Affected GCD (T17, T21) Affected GCD (T17, T66) Affected GCD (T17, T79)

Table APP-31. Generated RTS for ISDN model using M_O

Test #	Test Sequences	Expect Result
1	T1 T13 T87 T79 T83	
2	T1 T13 T87 T21 T81 T83	
3	T1 T32 T87 T66 T55 T56	
4	T1 T13 T88 T79 T83	
5	T1 T13 T88 T21 T81 T83	
6	T1 T32 T88 T66 T55 T56	
7	T1 T13 T89 T79 T83	
8	T1 T13 T89 T21 T81 T83	
9	T1 T32 T89 T66 T55 T56	
10	T1 T13 T17dummy T79 T83	We expect this test case to fail
11	T1 T13 T17dummy T21 T81 T83	We expect this test case to fail
12	T1 T32 T17dummy T66 T55 T56	We expect this test case to fail

Table APP-32. M_C and identified NDPMs for ISDN model

EMs	NDPMs
Change T17 from: <6, 8, CONNECTLINE(lineNum), lineNum > 0, , Write("T17 executed: move to state 8")> to: <6, 8, CONNECTLINE(lineNum), (lineNum >= 0 and lineNum <= 50), Write("Call function1 to connect line"); Write("T87 executed: move to state 8"), >	NONE
ADD T88: <6, 8, CONNECTLINE(lineNum), (lineNum > 50 and lineNum <= 100), Write("Call function2 to connect line"); Write("T87 executed: move to state 8"), >	Affecting CD (T13, T88) Affecting CD (T32, T88) Affected CD (T88, T21) Affected CD (T88, T66) Affected CD (T88, T79)
ADD T89: <6, 8, CONNECTLINE(lineNum), lineNum > 100, Write("Call function3 to connect line"); Write("T87 executed: move to state 8"), >	Affecting CD (T13, T89) Affecting CD (T32, T89) Affected CD (T89, T21) Affected CD (T89, T66) Affected CD (T89, T79)

Table APP-33. Generated RTS for ISDN model using M_C

Test #	Test Sequences
1	T1 T13 T17 T79 T83
2	T1 T13 T88 T79 T83
3	T1 T32 T88 T79 T83
4	T1 T13 T88 T21 T81 T83
5	T1 T32 T88 T66 T55 T56
6	T1 T13 T89 T79 T83

7	T1 T32 T89 T79 T83
8	T1 T13 T89 T21 T81 T83
9	T1 T32 T89 T66 T55 T56

Note: None of the test case in this table is expected to fail

5. TCP_Dailer model

The original EFSM model of the TCP Dialer system:

T1: <0, 1, Open(Connect, Optype), Connect == 0, Write("This is a startup state"), Dialup := 1; Mode := Optype; attempts := 0>

T2: <0, 7, Open(Connect, Optype), (Connect == 1) and (Optype == Passive), Write("Syn."), Dialup := 0; Mode := Passive>

T3: <0, 8, Open(Connect, Optype), (Connect == 1) and (Optype == Active), Write("Syn."), Mode := Active; Dialup := 0; Interval := Synwait>

T4: <1, 17, Tone(Return), Return == "None", Write("No Dail Tone"), Return("None") >

T5: <1, 4, Tone(Return), Return == "Dial", , Return("Dial")>

T6: <4, 4, Digit(Inbuffer, Digitdial), Inbuffer == 0, , CdPn := Digitdial; Write("Called Party Number=" + CdPn)>

T7: <4, 3, Digit(Inbuffer, Digitdial), Inbuffer == 1, , Interval := Digitdial; Write("Tone [Digitdial]") >

T8: <4, 17, Tone(Return), (Return != Ring) and (Attempts == 3), Write("No Answer"), return>

T9: <4, 5, Tone(Return), Return == "Ring", Write("Ring"), Interval := Ring; Return("Ring")>

T10: <4, 2, Tone(Return), (Return == Busy) or (Return == Reorder) and (Attempts < 3), Write("OffHook-Redial"), Interval := Redial * 2 ; Write(Interval)>

T11: <3, 3, Wait(Interval), Interval != 0, , >

T12: <3, 4, Wait(Interval), Interval == 0, , >

T13: <5, 5, Wait(Interval), Interval != 0, , >

T14: <5, 17, Wait(Interval), (Interval == 0) and (Attempts == 3), Write("OnHook"); Write("No Answer. Retry"), >

T15: <5, 6, Answer(), , , Train := 4; Write(Train)>

T16: <5, 2, Wait(Interval), (Interval == 0) and (Attempts <3), Write("OnHook"); Write("Redail"), >

T17: <6, 17, AckTrain(Return, Train), (Return == 0) and (Train == 0), Write("OnHook"); Write("Can not Connect"), >

T18: <6, 6, AckTrain(Return, Train), (Return == 0) and (Train > 0), Write(Train), Train := Train – 1; Return(0)>

T19: <2, 2, Wait(Interval), Interval != 0, , >

T20: <2, 17, Wait(Interval), Interval == 0, Write("OffHook"), Attempts := Attempts + 1>

T21: <6, 7, AckTrain(Return, Train), (Return == 1) and (Train == 0), , Return(1)>

T22: <6, 8, AckTrain(Return, Train), (Return == 1) and (Train == 1), Write("Syn."), Return(1)>

T24: <8, 9, MsgRecv(Msg, size), Msg == Syn, Write("Syn. ACK"), Packetsent := Packetsent + size>

T25: <8, 10, MsgRecv(Msg, size), (Msg==Ack) and (size >=50), , >

T26: <8, 17, Wait(Interval), Interval == 0, , >

T27: <10, 11, Close(), , Write("FIN"), >

T28: <10, 12, MsgRecv(Msg, size), (Msg == Fin) and (size > 0), Write("ACK"), >

T29: <12, 13, Close(), , Write("FIN"), >

T30: <13, 13, Wait(Interval), Interval != 0, , >

T31: <13, 17, Wait(Interval), Interval == 0, , >

T32: <13, 17, MsgRecv(Msg, size), (Msg == Ack) and (size > 0), Write("ACK"), >

T33: <7, 8, MsgSend(Msg), Msg == Syn, Write("Syn."), >

T34: <7, 9, MsgRecv(Msg, size), (Msg == Syn) and (size > 0), Write("Syn."); Write("ACK"), >

T35: <9, 7, MsgRecv(Msg, size), (Msg == Rst) and (size > 0), , >

T36: <9, 10, MsgRecv(Msg, size), (Msg == Ack) and (size > 0), , >

T37: <9, 11, Close(), , Write("FIN"), >

T38: <11, 15, MsgRecv(Msg, size), (Msg==Fin) and (size > 0), Write("ACK"), >

T39: <11, 16, MsgRecv1(Msg1, Msg2), (Msg1 == Fin) and (Msg2 == Ack), Write("ACK"), >

T40: <11, 14, MsgRecv(Msg, size), (Msg == Ack) and (size > 0), Write("ACK"), >

T41: <16, 16, Wait(Interval), Interval != 0, , >

T42: <16, 17, Wait(Interval), Interval = 0, , >

T43: <15, 16, MsgRecv(Msg, size), (Msg == Ack) and (size > 0), Write("ACK"), >

T44: <14, 16, MsgRecv(Msg, size), (Msg == Fin) and (size > 0), Write("FIN"), >

T50: <16, 17, Exit(), , , >

Table APP-34. M_O and identified NDPMs for TCP_Dailer model

EMs	NDPMs
ADD T47: <8, 10, MsgRecv(Msg, size), ((Msg == Ack) and (size >= 50) and (size < 250)), , Packetsent := Packetsent + size>	Affecting DD (T1, T47, Packetsent) Affecting CD (T3, T47) Affecting CD (T22, T47) Affecting CD (T33, T47) Affected CD (T47, T27) Affected CD (T47, T28)
ADD T48: <8, 10, MsgRecv(Msg, size), ((Msg == Ack) and (size < 50)), Write("msg less than 50- display error"), >	Affecting CD (T3, T48) Affecting CD (T22, T48) Affecting CD (T33, T48) Affected CD (T48, T27) Affected CD (T48, T28)
ADD T49: <8, -1, MsgRecv(Msg, size), (((Msg!=Ack) or (Msg!=Syn)) and (size >= 250)), Write("msg greater than 250- display error"), >	Affecting CD (T3, T49) Affecting CD (T22, T49) Affecting CD (T33, T49) Affected CD (T49, T27) Affected CD (T49, T28)
DELETE T25	Affecting GCD (T3, T25) Affecting GCD (T22, T25) Affecting GCD (T33, T25) Affected CD (T25, T27) Affected CD (T25, T28)

Table APP-35. Generated RTS for TCP_Dailer model using M_O

Test #	Test Sequences	Expect Result
1	T1 T5 T9 T15 T22 T47 T27 T39 T42	
2	T2 T33 T47 T27 T39 T42	
3	T3 T47 T28 T29 T32	
4	T1 T5 T9 T15 T22 T48 T27 T39 T42	
5	T2 T33 T48 T27 T39 T42	
6	T3 T48 T28 T29 T32	
7	T1 T5 T9 T15 T22 T49 T27 T39 T42	
8	T2 T33 T49 T27 T39 T42	
9	T3 T49 T28 T29 T32	
10	T1 T5 T9 T15 T22 T25dummy T27 T39 T42	We expect this test case to fail
11	T3 T25dummy T27 T39 T42	We expect this test case to fail
12	T2 T33 T25dummy T27 T39 T42	We expect this test case to fail

Table APP-36. M_C and identified NDPMs for TCP_Dailer model

EMs	NDPMs
Change T25 from : <8, 10, MsgRecv(Msg, size), (Msg == Ack) and (size >= 50), , > to: <8, 10, MsgRecv(Msg, size), ((Msg == Ack) and (size >= 50) and (size < 250)), , Packetsent := Packetsent + size>	Affecting DD (T1, T25, Packetsent)
ADD T48: <8, 10, MsgRecv(Msg, size), ((Msg == Ack) and (size < 50)), Write("msg less than 50-display error"), >	Affecting CD (T3, T48) Affecting CD (T22, T48) Affecting CD (T33, T48) Affected CD (T48, T27) Affected CD (T48, T28)

ADD T49: <8, -1, MsgRecv(Msg, size), (((Msg != Ack) or (Msg != Syn)) and (size >= 250)), Write("msg greater than 250- display error"), >	Affecting CD (T3, T49) Affecting CD (T22, T49) Affecting CD (T33, T49) Affected CD (T49, T27) Affected CD (T49, T28)

Table APP-37. Generated RTS for TCP_Dailer model using M_C

Test #	Test Sequences
1	T1 T5 T9 T15 T22 T25 T27 T39 T42
2	T1 T5 T9 T15 T22 T48 T27 T39 T42
3	T2 T33 T48 T27 T39 T42
4	T3 T48 T28 T29 T32
5	T1 T5 T9 T15 T22 T49 T27 T39 T42
6	T2 T33 T49 T27 T39 T42
7	T3 T49 T28 T29 T32

Note: None of the test case in this table is expected to fail