Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

On the Influence of Test Adequacy Criteria on Test Suite Reduction for Model-Based Testing of Real-Time Systems

Alan Kelon Oliveira de Moraes

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Patrícia Duarte de Lima Machado e Wilkerson de Lucena Andrade
(Orientadores)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

M827o Moraes, Alan Kelon Oliveira de.

On the influence of test adequacy criteria on test suite reduction for model-based testing of real-time systems / Alan Kelon Oliveira de Moraes. – Campina Grande, 2017.

115 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2017.

"Orientação: Profa. Dra. Patrícia Duarte de Lima Machado, Prof. Dr. Wilkerson de Lucena Andrade".

Referências.

1. Teste de Software. 2. Teste Baseado em Modelos. 3. Sistemas de Tempo-Real. 4. Redução de Suítes de Testes. 5. Critérios de Adequação de Testes. 6. Timed Input-Output Symbolic Transition System. I. Machado, Patrícia Duarte de Lima. II. Andrade, Wilkerson de Lucena. III. Título.

CDU 004.41(043)

“ON THE INFLUENCE OF TEST ADEQUACY CRITERIA ON TEST SUITE REDUCTION FOR MODEL-BASED TESTING OF REAL-TIME SYSTEMS”

ALAN KELON OLIVEIRA DE MORAES

TESE APROVADA EM 31/08/2017

CAMPINA GRANDE – PB

Resumo

O teste baseado em modelos é uma abordagem de teste de software que usa modelos abstratos de uma aplicação para gerar, executar e avaliar os testes. A geração de casos de testes exerce um papel importante no teste baseado em modelos. Como essa geração consiste na busca sistemática por casos de testes que possam ser extraídos dos modelos, o teste baseado em modelos geralmente produz suítes de testes que são caras demais para serem executadas completamente. Técnicas de redução de suítes de testes têm sido propostas para abordar este problema. O objetivo dessas técnicas é obter suítes de testes reduzidas que são mais baratas de serem executadas e tão efetivas na detecção de faltas quanto as suítes completas, dado que as suítes reduzidas mantém o mesmo nível de cobertura, definido por um critério de adequação de testes, da suíte completa. Esses critérios definem que partes do sistema serão testados, com que frequência e sob quais circunstâncias. Entretanto, pouca atenção tem sido dada ao impacto que a escolha do critério tem na redução de suítes de testes. Por outro lado, sistemas de tempo-real são sistemas reativos cujos comportamentos são restringidos pelo tempo. Consequentemente, faltas relacionadas ao tempo são específicas desses sistemas. Para lidar com isso, modelos para sistemas de tempo real devem trabalhar com tempo e, consequentemente, há critérios de adequação de testes específicos para eles. Contudo, a pesquisa sobre redução de suítes de testes não tem focado em sistemas de tempo-real, portanto o impacto de critérios de adequação de testes na redução de suítes é desconhecido. Nesta pesquisa de doutorado objetivamos investigar a influência de critérios de adequação de testes nos resultados da redução de suítes de testes no contexto de teste baseado em modelos de sistemas de tempo-real. Em particular, nós estamos interessados no modelo Timed Input-Output Symbolic Transition Systems (TIOSTS), porque ele é um modelo de sistema de transições no qual dados e tempo são definidos simbolicamente, já que sistemas de transição são a base para o teste de conformidade de sistemas de tempo real. Para alcançar o objetivo da pesquisa, primeiramente, nós definimos 19 critérios de adequação de testes para o modelo TIOSTS. Os critérios definidos incluem critérios baseados em transições, fluxo de dados e tempo. Depois nós formalizamos uma hierarquia com esses critérios, onde eles estão parcialmente ordenados pela relação de inclusão estrita. Segundamente, nós avaliamos empiricamente o custo-benefício de doze dos critérios definidos e cinco técnicas de redução de suítes de testes. Nós avaliamos o tamanho, o tempo de execução e a detecção de faltas das suítes de testes reduzidas de cada uma das 60 combinações de critério e técnica. No experimento, nós usamos modelos de especificação, em TIOSTS, de uma máquina de recarga de cartão do metrô, de um sistema de alarme anti-roubo e de um limitador automático de velocidade de carros. Além disso, usamos simulações das implementações, que geram rastros corretos para os modelos. Por fim, o teste de mutação foi usado para gerar mutantes dos modelos de especificação, que, por sua vez, foram traduzidos para simulações com a finalidade de simular modelos de implementações defeituosas. As evidências empíricas sugerem que os critérios de adequação de testes mais próximos do topo da hierarquia produziram suítes reduzidas com melhor custo-benefício com relação à detecção de faltas e tempo de execução. Com relação às técnicas de redução, a técnica aleatória obteve melhor custo-benefício dentre as técnicas avaliadas. Os resultados apontam que os critérios explicam mais a variação nos resultados do que as técnicas.

Palavras-chave: Teste de Software. Teste Baseado em Modelos. Sistemas de Tempo-Real. Redução de Suítes de Testes. Critérios de Adequação de Testes. Timed Input-Output Symbolic Transition System.

Abstract

Model-based testing is a testing approach that relies on the existence of abstract models of an application to generate, execute and evaluate tests. Test case generation plays an important role in model-based testing. Since it consists of a systematic search for test cases that can be extracted from models, model-based testing usually generates large test suites which are too expensive to execute in full. Test suite reduction techniques have been proposed to address this problem. The goal of the techniques is to obtain reduced test suites that are both cheaper to execute and as effective at detecting faults as the original suite, given that the reduced test suites maintain the same coverage level of the complete test suite required by a test adequacy criterion. These criteria define which parts of the system are going to be tested, how often and under what circumstances. Nevertheless, little attention has been paid to the impact of the criterion choice in test suite reduction research. On the other hand, real-time systems are reactive systems whose behavior is constrained by time. Consequently, time-related faults are specific to these systems. In order to cope with this issue, models for real-time systems must deal with time and, consequently, there are specific test adequacy criteria for them. However, test suite reduction research has not focused on real-time systems, therefore the impact of test adequacy criteria for models of real-time systems on test suite reduction is unknown. In this doctoral research, we aim at investigating the influence of test adequacy criteria on the outcomes of test suite reduction techniques in the context of model-based testing of real-time systems. In particular, we are interested in the Timed Input-Output Symbolic Transition Systems (TIOSTS) model because it is an expressive transition system in which data and time are symbolically defined, and transition systems are the basis for conformance testing of real-time systems. In order to achieve the research objective, first, we defined 19 test adequacy criteria for TIOSTS models. The defined criteria include transition-based criteria, data-flow-oriented criteria and real-time systems criteria. Next, we formalized a hierarchy with these criteria which is partially ordered by the strict inclusion relation. Second, we evaluated the cost-effectiveness of twelve criteria and five test suite reduction techniques in empirical studies of test suite reduction. We evaluated the size, execution time and fault detection of reduced test suites obtained from each combination of criterion and technique. In the experiment, we used TIOSTS specification models of a refilling machine for charging the subway card, a burglar alarm system, and an automated car speed limiter; simulations of the implementations, which generate correct traces for the models; and mutation testing to generate mutants of the specification models, which were also translated to simulations in order to simulate faulty model implementations. Empirical evidence suggests that test adequacy criteria closer to the top of the family obtained reduced test suites with better cost-effectiveness regarding fault detection and execution time. With respect to the test suite reduction techniques, the Random technique obtained better cost-effectiveness among the evaluated criteria. Results also suggests that the criteria explain more the variation in fault detection and execution time of reduced test suites than the techniques.

Keywords: Software Testing. Model-Based Testing. Real-Time Systems. Test Suite Reduction. Test Adequacy Criteria. Timed Input-Output Symbolic Transition System.

Agradecimentos

A Deus porque sem Ele nada é possível. Agradeço por tudo o que Ele permitiu acontecer em minha vida, pelas oportunidades a mim oferecidas, por colocar as pessoas certas nos momentos certos no meu caminho, várias delas verdadeiros anjos-da-guarda.

A minha esposa Fabíola por estar sempre comigo ao longo de toda esta jornada. Com muita alegria e fé, superamos juntos as dificuldades e sobrepomos, um por um, os obstáculos que nos foram impostos. Nas oras mais difíceis, você era meu porto seguro. Eu sei que vários de nossos planos foram adiados por causa do meu doutorado e que sua renúncia foi maior que a minha, mas saiba que sem você eu não teria chegado até aqui e que esta tese simplesmente não teria acontecido, porque seu amor e apoio incondicional foram fundamentais para o começo, meio e fim deste trabalho. Esta tese é tão sua quanto minha! "Enquanto houver você do outro lado / Aqui do outro eu consigo me orientar". Eu te amo!

Aos meus pais, Seu Raimundinho e Dona Neidinha, pelo amor, educação e permanente incentivo aos estudos que me deram durante toda minha vida. Seus ensinamentos e exemplos me guiam em todos os meus passos. Eu tenho muito orgulho de ser seu filho. À minha irmã Amanda que também faz parte de quem eu sou e que sempre torceu pelo meu bem. Mesmo que a distância nos separe, vocês estão sempre comigo. Eu amo vocês!

Aos meus orientadores, Patrícia e Wilkerson, pela formação acadêmica e pessoal que recebi. É impressionante a forma como vocês conseguem ser extremamente rigorosos e minuciosos com os detalhes técnicos da pesquisa ao mesmo tempo em que demonstram uma sensibilidade extraordinária no trato pessoal. Era muito reconfortante saber que podia contar com vocês para qualquer coisa a qualquer hora, literalmente. Também gostaria de registrar que, nos momentos de incertezas e dúvidas, seus incentivos e demonstrações de entusiasmo foram essenciais para que eu seguisse em frente. Serei eternamente grato por tudo o que fizeram por mim!

À banca avaliadora, Adenilso Simão, Fátima Mattiello-Francisco, Jorge Figueiredo e Tiago Massoni, pelas valiosas contribuições a minha formação. Seus comentários e críticas pertinentes me fizeram refletir para tornar este trabalho melhor.

Aos amigos do SPLAB, Alysson, Ana, Adriana, Bruno, Dalton, Everton, João, Kláudio, Lilian, Marilene, Matheus, Paloma e Taciano, pela acolhida no laboratório, pelas colaborações em trabalhos e pelos momentos de descontração que partilhamos.

Ao Departamento de Informática da Universidade Federal da Paraíba pela licença que me permitiu fazer este trabalho com mais tranquilidade. Registro agradecimento especial aos amigos Hamilton e Gustavo pelo encorajamento e por tornarem esta oportuniade possível.

À Campina Grande, pela hospitalidade, pela comida boa, pelo Maior São João do Mundo, pelo clima frio gostoso, pelo reencontro com os velhos amigos Rodrigo e Fabrícia, pela nova amiga Elizete que a cidade me deu de presente, enfim pelas tantas lembranças boas do tempo feliz que passei na Rainha da Borborema.

Por fim, agradeço a todos que, embora não citados nominalmente, tenham contribuído direta ou indiretamente para a conclusão deste trabalho.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time systems are ubiquitous in our society — they are embedded in cars, banks, medical devices, TVs, video games, etc. A distinctive feature of a real-time system is that its behavior is constrained by time [47]. Correctness of a real-time system depends not just on the correctness of its outputs but also on the time at which these outputs are produced. Therefore, if a real-time system does not deliver outputs on time, then system performance is degraded or, in a critical system, a complete system failure is expected. Some failures may even lead to catastrophic consequences such as significant financial loss, injury, or loss of lives [43; 51; 52; 55].

Testing real-time systems is challenging, because it must detect time-related faults and this may require specific test cases to be designed and executed. This means that, in addition to checking for correct behavior, which is already a hard task, testing of these systems must also check for output conformance to the specified deadlines. Real-time systems are further tested for safety, reliability, security, etc.

Model-Based Testing is a testing approach that relies on the existence of abstract models of an application to generate, execute and evaluate tests [30; 60; 68; 69]. It has been applied with success in testing of real-time systems, with special emphasis on avionic, railway, and automotive domains [58]. In a recent survey with users of model-based testing, it was reported that model-based testing is as important as other kinds of test automation. For the users surveyed, model-based testing has had a positive effect on testing efficiency and effectiveness [17].

However, also according to the survey, model-based testing does not completely fulfill all testers' expectations. An expectation not fully fulfilled is cost reduction, because model-based testing usually generates large test suites, which are potentially expensive to manage and to execute [37]. Also with respect to the cost of model-based testing, there is no consensus in literature whether model-based testing is more cost-effective with respect to (1) time spent in executing and evaluating tests and (2) the number of faults detected than traditional testing techniques or other quality assurance techniques like reviews or inspections [59; 69].

Some approaches have been proposed to reduce the cost of executing large test suites. Promising techniques include test case prioritization [56], test case selection [21], and test suite reduction [27]. Test case prioritization techniques define an execution order of test cases according to a given testing goal, particularly, detecting failures as early as possible [64]. Test case selection techniques are focused on the identification of the modified parts of the program and on selection a subset of test cases that is relevant to the changed parts of the system, while test suite reduction techniques focus on selecting a subset of test cases that also satisfy the same test adequacy criterion as the original test suite [38]. In this research, we focus on the test suite reduction problem.

One factor that may affect the outcome of the test suite reduction process is the choice of the test adequacy criterion. A test adequacy criterion defines the requirements a test suite must satisfy [76]. For example, in a finite state machine, the ALL-STATES criterion demands that each state of the machine must be reached by at least one test case of the test suite [66].

Although both the size and fault detection capability of a reduced test suite may be affected by the criterion choice, little research has been carried on this subject [26]. Expressive symbolic models for real-time systems such as the Timed Input-Output Symbolic Transition System (TIOSTS) model [10; 11] allow time and data-flow-based criteria besides the transition-based ones. However, to the best of our knowledge, there is no research on the influence of test adequacy criteria for symbolic models of real-time systems on the outcomes of test suite reduction techniques.

1.1 Problem Statement

We illustrate the problem of not knowing the impact of test adequacy criterion choice on test suite reduction with an example. Consider the TIOSTS model of an authentication system of a building in Figure 1.1. The system operation is very simple. If the user enters the correct password, the system grants access to the building; otherwise access is denied. The user can try entering the password twice. In each attempt, the user must enter the password in at most 10 seconds, otherwise access is denied. The correct password is fixed during system operation.

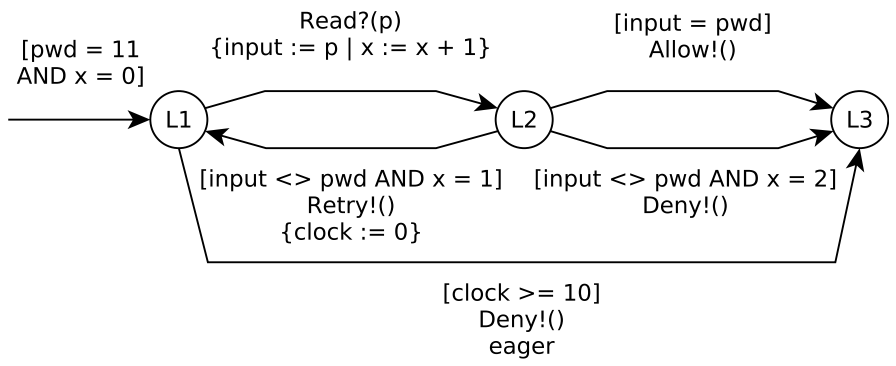


Figure 1.1: Authentication system.

There are five test cases for this system (Figure 1.2). In test case *A*, the user enters the correct password before the 10-second timeout, and the system allows access to the building. In test case *B*, the user enters a wrong password before the 10-second timeout, and the system outputs a message asking the user to retry entering the password and also resets the timeout clock. Next, the user enters the correct password before the 10-second timeout, and the system allows access to the building. In test case *C*, the user enters a wrong password before the 10-second timeout, and the systems outputs a message asking the user to retry entering the password and also resets the timeout clock. Next, the user enters again a wrong password before the 10-second timeout, and the system denies access to the building. In test case *D*, the user does not enter a password before the 10-second timeout, and the system denies access to the building. Finally in test case *E*, the user enters a wrong password before the 10-second timeout, and the systems outputs a message asking the user to retry entering the password and also resets the timeout clock. Next, the user does not enter a password before

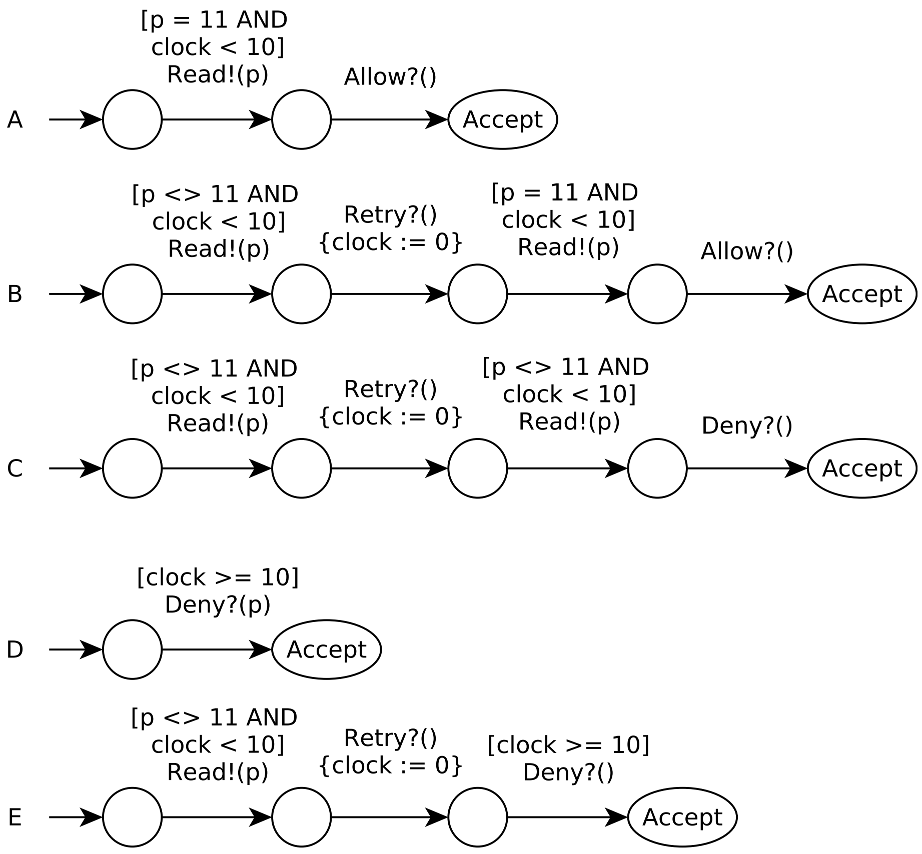


Figure 1.2: Test cases to authentication system.

the 10-second timeout, and the system denies access to the building.

For the sake of the example, let's consider that the tester cannot execute the complete test suite against a given implementation. Besides, since it is generally hard to program the notion of timeout after clock reset, let test case *E* be the only test case that fails in the given implementation. But the tester does not know that test case *E* fails because he has not executed any test case yet.

The tester must reduce the test suite but he is not sure which *Test Suite Reduction* (TSR) technique to use. Despite its simplicity, random algorithms have several successful real-world applications reported in the software testing literature [13]. The Random heuristic works by randomly selecting test cases from the complete test suite until all test requirements are covered. On the other side, the Greedy heuristic [25] generally produces good enough results in test suite reduction [22]. The Greedy heuristic works like the Random heuristic, but instead of randomly selecting a test case in each iteration, the greedy heuristic selects the test case that covers the highest number of uncovered test requirements.

Table 1.1: Criterion requirements covered by test cases.

Test Case	All-Locations	All-Clock-Guards
A	L1, L2, L3	—
B	L1, L2, L3	—
C	L1, L2, L3	—
D	L1, L3	CG
E	L1, L2, L3	CG

In addition to the TSR technique, the tester must also choose a test adequacy criterion [76] in order to define the coverage level of the reduced test suite. The tester has chosen the ALL-LOCATIONS[1] criterion [5], because it is simple to implement and he thinks that the choice of the TSR technique is much more important than the choice of the criterion. However, *is it true that test adequacy criteria do not impact test suite reduction outcomes?*

We analyze the aforementioned question by considering an additional criterion. We include the ALL-CLOCK-GUARDS criterion in the analysis because it is a real-time system criterion and it is in the same level in the family of TIOSTS criteria[2]. This criterion requires that every transition with a clock guard to be covered by a test suite [5]. The model in Figure 1.1 contains one clock guard (CG: `[c >= 10]`) in the transition from location L1 to location L3. Likewise, the model has three locations (L1, L2, and L3). Table 1.1 shows the requirements satisfied by each test case for each criterion.

Analysis. Table 1.2 shows the reduced test suites that can obtained by each technique for each criterion, and Table 1.3 shows whether a reduced test suite can reveal the failure, i.e. if the reduced test suite include the test case *E*.

For criterion ALL-LOCATION, the Greedy technique selects only one test case, while the Random technique selects 1.5 test cases on average. On the other hand, the odds of selecting a reduced test suite that reveals the failure is the same for both techniques: 25%. Thus, if the tester had chosen the Greedy technique, he would have done a good choice because this technique is more cost-effective than the Random technique as the former technique selects

[1]A location in a TIOSTS model is the equivalent of a state in a Finite State Machine.

[2]A family of criteria for TIOSTS models is discussed in Section 3.

Table 1.2: Reduced test suites that can be obtained by using each combination of technique and criterion.

Technique	All-Locations	All-Clock-Guards
Greedy	{A}, {B}, {C}, {E}	{D}, {E}
Random	{A}, {B}, {C}, {D, A}, {D, B}, {D, C}, {D, E}, {E}	{D}, {E}

Table 1.3: Failure detection of reduced test suites in Table 1.2.

Technique	All-Locations	All-Clock-Guards
Greedy	No, No, No, Yes	No, Yes
Random	No, No, No, No, No, No, Yes, Yes	No, Yes

less test cases than the latter while both have the same failure detection capability. However, by using the ALL-CLOCK-GUARDS criterion we get different results, because there is no difference neither in number of test cases selected nor in the odds of revealing the failure between the techniques. In fact, both techniques select the same reduced test suites.

Furthermore, we can see that differences in the results are more due to changing criteria than changing techniques. By changing the techniques, the odds of detecting the failure changes from 30% (7/10) in the Random technique to 33.33% (2/6) in the Greedy technique. But by changing the criteria, the odds changes from 25% (3/12) in ALL-LOCATIONS to 50% (2/4) in ALL-CLOCK-GUARDS. Similar results would be obtained for the number of test cases selected. This result suggests that test adequacy criteria may have more impact on test suite reduction outcomes.

Of course we cannot generalize results from a toy example, but the example is still thought provoking. The theoretical relationship among test adequacy criteria is important, but it may not tell much about its implications for practice. For example, its common knowledge that ALL-TRANSITIONS-adequate test suites are expected to be larger than ALL-LOCATIONS-adequate test suites. However, on average how large would ALL-TRANSITIONS-adequate test suites be in comparison to ALL-LOCATIONS-adequate test suites? Besides size, which criterion drives test suite reduction techniques to select test cases that detects more failures in less time? What should we expect when comparing two criteria that cannot be related by the strict inclusion relation [61] such as ALL-LOCATIONS and ALL-CLOCK-GUARDS? In the end, the main concern of the tester is which test adequacy criterion yields the best cost-effectiveness in test suite reduction?

Lack of empirical knowledge about the impact of test adequacy criteria on test suite reduction may cause the tester to choose a suboptimal criterion that may lead to loss of time, resources and money during software testing. However, cost-effectiveness of criteria for test suite reduction in the context of model-based testing is still under-investigated in literature, and hence little is known about it [19; 26]. Narrowing the context to model-based testing of real-time systems, specific criteria have been proposed but works are mostly theoretical [31]. Therefore, *rigorous empirical evaluation of the effect of the choice of the test adequacy criterion on the outcomes of test suite reduction for model-based testing of real-time systems is an open research problem.*

1.2 Research Questions and Methods

This doctoral research aims at evaluating the impact of test adequacy criteria on test suite reduction for model-based testing of real-time systems. We are particularly interested in the Timed Input-Output Symbolic Transition System (TIOSTS) model, a symbolic model to represent real-time systems. In this model, the system under test is represented by a transition system that interacts with its environment through input and output actions; variables are used to represent the system data, and a finite set of clocks are used to represent time evolution. This makes the TIOSTS a powerful and expressive model for real-time systems. Because of these characteristics, many criteria can be applied to TIOSTS models (with or without adaptation). For example, beyond criteria that covers time-related concepts, which are of special interest to real-time systems, the well known family of data-flow test adequacy criteria are applicable to TIOSTS models as well because the TIOSTS model handles data. That said, here are our research questions and methods:

Research Question 1 What is the theoretical relationship among TIOSTS test adequacy criteria?

The first step of this research is to identify test adequacy criteria that can be applied or adapted to TIOSTS models. The research method to answer this question is a literature review. Both criteria for source code and criteria for other models must be considered in the search. Each criterion identified must be analyzed for applicability to the TIOSTS model, and the applicable ones must be formalized.

The second step of this research is to establish the theoretical relationship between the TIOSTS test adequacy criteria. The standard method in literature to theoretically compare two criteria is the strict inclusion relation [61]. To answer this question, we must analyze each pair of criteria and produce a proof for the pairs in which there is strict inclusion between them.

Research Question 2 What is the influence of test adequacy criteria for TIOSTS models on test suite reduction for model-based testing of real-time systems?

The third and main step of this research is to empirically evaluate the impact of TIOSTS criteria on test suite reduction. Besides theoretical results, empirical comparison among test suite reduction techniques in combination with different criteria is necessary to understand the cost and effectiveness of reduced test suites in order to provide guidance to testers during the testing process. Research method to answer this question is an experiment.

1.3 Contributions

In our doctoral work we achieved the following contributions:

1. The definition of 19 test adequacy criteria for symbolic models of real-time systems;
2. The formalization of the theoretical relationship among the defined criteria [5; 54];
3. Empirical assessment of the influence of test adequacy criteria on test suite reduction for model-based testing of real-time systems.

In summary, we evaluated —from theory to practice— test adequacy criteria for TIOSTS models in the context of test suite reduction. Empirical evidence of the experiment suggests that test adequacy criteria have different cost-effectiveness as test suite reduction techniques also do, however the influence of criteria is larger than the influence of techniques in test suite reduction outcomes.

1.4 Document Organization

The rest of the document is organized as follows. Chapter 2 introduces the concepts of test adequacy criteria and TIOSTS model. Chapter 2.3 defines the test suite reduction problem, and reviews main heuristics and empirical works on test suite reduction for model-based testing. Chapter 3 presents a family of 19 test adequacy criteria, partially ordered by strict inclusion, for Timed Input-Output Symbolic Transition System models. Chapter 4 presents one experiment that assess the influence of test adequacy criteria for TIOSTS models on test suite reduction for model-based testing of real-time systems. Chapter 5 contains the final remarks and points out future works.

Chapter 2

Background

This chapter presents some basic concepts required to make this document self-contained. We define test adequacy criterion and why it is important to model-based testing and test suite reduction and then we introduce the Timed Input-Output Symbolic Transition System Model (TIOSTS) model which is the model used in this research. Next, we define the test suite reduction problem and the main heuristics for the problem. Finally, we present a brief review of relevant empirical work for test suite reduction in the context model-based testing.

2.1 Test Adequacy Criteria

According to Goodenough and Gerhart [35, p. 156], a software test data adequacy criterion (also known as test coverage criterion or test selection criterion) is a predicate that defines "what properties of a program must be exercised to constitute a 'thorough' test, i.e., one whose successful execution implies no errors in a tested program". Adequacy criteria are essential to software testing, because they specify the test requirements, and hence determines what test cases are needed to satisfy the requirements [76].

Mathematically speaking, test data adequacy criteria can be seen as either *generators*, i.e. functions that produce a class of test suites from the program under test and the specification such that any test suite in this class is adequate, or *acceptors*, i.e. functions from the program under test, the specification of the software and the test suite to a boolean value indicating whether the test suite is adequate. Generators and acceptors are equivalent in the sense of one-to-one correspondence, and hence "test adequacy criteria" can be used to denote both of them [76].

Formally, let P be a set of programs, S be a set of specifications, D be the set of inputs of the programs in P, T be the class of test suites, i.e. $T = 2^D$, where 2^X denotes the set of subsets of X.

Definition 1 (Test Data Adequacy Criteria as Generators)**.** *A test data adequacy criterion C is a function $C\colon P \times S \rightarrow 2^T$. A test set $t \in C(p, s)$ means that t satisfies C with respect to p and s, and it is said that t is adequate for (p, s) according to C [76].*

Definition 2 (Test Data Adequacy Criteria as Acceptors)**.** *A test data adequacy criterion C is a function $C\colon P \times S \times T \rightarrow \{true, false\}$. $C(p, s, t) = true$ means that t is adequate for testing program p against specification s according to the criterion C, otherwise t is inadequate [76].* ◇

There are various classifications of adequacy criteria. Zhu, Hall and May [76] classify criteria as either specification-based or program-based, depending on the source of information used to specify testing requirements. They also cite another classification: structural testing, fault-based testing, and error-based testing, which is based on the underlying testing approach. On the other hand, Ammann and Offut [7] classify criteria in a very abstract way: graph coverage, logic coverage, input space partitioning, and syntax-based testing. In each category, first criteria are defined in a generic model and then the authors demonstrate how that criteria can be applied to programs and models. These classifications apply to criteria for models as well criteria for programs. Conversely, Utting and Legeard [69] define a classification which is specific for model-based testing: structural model coverage criteria, data coverage criteria, fault-model criteria, requirements-based criteria, explicit test case specifications, and statistical test generation methods.

Test adequacy criteria play an important role in test case generation within model-based testing because criteria control the test case generation process [69]. In this work, we investigate their role in test suite reduction.

2.2 TIOSTS Model

Timed Input-Output Symbolic Transition System (TIOSTS) [10; 11] is a symbolic model for real-time systems that handles both data and time. The TIOSTS model was defined as an extension of two existing models: Timed Automata [6] and Input-Output Symbolic Transition Systems [41; 65]. Basically, a TIOSTS is an automaton with a finite set of locations where system data and time evolution are respectively represented by variables and a finite set of clocks. The transitions of the model are composed of a guard on variables and clocks, an action with parameters, an assignment of variables, and a set of clocks to reset.

Figure 2.1 shows an example of TIOSTS that models a machine for refilling a card for using the subway (SUB)[1]. Initially, the system is in the `Idle` location where it expects the `Credit` input carrying the desired `value` to refill, then this value is saved into the `refillValue` variable[2] and `balance` is initialized to zero.

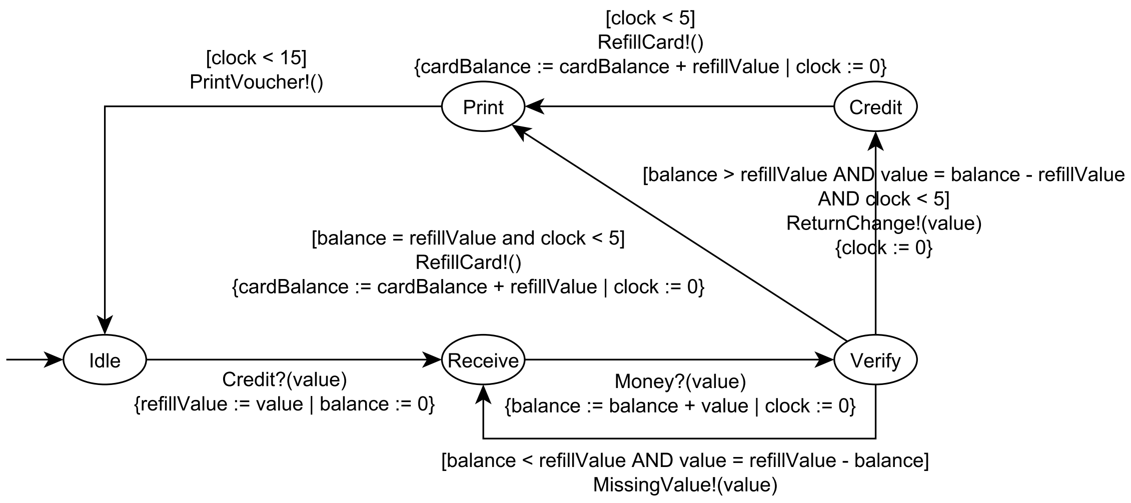

[clock < 5]
RefillCard!()
{cardBalance := cardBalance + refillValue | clock := 0}
[clock < 15]
PrintVoucher!()
Print
Credit
[balance > refillValue AND value = balance - refillValue
AND clock < 5]
ReturnChange!(value)
{clock := 0}
[balance = refillValue and clock < 5]
RefillCard!()
{cardBalance := cardBalance + refillValue | clock := 0}
Idle
Receive
Verify
Credit?(value)
{refillValue := value | balance := 0}
Money?(value)
{balance := balance + value | clock := 0}
[balance < refillValue AND value = refillValue - balance]
MissingValue!(value)

Figure 2.1: TIOSTS model of a refilling machine.

From the `Receive` location to `Verify`, the client informs the amount to be credited to the card. This value is accumulated in the `balance` variable and the `clock` is set to zero. If the current balance is less than the desired value to refill, then the `Receive` location is reached again and the `MissingValue` output is emitted for informing the remaining value (the condition $\texttt{value} = \texttt{refillValue} - \texttt{balance}$ contained in the guard means "choose a value for the `value` parameter that, with the values of `refillValue`

[1]In graphical representations of TIOSTS models, the pipe symbol is used to separate multiple assignments and clock resets in the same transition. Also, multiple assignments are executed from left to right.

[2]Action parameters have local scope, thus their values must be stored in variables for future references.

and `balance` variables, satisfies the guard").

From the `Verify` location, if the `balance` is greater than `refillValue` some value must be returned to the client in less than 5 time units. After that, the `clock` is reset to zero again. Then, the `RefillCard` output action must be performed in less than 5 time units and the `cardBalance` is increased by `refillValue`. Otherwise, from `Verify`, if `balance` is exactly equals to `refillValue`, then the card must be refilled in less than 5 time units. Finally, from the `Print` location, the voucher must be printed in less than 15 time units and `Idle` location is reached again. The formal definition of TIOSTS models is presented in Definition 3 [10].

Definition 3 (TIOSTS)**.** *A TIOSTS is a tuple $W = \langle V, P, \Theta, L, l^0, \Sigma, C, \mathcal{T}\rangle$, where:*

- *V is a finite set of typed variables, and P is a finite set of parameters. For $x \in V \cup P$, $type(x)$ denotes the type of x;*
- *Θ is the initial condition, a predicate with variables in V;*
- *L is a finite, non-empty set of locations and $l^0 \in L$ is the initial location;*
- *$\Sigma = \Sigma^? \cup \Sigma^!$ is a non-empty, finite alphabet, which is the disjoint union of a set $\Sigma^?$ of input actions and a set $\Sigma^!$ of output actions. For each action $a \in \Sigma$, its signature $sig(a) = \langle p_1, ..., p_n\rangle$ is a tuple of distinct parameters, where each $p_i \in P$ ($i = 1, ..., n$);*
- *C is a finite set of clocks with values in the set of non-negative real numbers, denoted by $\mathbb{R}^{\geq 0}$;*
- *$\mathcal{T}$ is a finite set of transitions. Each transition $t \in \mathcal{T}$ is a tuple $\langle l, a, G, A, y, l'\rangle$, where:*
  - *$l \in L$ is the origin location of the transition;*
  - *$a \in \Sigma$ is the action;*
  - *$G = G^D \wedge G^C$ is the guard. G^D is a predicate over variables in $V \cup set(sig(a))$[3], and is assumed to be expressed in a theory in which satisfiability is decidable. G^C is a predicate over clocks in C defined as a conjunction of constraints of the form $\alpha \# c$, where $\alpha \in C$, $\# \in \{<, \leq, =, \geq, >\}$, and $c \in \mathbb{N}$;*

[3] $set(j)$ is the function that converts the tuple j in a set.

- *$A = \langle A^D, A^C \rangle$ is the assignment of the transition. A^D is a sequence of assignments such that for each variable $x \in V$ there is exactly one assignment in A^D of the form $x := A^{D^x}$, where A^{D^x} is an expression on $V \cup set(sig(a))$. $A^C \subseteq C$ is the set of clocks to be reset;*
- *$y \in \{$lazy, delayable, eager$\}$ is the deadline of the transition;*
- *$l' \in L$ is the destination location of the transition.*

The locations and transitions of the TIOSTS W form a connected graph. $\diamond$

The *lazy* deadline imposes no urgency to the transition to be taken, *delayable* means that once enabled the transition must be taken before it becomes disabled, and *eager* means the transition must be taken as soon as it becomes enabled. In graphical representations, when not specified, the deadline of transitions with output actions is assumed to be delayable and the deadline of transitions with input actions is assumed to be lazy. Finally, it should be noted that terminal locations are not mandatory for TIOSTS models. In fact, since most real-time systems run continually, TIOSTS models are usually modeled as a big loop and possibly some inner loops.

Semantics of TIOSTS. Timed Input-Output Labeled Transition Systems (TIOLTS) [45] can be used to describe the semantics of a TIOSTS $\langle V, P, \Theta, L, l^0, \Sigma, C, \mathcal{T} \rangle$ as follows. Basically, the TIOLTS states represent the sets of locations, of *valuations* of variables V and clocks C, while transitions represent the sets of actions Σ associated with parameters values P. A *valuation* of the variables in V is a mapping ν which maps every variable $x \in V$ to a value $\nu(x)$ in the domain of x. Valuations of parameters P are defined similarly for γ, Let $\mathcal{V}$ denote the set of valuations of the variables V, and let Γ denote the set of valuations of the parameters P. The set of valuations of all parameters of an action a is denoted by $\Gamma_{sig(a)}$. Valuation of the clocks in C is a mapping ψ which maps every clock $c \in C$ to a value $\psi(c)$ in the domain of $\mathbb{R}^{\geq 0}$. Let Ψ denote the set of valuation of the clocks C. We note $\overline{0}$ the clock valuation that assigns 0 to all clocks and $\psi' = \psi[A^C \leftarrow 0]$ the new valuation obtained by resetting to 0 all clocks in A^C and leaving the others unchanged. Considering $\nu \in \mathcal{V}$ and $\gamma \in \Gamma$, we denote $E^{A^D}(\nu, \gamma)$ the new variable valuation ν' obtained by evaluating the result of substituting in A^D each variable by its value according to ν and each parameter by its value according to γ.

Definition 4 (TIOLTS semantics of a TIOSTS). *The semantics of a TIOSTS $W = \langle V, P, \Theta, L, l^0, \Sigma, C, \mathcal{T}\rangle$ is a TIOLTS $[\![W]\!] = \langle S, S^0, Act, T\rangle$, defined as follows:*

- *$S = L \times \mathcal{V} \times (C \to \mathbb{R}^{\geq 0})$ is the set of states of the form $s = \langle l, \nu, \psi\rangle$ where $l \in L$ is a location, $\nu \in \mathcal{V}$ is a specific valuation for all variables V, and $\psi \in \Psi$ is a clock valuation;*

- *$S^0 = \{\langle l^0, \nu, \psi\rangle \mid \Theta(\nu) = true, \psi = \overline{0}\}$ is the set of initial states;*

- *$Act = \Lambda \cup D$ is the set of actions, where $\Lambda = \{\langle a, \gamma\rangle \mid a \in \Sigma, \gamma \in \Gamma_{sig(a)}\}$ is the set of discrete actions and $D = \mathbb{R}^{\geq 0}$ is the set of time-elapsing actions. Λ is partitioned into the sets $\Lambda^?$ of input actions and $\Lambda^!$ of output actions;*

- *T is the transition relation defined as the least set of the following rules:*

  Discrete actions:

$$\frac{\begin{array}{ll} \langle l, \nu, \psi\rangle, \langle l', \nu', \psi'\rangle \in S & \langle a, \gamma\rangle \in \Lambda \\ t : \langle l, a, G, A, y, l'\rangle \in \mathcal{T} & G = true \\ \nu' = E^{A^D}(\nu, \gamma) & \psi' = \psi[A^C \leftarrow 0] \end{array}}{\langle l, \nu, \psi\rangle \xrightarrow{\langle a,\gamma\rangle} \langle l', \nu', \psi'\rangle}$$

  Time-elapsing actions:

$$\frac{\begin{array}{l} d \in D \qquad \langle l, \nu, \psi\rangle, \langle l, \nu, \psi + d\rangle \in S \\ t : \langle l, a, G, A, y, l'\rangle \in \mathcal{T} \\ y = \text{eager} \implies \psi \not\models G^C \\ y = \text{delayable} \implies \nexists d_1, d_2 \in D : \\ \qquad (0 \leq d_1 < d_2 \leq d) \wedge (\psi + d_1 \models G^C) \wedge (\psi + d_2 \not\models G^C) \end{array}}{\langle l, \nu, \psi\rangle \xrightarrow{d} \langle l, \nu, \psi + d\rangle}$$

$\diamond$

Remark 1. *Once the lazy deadline is used only to denote the absence of deadlines, it does not impact the semantics. As in [45], delayable transitions with guards of the form $\alpha < c$ are not allowed because there is no latest time so that the guard is still true. Also, eager* *transitions with guards of the form $\alpha > c$ are not allowed because there is no earliest time so that the guard becomes true.* ◇

Important definitions and properties of TIOSTS are defined in terms of their underlying TIOLTS semantics (Definition 4) as follows. Let $W = \langle V, P, \Theta, L, l^0, \Sigma, C, \mathcal{T} \rangle$ be a TIOSTS whose semantics is defined by the TIOLTS $[\![W]\!] = \langle S, S^0, \mathit{Act}, T \rangle$. Let $s, s', s_i \in S$ be some states, $a, a_i \in \mathit{Act}$ some actions, and Act^* the set of all finite sequences of discrete actions and time-elapsing actions. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in T$. Given $(a_1, \cdots, a_n) \in \mathit{Act}^*$, an *execution* is defined as $s \xrightarrow{a_1 \cdots a_n} s' \triangleq \exists s_0, ..., s_n : s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = s'$. For $\sigma \in \mathit{Act}^*$, we also define $s \xrightarrow{\sigma} \triangleq \exists s' : s \xrightarrow{\sigma} s'$. The set of sequences of discrete and time-elapsing actions fireable from s is defined by $Traces(s) \triangleq \{\sigma \in \mathit{Act}^* : s \xrightarrow{\sigma}\}$. The set of sequences of behaviors fireable from the initial state of a TIOSTS W is defined by $Traces(W) \triangleq \bigcup_{s \in S_0} Traces(s)$. For $\sigma \in \mathit{Act}^*$, the set s *after* $\sigma \triangleq \{s' \in S \mid s \xrightarrow{\sigma} s'\}$ is the set of states reachable from s after the execution of σ, and P *after* $\sigma \triangleq \bigcup_{s \in P} s$ *after* σ is the set of states reachable from the set P after the execution of σ.

Finally, W is said to be *deterministic* if these conditions are satisfied [41; 65]: (1) $\mid S_0 \mid = 1$; (2) for all $l \in L$ and for each pair of distinct transitions $t_1 = \langle l, a, G_1, A_1, y_1, l'_1 \rangle$ and $t_2 = \langle l, a, G_2, A_2, y_2, l'_2 \rangle$ with origin in l carrying the same action a, the guards G_1 and G_2 are mutually exclusive, i.e. $G_1 \wedge G_2$ is unsatisfiable.

Definition 5 (Test Case)**.** *A test case is a deterministic TIOSTS $tc = \langle V_{tc}, P_{tc}, \Theta_{tc}, L_{tc}, l^0_{tc}, \Sigma_{tc}, C_{tc}, \mathcal{T}_{tc} \rangle$, where $\Sigma^?_{tc} = \Sigma^!_W$ and $\Sigma^!_{tc} = \Sigma^?_W$ (the actions Σ_{tc} of the test case tc are mirrored with respect to the actions Σ_W of the specification model W), equipped with the disjoint set of verdict locations $\{Pass, Fail, Inconclusive\}$.* ◇

According to Definition 5, an execution of a symbolic test case emits one of three possible verdicts: *Pass*, *Fail*, and *Inconclusive*. *Pass* means that some targeted behavior of the system under test has been reached, *Fail* means the rejection of the system under test, and *Inconclusive* means that the targeted behavior cannot be reached anymore.

Figure 2.2 is a test case for the TIOSTS model of the refilling machine. The test case aims to exercise the scenario where the system emits the `RefillCard` output when the amount to be credited to the card (`value_2`) is equal to desired value to refill (`value_1`). In this case, the verdict is *Pass*. If the amount to be credited to the card (`value_2`) is

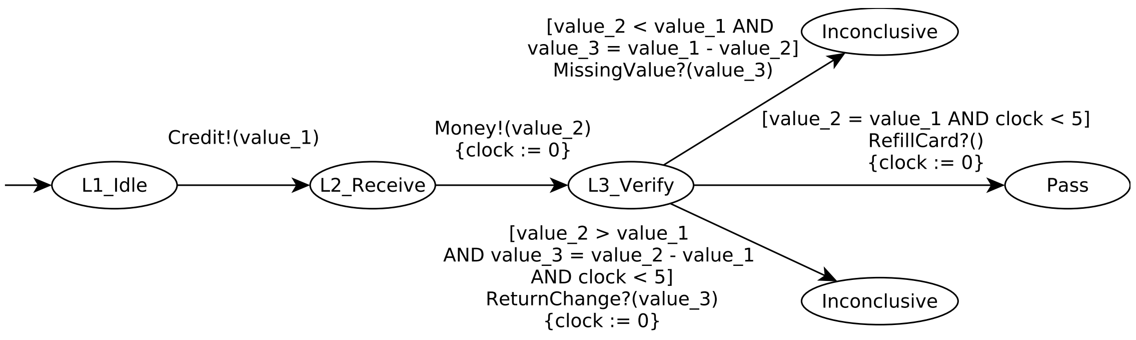


Figure 2.2: A test case for the refilling machine.

less than the desired value to refill (`value_1`), and the system emits the `MissingValue` output with parameter equals to `value_1 − value_2`, then the verdict is *Inconclusive*. It is *Inconclusive* because this behavior is specified in the model, but it is not the scenario the tester would like to observe in the test case execution. The same applies to `ReturnChange` output action of the test case. All other cases lead to the implicit *Fail* verdict.

2.3 Test Suite Reduction

The Test Suite Reduction (TSR) problem is classically defined as [38, p. 273]:

Definition 6 (Test Suite Reduction Problem)**.** Given*: Test suite T, a set of test case requirements $r_1, r_2, \dots, r_n$ that must be satisfied to provide the desired test coverage of the program, and subsets of T, $T_1, T_2, \dots, T_n$, one associated with each of the r_i's such that any one of the test cases t_j belonging to T_i can be used to test r_i.* Problem*: Find a representative set of test cases from T that satisfies all r_i's.*

The r_i's can mapped to any meaningful requirements for a given context, e.g. they could be source code statements or model transitions. A representative set of test cases that satisfies all r_i's must contain at least one test case from each T_i. In other words, a representative set of test cases $T' \subseteq T$ must satisfy the predicate $\forall r(r \in R)(r \cap T' \neq \emptyset)$, where $R = \{T_1, T_2, \dots, T_n\}$. Such set is called a *Hitting Set* of the set of sets $T_1, T_2, \dots, T_n$ [63]. Maximum test suite reduction is achieved when the smallest representative set of test cases is found. This is the same as finding the minimum cardinality hitting set of the T_i's, which is a NP-hard problem [34]. Thus, most TSR techniques in literature are approximations that do not guarantee minimal reduced test suites with respect to the number of test cases [74]. For this reason, we use the more general term *test suite reduction* instead of *test suite minimization* to describe such techniques.

Another (and more intuitive) formulation of the TSR problem is to define it as a *Set Covering* problem (SCP) [25]:

Definition 7 (Test Suite Reduction Problem)**.** Given*: A set of test requirements R, and a set T of subsets of R whose union of the elements of T is equal to R, i.e. $\bigcup_{t \in T} t = R$. The set T represents a test suite, and each element of T is associated to one test case and the test requirements satisfied by the test case.* Problem*: Find the smallest $T' \subseteq T$ whose union of the elements of T' is equal to R.*

Many algorithms are proposed in literature for the SCP problem [20; 32; 46] — from the greedy heuristic [23] that efficiently finds near-optimal solutions to exact algorithms [16] that, although computationally expensive, guarantee finding the smallest T'. However, the challenge in TSR is to obtain a reduced test suite that maintains the fault detection capability of the complete test suite without upfront knowledge of the fault detection of each test case. Otherwise if fault detection of test cases is known in advance, the TSR problem would be simply a direct application of SCP from test cases to faults (instead of test requirements).

It should be noted that there are basically two opportunities to reduce a test suite: during or after test case generation. We name *a priori test suite reduction* the techniques that reduce test suite size by avoiding the generation of test cases. Those techniques are test case generation algorithms that use test adequacy criteria as generators (Definition 1) like in model-based testing [4; 69; 75]. On the other hand, we call *a posteriori test suite reduction* the techniques that select a subset of generated test suites as defined in Definition 7. In this case, it does not matter how test suites are generated, e.g. by random [8] or search-based testing [14]. These techniques use test adequacy criteria as acceptors (Definition 2), and they are the focus of this research.

2.4 Heuristics for Test Suite Reduction

In order to deal with computationally expensive execution of exact algorithms, many heuristics for TSR were proposed in literature. A recent and comprehensive survey of regressing testing by Yoo and Harman includes TSR techniques [74]. However, it is out of scope of this doctoral research surveying all of them. Instead, we give an overview of classic TSR heuristics.

The greedy heuristic G [22; 23] works by moving one test case at a time from the complete test suite and to the reduced test suite[4]. In each iteration, the selected test case covers the highest number of test requirements not covered yet by the reduced test suite. If there is a tie among test cases, an arbitrary choice is made, although a second and stronger test adequacy criterion could be used as tiebreaker [48]. The heuristic keeps selecting test cases until all test requirements are covered by the reduced test suite.

Chen and Lau [22] propose two variations of G heuristic. The GE heuristic depends on the concept of *essential test case*. A test case t is essential if t is the only test case that covers some requirement. The heuristic moves all essential test cases to the reduced test suite, and then applies the G heuristic to cover the remaining uncovered test requirements. The GRE heuristic depends on the concepts of (1) essential test case and (2) *1-to-1 test case redundancy*. A test case t_1 is 1-to-1 redundant if there is another test case t_2 that covers all test requirements covered by t_1. The heuristic alternatively tries to remove one 1-to-1 redundant test case from the complete test suite, and then tries to move one essential test case to the reduced test suite. When there is no more essential or 1-to-1 redundant test case, the heuristic applies the G heuristic to cover the remaining uncovered test requirements.

Harrold, Gupta and Soffa [38] propose the heuristic known as HGS. In each iteration, the set of uncovered test requirements with the smallest cardinality is selected. The cardinality of a test requirement is the number of test cases that cover it. Then, the test case that covers the maximum number of uncovered test requirements of that set is moved to the reduced test suite. If there is a tie among test cases, the set of uncovered requirements with the

[4]Technically speaking, the test case is not really moved from the complete test suite to the reduced test suite. Moving a test case means it was selected for inclusion in the reduced test suite, and it should not be considered again in future iterations of the heuristic.

second-smallest cardinality is selected, and the test case involved in the tie which covers the maximum number of elements of this second set is moved. In case of a new tie, the third-smallest set is used, and so on. Finally, if there is still a tie, a random choice is made. The heuristic keeps selecting one test requirement at a time until all of them are covered by the reduced test suite.

The SCP problem can also be expressed as a binary Integer Linear Programming (ILP) problem, and so the TSR problem [18]. Mathematical programming packages such as CPLEX[5] can compute exact and optimal solutions even for large-sized problems. In ILP model 2.1, which models the SCP problem (Definition 7), test case j is represented by decision variable x_j, whose value is 1 if test case t_j is included in the reduced test suite, otherwise its value is 0. The constraint matrix A has $|R|$ rows, one for each test requirement of the test adequacy criterion, and $|T|$ columns, one for each test case. And element a_{ij} of A is 1 if test case j covers the test requirement i, otherwise its value is 0.

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{|T|} x_j \\ \text{Subject to} \quad & \sum_{j=1}^{|T|} a_{ij} \cdot x_i \geq 1, i = 1, \ldots, |R| \\ & x_i \text{ binary for } j = 1, \ldots, |T| \end{aligned} \tag{2.1}$$

The objective function of the ILP model can be extended to consider additional constraints. Black, Melachrinoudis and Kaeli [18] added a secondary constraint to the objective function, which is to maximize the error detection rate by considering the known ability of individual test cases to reveal failures. To encode this constraint, a binary vector e of cardinality T is introduced. For each test case j in T, $e_j = 1$ if test case t_j is known to reveal an error in the subject program, otherwise its value is 0. A weighting factor ($0 \leq \alpha \leq 1$) is introduced in order to allow one goal to take precedence over the other. The final formulation

[5]IBM ILOG CPLEX Optimizer: `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`

of the bi-objective ILP model is:

$$
\begin{aligned}
\text{Minimize} \quad & \alpha \sum_{j=1}^{|T|} x_j - (1-\alpha) \sum_{j=1}^{|T|} e_j \cdot x_j \\
\text{Subject to} \quad & \sum_{j=1}^{|T|} a_{ij} \cdot x_i \geq 1, i = 1, \ldots, |R| \\
& x_i \text{ binary for } j = 1, \ldots, |T|
\end{aligned} \tag{2.2}
$$

Beyond these heuristics, other techniques based on different approaches have been proposed, for example genetic by using algorithms [57], similarity among test cases [26], and exact methods [36].

2.5 Empirical Studies on Test Suite Reduction for Model-Based Testing

In this section, we present a review of relevant empirical work on test suite reduction for model-based testing of real-time systems. As research related to model-based test suite reduction is very recent, there are few works aligned with that. To the best of our knowledge, works on the influence of test adequacy criteria for models of real-time system on test suite reduction are practically nonexistent [5]. Thus, the focus here is to describe empirical works on test suite reduction for model-based testing which compare cost-effectiveness of criteria.

Heimdahl and George [39] evaluate the cost-effectiveness of six test adequacy criteria for the RSML^{-e} model. The evaluated criteria were: variable domain coverage, transition coverage, decision coverage, decision coverage with single uses, modified condition and decision coverage, modified condition and decision coverage with single uses. A model of an industrial Flight Guidance System was used in the experiment. For each criterion, an adequate test suite with respect to the criterion was generated by using a naïve test generation algorithm, which covers one test requirement at a time and hence the resulting test suite has lots of redundancy, a reduced test suite were obtained by using the random technique.

Faulty versions of the model were used to evaluate the fault detection capability of reduced test suited in comparison with the complete test suite. Results show that the size of the specification based test-suites can be dramatically reduced but the fault detection of the reduced test-suites is adversely affected. Moreover, more rigorous criteria, such as Modified Condition and Decision Coverage, provide a better fault finding capability both for the complete test suites as well as the reduced test suites as compared to less rigorous criteria, such as variable domain and transition coverage.

Briand, Labiche and Wang [19] conducted a similar experiment to assess cost-effectiveness of criteria for UML statechart machines. The main difference is that they randomly generated one test suite that covered all test adequacy criteria instead of one test suite for each criterion, and then they used the greedy heuristic to reduce the generated test suite. Test cases were generated against the models of a data structure, a cruise control, and a video recorder software (all of them academic projects) to cover the criteria all transitions (AT), all transition pairs (ATP), all paths in transition trees (TT), and full predicate (FP). Faults were seeded into implementations to assess fault detection of reduced test suites. Their results show that criterion AT is not sufficient to ensure adequate level of fault detection. On the other hand, the ATP criterion offers a very strong guarantee (though no certainty) that (nearly) all faults will be detected, but it costs much more than AT. Criterion TT is not always cost-effective in the context of class (cluster) testing. Finally, under the same circumstances, FP does not appear very cost-effective as it is more expensive than ATP but it has the same effectiveness.

Coutinho, Cartaxo and Machado [26] present an evaluation of five test suite reduction heuristics in conjunction with three test adequacy criteria for the LTS model. They proposed a new test suite reduction technique (Sim) based on similarity among test cases, and compare it against the heuristics G, GE, GRE and HGS described in the previous section. The evaluated criteria were all transitions, all transition pairs, and the combination of all transitions + all transition pairs. For each model, one test suite that satisfies the evaluated criteria was generated by using a depth-first search algorithm. They used three real models in the experiment (a biometrics processing software, a desktop PDF processing application, and a software testing tool), and 90 synthetic models which were generated based on the characteristics of these three models. Defects found during testing of the three software were used to evaluate the fault detection capability of reduced test suites. And for the synthetic models, failure profiles were also generated. They investigated which criterion is more cost-effective for each reduction strategy, and which reduction strategy is more effective is more cost-effective for each criterion. Experimental results show that the choice of coverage criteria can influence test suite size reduction and fault coverage. Among all combinations of heuristics and criteria, the combination of the Sim heuristic with the all-transitions + all-transition-pairs criterion significantly increase the fault coverage rate without significant increase on the size of the reduced test suite.

Other studies also evaluated the role of criteria in test suite reduction for model-based testing, but effectiveness at detecting faults was not assessed [44; 66]. Finally, some researchers propose reducing test suites by merging test cases instead of selecting some of them [24; 33].

2.6 Concluding Remarks

In this chapter, we introduced the concept of test adequacy criterion, which is a key concept to both model-based testing and test suite reduction. Next, we presented the TIOSTS model which we use in this work. We also formalized the Test Suite Reduction Problem and presented heuristics for the TSR problem. Finally, we reviewed empirical works on TSR for model-based testing.

We remark that previous empirical studies conducted on TSR techniques on source code have shown that techniques can significantly reduce the number of test cases in a test suite [62; 63; 72; 73]. However, the effect on the fault detection capability of reduced test suites is unclear because of conflicting evidence. Wong et al. [72; 73] found no significant effect in fault finding ability between the full suites and the reduced suites, while Rothermel et al. [62; 63] and Jones and Harrold [42] showed that the reduced test-suites can be dramatically worse with respect to fault finding.

On the other hand, evidence point out significant reduction in both size and fault detection of reduced test suites in TSR for model-based testing. However, empirical studies did not assess the effect of TSR techniques combined with different test adequacy criteria from a global perspective. Moreover, the influence of criteria for models of real-time systems on TSR outcomes is an open problem.

Chapter 3

A Family of Test Adequacy Criteria for TIOSTS models

In this chapter, we propose a family of 19 test adequacy criteria, partially ordered by strict inclusion, for Timed Input-Output Symbolic Transition System models.

En-Nouaary [31] proposes a family of test adequacy criteria partially ordered by strict inclusion for Timed Input-Output Automata (TIOA). His family combines transition-based criteria, reactive systems criteria, and real-time systems criteria, but data-related criteria are not included because the TIOA model does not support data abstraction. Here, we extend En-Nouaary's family [31] to include data-related criteria because the TIOSTS model supports data variables. Thus, our proposed family of criteria combines transition-based criteria, reactive systems criteria, real-time systems criteria and data-flow-oriented criteria. Almost all criteria are described elsewhere [7; 31; 40; 69], however the level of rigor of definitions and the underlying models vary. In this work, we unify the criteria within a common formal framework and define them in terms of requirements a test suite must satisfy [7].

3.1 Preliminaries

The following definitions assume a specification $W = \langle V, P, \Theta, L, l^0, \Sigma, C, \mathcal{T} \rangle$, the semantics of W defined by the TIOLTS $[\![W]\!] = \langle S, S^0, Act, T \rangle$, and a test case $tc = \langle V_{tc}, P_{tc}, \Theta_{tc}, L_{tc}, l^0_{tc}, \Sigma_{tc}, C_{tc}, \mathcal{T}_{tc} \rangle$.

Given $t_1, \cdots, t_n \in \mathcal{T}, n \geq 1$, let $p \triangleq (t_1, ..., t_n)$ be a non-empty sequence of transitions from $\mathcal{T}$ of the model W, in which element repetitions are allowed in the sequence. The length of the sequence is denoted by $|p|$. The first element of p is denoted by p_1, the last element by $p_{|p|}$, and the i^{th} element by p_i. Let $\mathcal{T}^*$ be the set of all finite sequences that can be composed from the set of transition $\mathcal{T}$.

Definition 8 (Path). *A path is a sequence of transitions such that the destination location of the i^{th} transition is equal to the origin location of the $(i+1)^{th}$ transition. The function $ispath(p)$ returns whether p is a path, such that $ispath\colon \mathcal{T}^* \to \{true, false\}$ is given by $ispath(p) \mapsto (|p| \geq 1) \wedge \forall i[(1 \leq i < |p|) \implies (l'_{p_i} = l_{p_{i+1}})]$, where $p \in \mathcal{T}^*$, l'_{p_i} is the destination location of the i^{th} transition of the list p, $l_{p_{i+1}}$ is the origin location of the $(i+1)^{th}$ transition of p, and $i \in \mathbb{N}$. The empty list is not a path by definition.* $\diamond$

A path is a requirement of a criterion only if the path is feasible and reachable, otherwise the criterion would never be satisfied. A path p is *feasible* if there is at least one state which contains the initial location of the path[1] such that the state makes possible the guard of each transition of p to be satisfiable when the path is executed in order. For example, the guard of the second transition of the path must be satisfiable given the state of the model after the first transition of the path was executed. Let $isfeasible(p)$ be the function $isfeasible\colon \mathcal{T}^* \to \{true, false\}$ that returns whether p is a path and it is feasible as described above.

A path p is *reachable* if either (1) p starts at the initial location of the model and the initial condition of the model is satisfiable, or (2) p does not start at the initial location of the model and there is another feasible path q that starts at the initial location of the model such that the destination location of the last transition of q is the origin location of the first transition of p. Let $isreachable(p)$ be the function $isreacheable(p)\colon \mathcal{T}^* \to \{true, false\}$ that returns whether p is a path and it is reachable as described above.

Thus, a path p is feasible and reachable if either (1) p starts at the initial location of the model, and there is at least one variable valuation that satisfies the initial condition of the model that can make p feasible, or (2) p does not start at the initial location of the model, and there is a feasible path q that starts at the initial location of the model such that the concatenation of paths q and p can make p feasible.

[1]The initial location of a path is the origin location of the first transition of the path. Additionally, we write "a path starts at location l" to mean "the initial location of the path is equal to location l".

Definition 9 (Feasible and reachable paths)**.** *A path p is feasible and reachable if there is at least one execution $e \in Traces(W)$ such that p is included in e. The set of all feasible and reachable paths over the set of transitions $\mathcal{T}$ of the model W is defined by the function $paths(\mathcal{T})$ such that $paths\colon \mathcal{T}^* \to \mathcal{T}^*$ is given by $paths(\mathcal{T}) \mapsto \{p : ispath(p) \wedge isfeasible(p) \wedge isreacheable(p)\}$, where $p \in \mathcal{T}^*$. The set of all feasible and reachable paths in a model may be infinite if the model contains loops.* $\diamond$

Definition 10 (Pass paths)**.** *The pass paths of a test case tc are the paths over the transitions of the test case that start at the initial location of the test case and end in the $Pass$ verdict location. The function $passpaths(tc)$ returns the set of the pass paths of the test case tc such that $passpaths\colon paths(\mathcal{T}_{tc}) \to paths(\mathcal{T}_{tc})$ is given by $passpaths(tc) \mapsto \{p : (l_{p_1} = l^0_{TC}) \wedge (l'_{p_{|p|}} = Pass)\}$, where $p \in paths(\mathcal{T}_{tc})$, l_{p_1} is the origin location of the first transition of the path p, l^0_{tc} is the initial location of tc, $l'_{p_{|p|}}$ is the destination location of the last transition of p, $Pass$ is the $Pass$ verdict location.* $\diamond$

Let $map(u)$ be the function that maps a transition $u \in \mathcal{T}_{tc}$ in a pass path of tc to the corresponding transition $t \in \mathcal{T}$ of the specification W. Thus, by applying the map function to the transitions of the pass paths of a test case, we can determine whether a path in the specification may be covered by the test case.

Definition 11 (Path coverage)**.** *A path p is covered by a test case tc if p is a sub-path of any of the pass paths of tc. The function $pcovered(p, tc)$ returns whether the path p is covered by the test case tc such that $pcovered\colon paths(\mathcal{T}) \times passpaths(\mathcal{T}_{tc}) \to \{true, false\}$ is given by $pcovered(p, tc) \mapsto \exists q \exists n \forall i[(1 \leq i \leq |p|) \implies (p_i = map(q_{i+n}))]$, where $p \in paths(\mathcal{T})$, $q \in passpaths(tc)$, p_i is the i^{th} transition of the path p, q_{i+n} is the $(i+n)^{th}$ transition of the path q, and $n, i \in \mathbb{N}$.* $\diamond$

Definition 12 (Pass executions)**.** *Let $execs(tc) \subseteq Traces(W)$ be the set of finite sequences of discrete and time-elapsing actions corresponding to all executions of the pass paths of the test case tc.* $\diamond$

Definition 13 (Pass states)**.** *The set of states covered by all executions of the pass paths of the test case tc is $states(tc)$ such that $states(tc) \subseteq S$ and $states(tc) \triangleq \{s : (s \in S_0) \vee \forall \sigma[s \in (S_0 \text{ after } \sigma)]\}$, where $s \in S$, $\sigma \in execs(tc)$.* $\diamond$

It should be noted that the set of states of W can also be defined as $S \triangleq \{s : (s \in S_0) \vee \forall\sigma[s \in (S_0 \text{ after } \sigma)]\}$, where $\sigma \in Traces(W)$.

Definition 14 (State coverage). *A state s is covered by a test case tc if s is included in the states covered by any execution of any pass path of tc. The function $scovered(s, tc)$ returns whether the state s is coverage by the test case tc such that $scovered \colon S \times execs(tc) \to \{true, false\}$ is given by $scovered(s, tc) \mapsto s \in states(tc)$.* $\diamond$

3.2 Criteria for TIOSTS models

The first criterion for TIOSTS models is ALL-TRACES criterion.

Criterion 1 (All-Traces). *For each trace e of the specification, there is at least one test case that covers e [31]. Formally, the criterion* ALL-TRACES *is satisfied if $\forall e \exists tc[e \in execs(tc)]$, where $e \in Traces(W)$, $tc \in TS$.* $\diamond$

We classify other criteria for TIOSTS models as either path-based or state-based criteria depending on the type of the requirements used to check for criterion satisfaction.

3.2.1 Path-based criteria

Criterion 2 (All-Paths). *For each path p of the specification, there is at least one test case that covers p [31; 40; 69]. Formally, the criterion* ALL-PATHS *is satisfied if $\forall p \exists tc[pcovered(p, tc)]$, where $p \in paths(\mathcal{T})$, and $tc \in TS$.* $\diamond$

Criterion 3 (All-Transition-Pairs). *For each path p of the specification with length either one or two, there is at least one test case that covers p [69]. Formally, the criterion* ALL-TRANSITION-PAIRS *is satisfied if $\forall p \exists tc[(1 \leq |p| \leq 2) \implies pcovered(p, tc)]$, where $p \in paths(\mathcal{T})$, and $tc \in TS$.* $\diamond$

Criterion 4 (All-Transitions). *For each transition t of the specification, there is at least one test case that covers t [7; 31; 40; 69]. Formally, the criterion* ALL-TRANSITIONS *is satisfied by a test suite TS if $\forall t \exists p \exists tc[((t = p_1) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, where $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, and $tc \in TS$.* $\diamond$

Criterion 5 (All-Clock-Resets). *For each transition t of the specification that resets any clock, there is at least one test case that covers t [31]. If there is no clock to be reset by a transition, the set A^C of the transition is empty. Formally, the criterion* ALL-CLOCK-RESETS *is satisfied by a test suite TS if* $\forall t \exists p \exists tc[((A_t^C \neq \emptyset) \wedge (p_1 = t) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, $tc \in TS$, A_t^C *is the set of clocks to be reset in transition* t. ◇

Criterion 6 (All-Clock-Guards). *For each transition t of the specification that has a clock constraint over some clock, there is at least one test case that covers t [31]. The expression $G^C = true$ means there is no constraint over any clock in the transition. Formally, the criterion* ALL-CLOCK-CONSTRAINTS *is satisfied by a test suite TS if* $\forall t \exists p \exists tc[((G_t^C \neq true) \wedge (p_1 = t) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, *where* $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, $p \in paths(\mathcal{T})$, $tc \in TS$, *and* G_t^C *is the clock constraint over C of transition t, p_1 is the first transition of path p.* ◇

Criterion 7 (All-Locations). *For each location l of the specification, there is at least one test case that covers l. A location is covered by a test case if the location is the origin or destination location of a transition in a path covered by the test case [7; 31; 40; 69]. Formally, the criterion* ALL-LOCATIONS *is satisfied by a test suite TS if* $\forall l \exists t \exists p \exists tc[(((l = l_t) \vee (l = l_t')) \wedge (p_1 = t) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, *where* $l \in L$, $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, $tc \in TS$, $l_t \in L$ *is the origin location of transition t,* $l_t' \in L$ *is the destination location of t, p_1 is the first transition of path p. If* $\mathcal{T} = \emptyset$, *any test suite satisfies the criterion.* ◇

Criterion 8 (All-Input-Actions). *For each input action a of the specification, there is at least one test case that covers a. An input action is covered by a test case if the input action is equal to the action of a transition in a path covered by the test case [31]. Formally, the criterion* ALL-INPUT-ACTIONS *is satisfied if* $\forall a \exists t \exists p \exists tc[((a = a_t) \wedge (p_1 = t) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, *where* $a \in \Sigma^?$, $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, a_t *is the input action of the transition t,* $tc \in TS$. ◇

Criterion 9 (All-Output-Actions). *For each output action a of the specification, there is at least one test case that covers a. An output action is covered by a test case if the output action is equal to the action of a transition in a path covered by the test case [31]. Formally,* *the criterion* ALL-OUTPUT-ACTIONS *is satisfied if* $\forall a \exists t \exists p \exists tc[((a = a_t) \wedge (p_1 = t) \wedge (|p| = 1)) \implies pcovered(p, tc)]$, *where* $a \in \Sigma^!$, $t \in \mathcal{T}$, $p \in paths(\mathcal{T})$, a_t *is the output action of the transition* t, $tc \in TS$. ◇

The criteria ALL-INPUT-ACTIONS and ALL-OUTPUT-ACTIONS are concerned with verifying at least one occurrence of each action with the correct number of parameters, but the values of its parameters do not matter. Since these criteria do not deal with values of its parameters, the criteria do not require all possible values of its parameters to be covered, and variable renaming and output expression equivalences do not matter to the criteria.

Now, we define the concepts of location view, loop-free path and one-loop path. The location view of a path p is a sequence of locations such that the first element of the sequence is the origin location of the first transition of the path p, the following elements of the sequence are the destination locations of the transitions of p in the order they appear in p, and the sequence may have repeated elements. The function $locs(p)$ returns the location view of the path p such that $locs\colon paths(\mathcal{T}) \to L^*$ is given by $locs(p) \mapsto z : (|z| = |p| + 1) \wedge (z_1 = l_{p_1}) \wedge \forall i[(2 \leq i \leq |z|) \implies (z_i = l'_{p_{i-1}})]$, where $p \in paths(\mathcal{T})$, L^* is the set of all sequences that can be composed from the set of locations L, $z \in L^*$, z_i is the i^{th} location in the sequence y, l_{p_1} is the origin location of the first transition of p, $l'_{p_{i-1}}$ is the destination location of the $(i-1)^{th}$ transition of p, and $i \in \mathbb{N}$.

Definition 15 (Loop-free path). *A loop-free path is a path* p *of the TIOSTS such that every location in* $locs(p)$ *appears once. The function isloopfree*(p) *return whether the path* p *is loop-free such that isloopfree*: $paths(\mathcal{T}) \to \{true, false\}$ *given by isloopfree*$(p) \mapsto (z = locs(p)) \wedge \neg\exists i \neg\exists j[(1 \leq i \leq |z|) \wedge (1 \leq j \leq |z|) \wedge (i \neq j) \wedge (z_i = z_j)]$, *where* $p \in paths(\mathcal{T})$, $z \in L^*$, *and* $i, j \in \mathbb{N}$. ◇

Definition 16 (One-loop path). *A one-loop path* p *is a path such that either (1) every location of* $locs(p)$ *appears once in* $locs(p)$, *or (2) a location of* $locs(p)$ *appears twice in* $locs(p)$ *and every other location of* $locs(p)$ *appears once. The function* $isoneloop(p)$ *returns whether the path* p *is one-loop path such that* $isoneloop\colon paths(\mathcal{T}) \to \{true, false\}$ *given by* $isoneloop(p) \mapsto (z = locs(p)) \wedge \neg\exists i \neg\exists j \neg\exists m \neg\exists n[(1 \leq i \leq |z|) \wedge (1 \leq j \leq |z|) \wedge (i \neq j) \wedge (m > 0) \wedge (n > 0) \wedge (i + m \leq |z|) \wedge (j + n \leq |z|) \wedge (z_i = z_{i+m}) \wedge (z_j = z_{j+n})]$, *where* $p \in paths(\mathcal{T})$, $z \in L^*$, *and* $i, j, m, n \in \mathbb{N}$. ◇

Criterion 10 (All-One-Loop-Paths). *For each one-loop path p of the specification, there is at least one test case that covers p [69]. Formally, the criterion* ALL-ONE-LOOP-PATHS *is satisfied if $\forall p \exists tc[isoneloop(p) \implies pcovered(p, tc)]$, where $p \in paths(\mathcal{T})$, and $tc \in TS$.* ◇

Definition 17 (Simple path). *A simple path p is a path such that it is either a loop-free path or a one-loop path such that the origin location of the first transition of p is equal to the destination location of the last transition of p. The function $issimple(p)$ returns whether p is a simple path such that $issimple\colon paths(\mathcal{T}) \to \{true, false\}$ is given by $issimple(p) \mapsto isloopfree(p) \vee (isoneloop(p) \wedge (l_{p_1} = l'_{p_{|p|}}))$, where $p \in paths(\mathcal{T})$, l_{p_1} is the origin location of the first transition of p, and $l'_{p_{|p|}}$ is the destination location of the last transition of p. It should be noted that every simple path is one-loop path too.* ◇

Criterion 11 (All-Simple-Paths). *For each simple path p of the specification, there is at least one test case that covers p [7]. Formally, the criterion* ALL-SIMPLE-PATHS *is satisfied if $\forall p \exists tc[issimple(p) \implies pcovered(p, tc)]$, where $p \in paths(\mathcal{T})$, and $tc \in TS$.* ◇

Data-flow-oriented criteria. Data-flow-oriented criteria define requirements based on paths from a variable definition to a variable use. Thus, we start by formalizing the necessary concepts and auxiliary functions for the criteria. The function $vars(expr)$ receives $expr$, either an expression over variables in $V \cup set(sig(action))$ or a constraint over clocks in C, and returns the set of variables in V or clocks in C that are used in $expr$. The function $left(asgmt)$ returns the variable in V being assigned in the assignment $asgmt$, i.e. the variable in the left-hand side of the assignment. The function $right(asgmt)$ returns the expression being assigned in the assignment $asgmt$, i.e. the expression in the right-hand side of the assignment. The function $isid(asgmt)$ returns whether the assignment $asgmt$ is an identity, i.e. $asgmt$ is of the form $x := x$, where x is a variable V.

Definition 18 (Variable definition). *A variable x is defined in transition t if the assignment of x in t is not an identity. The function $isdefined(x, t)$ returns whether the variable x is defined in transition t such that $isdefined\colon V \times \mathcal{T} \to \{true, false\}$ is given by $isdefined(x, t) \mapsto \neg isid(asgmt_x^t)$, where $x \in V$, $t \in \mathcal{T}$, $asgmt_x^t$ is the assignment of variable x in tuple A^D of the assignments of t.* ◇

Definition 19 (Variable use). *A variable x is used in transition t if either (1) x is part of the guard G^D of t, or (2) if the assignment of x in t is an identify, then x must also be on the right-hand side of another assignment in A^D of t, or (3) if the assignment of x is not an identity, then x must also be on the right-hand side of another assignment A^D of t that does not happen after the assignment of x in A^D of t. The function $isused(x, t)$ returns whether the variable x is used in a transition t such that $isused\colon V \times \mathcal{T} \to \{true, false\}$ is given by $isused(x, t) \mapsto (x \in vars(G_t)) \vee (\exists i \exists j[(x = left(asgmt_i^t)) \wedge isid(asgmt_i^t) \wedge (x \in vars(right(asgmt_j^t)) \wedge (i \neq j)]) \vee (\exists m \exists n[(x = left(asgmt_m^t)) \wedge \neg isid(asgmt_m^t) \wedge (x \in vars(right(asgmt_n^t)) \wedge (n \leq m)])$, where $x \in V$, $t \in \mathcal{T}$, G_t is the guard of t, $asgmt_k^t$ is the k^{th} assignment of A^D of t respectively, and $i, j, m, n \in \mathbb{N}$.* $\diamond$

Definition 20 (DU path). *A definition-use (DU) path dup of a variable x is a simple path of length greater or equal to 2 such that the x is defined in the first transition of dup, x is used in the last transition of dup, and no transition after the first and before the last transitions defines x. A DU path is denoted by $isdup(x, p)$ such that $isdup\colon V \times paths(\mathcal{T}) \to \{true, false\}$ is given by $isdup(x, p) \mapsto issimple(p) \wedge (|p| \geq 2) \wedge isdefined(x, p_1) \wedge isused(x, p_{|p|}) \wedge \forall i[(2 \leq i < |p|) \implies \neg isdefined(x, p_i)]$, where $x \in V$, $p \in paths(\mathcal{T})$, p_i is the i^{th} transition of p, and $i \in \mathbb{N}$.* $\diamond$

The set of all DU paths of a variable x such that x is defined in the transition d and x is used in the transition u is defined as $dups(x, d, u) \triangleq \{p : (p_1 = d) \wedge (p_{|p|} = u) \wedge isdup(x, p)\}$, where $x \in V$, $d, u \in \mathcal{T}$, $p \in paths(\mathcal{T})$.

Criterion 12 (All-DU-Paths). *For each variable x, for each transition d that defines x, for each transition u that uses x, for each DU path dup of x in the specification such that the first transition of dup is d and the last transition is u, there is at least one test case the covers dup [7; 40]. Formally, the criterion* ALL-DU-PATHS *is satisfied if $\forall x \forall d \forall u \forall p \exists tc[(p \in dups(x, d, u)) \implies pcovered(p, tc)]$, where $x \in V$, $d, u \in \mathcal{T}$, $p \in paths(\mathcal{T})$, and $tc \in TS$.* $\diamond$

Criterion 13 (All-Uses). *For each variable x, for each transition d that defines x, for each transition u that uses x, for at least one DU path dup of x in the specification such that the first transition of dup is d and the last transition is u, there is at least one test case that covers dup [7; 40]. Formally, the criterion* ALL-USES *is satisfied if $\forall x \forall d \forall u \exists p \exists tc[(p \in$

$dups(x, d, u)) \implies pcovered(p, tc)]$, *where* $x \in V$, $d, u \in \mathcal{T}$, $p \in paths(\mathcal{T})$, *and* $tc \in TS$. $\diamond$

Criterion 14 (All-Defs)**.** *For each variable* x*, for each transition* d *that defines* x*, for at least one transition* u *that uses* x*, for at least one DU path* dup *of* x *in the specification such that the first transition of* dup *is* d *and the last transition is* u*, exist at least one test case that covers* dup *[7; 40]. Formally, the criterion* ALL-DEFS *is satisfied if* $\forall x \forall d \exists u \exists p \exists tc[(p \in dups(x, d, u)) \implies pcovered(p, tc)]$, *where* $x \in V$, $d, u \in \mathcal{T}$, $p \in paths(\mathcal{T})$, *and* $tc \in TS$. $\diamond$

It should be noted that the values of the variables does not matter for the data-flow-oriented criteria. What matters for these criteria is to cover a set of simple paths specified by each criterion.

3.2.2 State-based criteria

Criterion 15 (All-States)**.** *For each state* s *of the specification, there is at least one test case that covers* s *[31; 40]. Formally, the criterion* ALL-STATES *is satisfied if* $\forall s \exists tc[scovered(s, tc)]$, *where* $s \in S$, $tc \in TS$. $\diamond$

Given $s \in S$, the location of s is denoted by l_s, the variable valuation by ν_s, and the clock valuation by ψ_s.

Criterion 16 (All-Clock-Valuations)**.** *For each clock valuation* ψ *of the specification, there is at least one test case that covers a state* s *in which its clock valuation is* ψ *[31]. Formally,* $\forall \psi \exists s \exists tc[\psi_s = \psi \implies scovered(s, tc)]$, *where* $\psi, \psi_s \in \Psi$, $s \in S$, $tc \in TS$. $\diamond$

Definition 21 (Bound of clock)**.** *The bound of clock of the clock* c *is the largest integer* b *appearing in a constraint over* c *within all clock constraints of the model. It is denoted by* B_c *[31].*

Criterion 17 (All-Clock-Bounds)**.** *For each clock* c *of the model, there is at least one test case that covers a state which its clock valuation satisfies the clock constraint* $c = B_c$ *[31]. Formally,* $\forall c \exists s \exists tc[(\psi_s \models (c = B_c)) \implies scovered(s, tc)]$, *where* $c \in C$, $\psi_s \in \Psi$, $tc \in TS$. $\diamond$

For any $\psi \in \Psi$, $\lfloor \psi \rfloor$ denotes the integral part of ψ, and $frac(\psi)$ denotes the fractional part of ψ; i.e. $\psi = \lfloor \psi \rfloor + frac(\psi)$. We assume that each clock in C appears in some clock constraint in W.

Definition 22 (Clock region). *Clock valuations ψ and ψ' in Ψ are said equivalent, denoted by $\psi \sim \psi'$, if and only if all the following three conditions hold:*

- $\forall x[(\lfloor \psi(x) \rfloor = \lfloor \psi'(x) \rfloor) \vee (\psi(x) > B_x \wedge \psi'(x) > B_x)]$, *where $c \in C$;*
- $\forall x \forall y[(\psi(x) \leq B_x \wedge \psi(y) \leq B_y) \implies (frac(\psi(x)) \leq frac(\psi(y)) \iff frac(\psi'(x)) \leq frac(\psi'(y)))]$, *where $x, y \in C$;*
- $\forall x[\psi(x) \leq B_x \implies (frac(\psi(x)) = 0 \iff frac(\psi'(x) = 0)]$, *where $x \in C$.*

A clock region *for W is an equivalence class of clock valuations induced by $\sim$, and it can be uniquely characterized by a (finite) set of clock constraints over C. The set of all clock regions is $Regions(W)$ [31].* $\diamond$

Criterion 18 (All-Clock-Regions). *For each clock region r of the specification, there is at least one test case that covers a state which its clock valuation satisfies cr [31]. Formally, $\forall r \exists s \exists tc[(\psi_s \models r) \implies scovered(s, tc)]$, where $r \in Regions(W)$, $s \in S$, $\psi_s \in \Psi$, $tc \in TS$.* $\diamond$

Definition 23 (Clock zone). *A zone of W is a convex polyhedron formed by the clock valuations that satisfy a clock a guard of some transition of the model [10]. A zone can be represented by union of some clock regions in $Regions(W)$. The set of zones of W is denoted by $Zones(W)$.* $\diamond$

Criterion 19 (All-Clock-Zones). *For each clock zone z of the specification, there is at least one test case that covers a state which its clock valuation satisfies z. Formally, $\forall z \exists s \exists tc[(\psi_s \models z) \implies scovered(s, tc)]$, where $z \in Zones(W)$, $s \in S$, $\psi_s \in \Psi$, $tc \in TS$.* $\diamond$

3.3 Formalization of a hierarchy of criteria

Our goal is to produce a sound hierarchy of test adequacy criteria partially ordered by strict inclusion relation, which is the standard theoretical comparison between two test adequacy

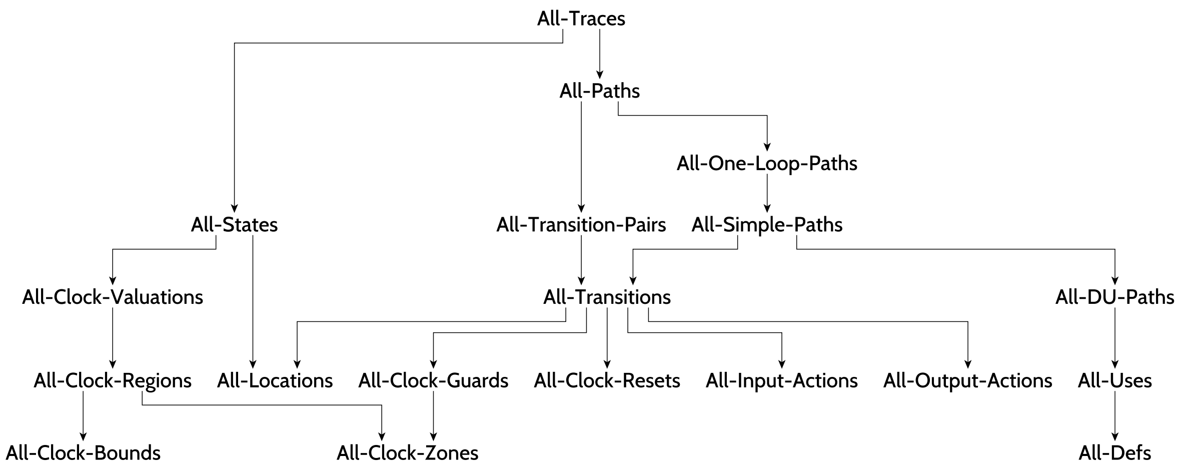


Figure 3.1: Family of test adequacy criteria for TIOSTS models ordered by strict inclusion.

criteria [61]. This relation is formalized in Definition 24, and the proposed hierarchy is formalized in Theorem 1.

Definition 24 (Strict Inclusion[2])**.** *A criterion c_1 strictly includes a criterion c_2 if any test suite that satisfies the criterion c_1 also satisfies the criterion c_2, but there is at least one test suite that satisfies the criterion c_2 that does not satisfy the criterion c_1 for some model. Strict inclusion from criterion c_1 to criterion c_2 is denoted by $c_1 \dashrightarrow c_2$. Strict inclusion is a transitive relation [61].* ◇

Theorem 1. *The family of criteria for TIOSTS is partially ordered by strict inclusion as shown in Figure 3.1. Furthermore, $c_1 \dashrightarrow c_2$ if and only if it is explicitly shown to be so in Figure 3.1 or follows from the transitivity of the relation.*

Proof. We prove the strict inclusions between pairs established in Figure 3.1.

1. ALL-TRACES $\dashrightarrow$ ALL-PATHS: Let TS be a test suite that satisfies ALL-PATHS. By Definition 12, the set of all executions covered by TS is $(\bigcup_{tc \in TS} execs(tc)) \subseteq Traces(W)$. By Criterion 1, ALL-TRACES requires coverage of every execution $e \in Traces(W)$, therefore any test suite that satisfies ALL-TRACES also satisfies ALL-PATHS. However, there might be $Traces(W) \setminus (\bigcup_{tc \in TS} execs(tc)) \neq \emptyset$ and, consequently, TS does not satisfy ALL-TRACES.

[2]Strict inclusion is also called subsume in literature.

2. ALL-TRACES $\rightarrow$ ALL-STATES: Let TS be a test suite that satisfies ALL-TRACES. By Definition 13, the set of all states covered by TS is $(\bigcup_{tc \in TS} states(tc)) \subseteq S$. By Criterion 15, ALL-STATES requires coverage of every state $s \in S$, therefore any test suite that satisfies ALL-TRACES also satisfies ALL-STATES. However, there might be $S \setminus (\bigcup_{tc \in TS} states(tc)) \neq \emptyset$ and, consequently, TS does not satisfy ALL-TRACES.

3. ALL-PATHS $\rightarrow$ ALL-ONE-LOOP-PATHS: Holds trivially because, from Definition 15, Definition 16, Criterion 2 and Criterion 10, every loop-free path is a path and every one-loop path is also a path, however not every path is a loop-free path or a one-loop path.

4. ALL-PATHS $\rightarrow$ ALL-TRANSITION-PAIRS: Holds trivially because, from Criterion 2 and Criterion 3, every path of length one or two is a path, however not every path has length one or two.

5. ALL-ONE-LOOP-PATHS $\rightarrow$ ALL-SIMPLE-PATHS: Holds trivially because, from Definition 15, Definition 16, Definition 17, Criterion 10 and Criterion 11, the set of paths that ALL-SIMPLE-PATHS must cover is a subset of the set of paths that ALL-ONE-LOOP-PATHS must cover.

6. ALL-SIMPLE-PATHS $\rightarrow$ ALL-TRANSITIONS: Holds trivially because, from Definition 17, Criterion 4 and Criterion 11, every path of length 1 is a simple path, however not every simple path has length 1.

7. ALL-SIMPLE-PATHS $\rightarrow$ ALL-DU-PATHS: Holds trivially because, from Definition 17, Definition 20, Criterion 11 and Criterion 12, every DU path is a simple path, however not every simple path is a DU path.

8. ALL-DU-PATHS $\rightarrow$ ALL-USES: Holds trivially because, from Criterion 12 and Criterion 13, the set of DU paths that ALL-USES must cover is a subset of the set of DU paths that ALL-DU-PATHS must cover.

9. ALL-USES $\rightarrow$ ALL-DEFS: Holds trivially because, from Criterion 13 and Criterion 14, the set of DU paths that ALL-DEFS must cover is a subset of the set of DU paths that ALL-USES must cover.

10. ALL-TRANSITION-PAIRS $\rightarrow$ ALL-TRANSITIONS: Holds trivially because, from Criterion 3 and Criterion 4, ALL-TRANSITIONS requires coverage of all paths of length 1, however ALL-TRANSITION-PAIRS requires coverage of all paths of length 1 and all paths of length 2.

11. ALL-TRANSITIONS $\rightarrow$ ALL-LOCATIONS: Holds trivially because, from Definition 3, Criterion 4 and Criterion 7, every location is a origin or destination location of a transition, and all locations are covered by covering all transitions. However, there might be two transitions t_1 and t_2 which have a location l in common such that, by covering t_1, l is covered and ALL-LOCATIONS is satisfied, but t_2 is not covered and, consequently, TS does not satisfy ALL-TRANSITIONS.

12. ALL-TRANSITIONS $\rightarrow$ ALL-CLOCK-GUARDS: Holds trivially because, from Criterion 4 and Criterion 6, some transitions may not have a clock guard, therefore the set of paths of length 1 required by ALL-CLOCK-GUARDS is a subset of all paths of length 1, which is required by ALL-TRANSITIONS,

13. ALL-TRANSITIONS $\rightarrow$ ALL-CLOCK-RESETS: Holds trivially because, from Criterion 4 and Criterion 5, some transitions may not have a clock reset, therefore the set of paths of length 1 required by ALL-CLOCK-GUARDS is a subset of all paths of length 1, which is required by ALL-TRANSITIONS.

14. ALL-TRANSITIONS $\rightarrow$ ALL-INPUT-ACTIONS: Holds trivially because, from Criterion 4 and Criterion 8, some transitions may have output actions instead of input actions, therefore the set of paths of length 1 required by ALL-INPUT-ACTIONS is a subset of all paths of length 1, which is required by ALL-TRANSITIONS.

15. ALL-TRANSITIONS $\rightarrow$ ALL-OUTPUT-ACTIONS: Holds trivially because, from Criterion 4 and Criterion 9, some transitions may have input actions instead of output actions, therefore the set of paths of length 1 required by ALL-OUTPUT-ACTIONS is a subset of all paths of length 1, which is required by ALL-TRANSITIONS.

16. ALL-STATES $\rightarrow$ ALL-LOCATIONS: Let TS be a test suite that satisfies ALL-LOCATIONS. By Definition 4, Definition 13, Definition 7, and Criterion 7, the coverage of the set

of states $\mathit{AL} \triangleq (\bigcup_{tc \in TS} states(tc)) \subseteq S$ satisfies ALL-LOCATIONS. By Criterion 15, ALL-STATES requires coverage of every $s \in S$. Since $\mathit{AL} \subseteq S$, then any test suite that satisfies ALL-STATES also satisfies ALL-LOCATIONS. However, there might be $S \setminus \mathit{AL} \neq \emptyset$ and, consequently, TS does not satisfy ALL-STATES.

17. ALL-STATES $\rightarrow$ ALL-CLOCK-VALUATIONS: Let TS be a test suite that satisfies ALL-CLOCK-VALUATIONS. By Definition 4, Definition 13, and Criterion 16, the coverage of the set of states $\mathit{ACV} \triangleq (\bigcup_{tc \in TS} states(tc)) \subseteq S$ satisfies ALL-CLOCK-VALUATIONS. By Criterion 15, ALL-STATES requires coverage of every $s \in S$. Since $\mathit{ACV} \subseteq S$, then any test suite that satisfies ALL-STATES also satisfies ALL-CLOCK-VALUATIONS. However, there might be $S \setminus \mathit{ACV} \neq \emptyset$ and, consequently, TS does not satisfy ALL-STATES.

18. ALL-CLOCK-VALUATIONS $\rightarrow$ ALL-CLOCK-REGIONS: Let TS be a test suite that satisfies ALL-CLOCK-REGIONS. By Definition 4, Definition 13, and Criterion 18, the coverage of the set of clock valuations $\mathit{ACR} \triangleq \{\psi_s : s \in (\bigcup_{tc \in TS} states(tc))\} \subseteq \Psi$, where ψ_s is the clock valuation of state s, satisfies ALL-CLOCK-REGIONS. By Criterion 16, ALL-CLOCK-VALUATIONS requires coverage of every $\psi \in \Psi$. Since $\mathit{ACR} \subseteq \Psi$, then any test suite that satisfies ALL-CLOCK-VALUATIONS also satisfies ALL-CLOCK-REGIONS. However, there might be $\Psi \setminus \mathit{ACR} \neq \emptyset$ and, consequently, TS does not satisfy ALL-CLOCK-VALUATIONS.

19. ALL-CLOCK-REGIONS $\rightarrow$ ALL-CLOCK-ZONES: Let TS be a test suite that satisfies ALL-CLOCK-ZONES. By Definition 22 and Definition 23, a clock zone in $Zones(W)$ can be represented by the union of some clock regions in $\mathit{Regions}(W)$. By Criterion 19, Definition 22, and Definition 23, the coverage of the set of clock regions $ACZ \triangleq \{r \in \mathit{Regions}(W) : (\psi_s \models r) \wedge s \in (\bigcup_{tc \in TS} states(tc))\} \subseteq \mathit{Regions}(W)$, where ψ_s is the clock valuation of state s, satisfies ALL-CLOCK-ZONES. Since $ACZ \subseteq \mathit{Regions}(W)$, then any test suite that satisfies ALL-CLOCK-REGIONS also satisfies ALL-CLOCK-ZONES. However, there might be $\mathit{Regions}(W) \setminus ACZ \neq \emptyset$ and, consequently, TS does not satisfy ALL-CLOCK-REGIONS.

20. ALL-CLOCK-REGIONS $\rightarrow$ ALL-CLOCK-BOUNDS: Let TS be a test suite that satis-

fies ALL-CLOCK-BOUNDS. By Definition 21 and Definition 22, a clock constraint of the form $c = B_c$, where B_c is the bound of clock c, can be satisfied by some clock regions in $Regions(W)$. By Criterion 17, Definition 21, and Definition 22, the coverage of the set of clock regions $ACB \triangleq \{r \in Regions(W) : (\psi_s \models r) \wedge s \in (\bigcup_{tc \in TS} states(tc))\} \subseteq Regions(W)$, where ψ_s is the clock valuation of state s, satisfies ALL-CLOCK-BOUNDS. Since $ACB \subseteq Regions(W)$, then any test suite that satisfies ALL-CLOCK-REGIONS also satisfies ALL-CLOCK-BOUNDS. However, there might be $Regions(W) \setminus ACB \neq \emptyset$ and, consequently, TS does not satisfy ALL-CLOCK-REGIONS.

21. ALL-CLOCK-GUARDS $\rightarrow$ ALL-CLOCK-ZONES: Holds trivially because, from Definition 23, Criterion 6, Criterion 19, each clock zone z is formed by the clock valuations that satisfy a clock a guard of some transition $t \in \mathcal{T}$, therefore, by covering all clock guards, all clock zones are also covered. However, there might be two transitions t_1 and t_2 with the same clock guard g and, by covering t_1, g is covered and ALL-CLOCK-ZONES is satisfied, but t_2 is not covered and, consequently, TS does not satisfy ALL-CLOCK-GUARDS.

□

3.4 Discussion

Besides our family, test adequacy criteria for different kinds of models of real-time systems have already been investigated, but most proposals just describe a criterion or a set of criteria without formal rigor and do not evaluate empirically the criteria. Since our work extends En-Nouaary's hierarchy of criteria [31], it is the most related to ours.

En-Nouaary proposes a hierarchy of test adequacy criteria for Timed Input-Output Automata (TIOA) models ordered by strict inclusion. His family combines transition-based criteria, reactive systems criteria, and real-time systems criteria, but data-related criteria are not included because the TIOA model does not support data abstraction. Conversely, the TIOSTS model symbolically abstracts both time and data, thus data-related criteria can be applied to it. In this paper, we extend En-Nouaary's family to include data-related criteria for TIOSTS models.

We refine the relation between ALL-CLOCK-RESETS and the transition-based criteria. In En-Nouaary's hierarchy, ALL-PATHS $\rightarrow$ ALL-CLOCK-RESETS, but we proved that the narrow relation ALL-TRANSITIONS $\rightarrow$ ALL-CLOCK-RESETS is true too. We also introduced the relation ALL-STATES $\rightarrow$ ALL-LOCATIONS that was missing. En-Nouaary's family has neither the criteria ALL-ONE-LOOP-PATHS and ALL-TRANSITION-PAIRS nor the data-flow-oriented criteria. We introduced the first two below the ALL-PATHS criterion and above the ALL-TRANSITIONS, and the data-flow-oriented criteria below ALL-ONE-LOOP-PATHS. We removed the criterion ALL-CLOCK-GUARD-BOUNDS because its definition was not clear. We also removed ALL-TIME-CONSTRAINTS because it was redundant with ALL-CLOCK-ZONES.

In this chapter, we proposed 19 test adequacy criteria for TIOSTS models, and we ordered them by strict inclusion. The strict inclusion relation between criteria indicates the relative effort to satisfy a criterion and its potential effectiveness. Yet, only empirical evaluations are capable of comparing actual cost-effectiveness of criteria in practice. Thus, we evaluated 9 of them in an experiment which is discussed in the next chapter.

3.5 Concluding Remarks

In this chapter, we presented test adequacy criteria that can be applied to symbolic transition models of real-time systems, particularly, the TIOSTS model. First, we searched for test adequacy criteria applicable to models of real-time systems. Next, we selected the ones applicable to TIOSTS and formalized a hierarchy of 19 test adequacy criteria partially ordered by strict inclusion. The family of criteria combines general-purpose transition-based and data-flow-oriented criteria with specific reactive and real-time systems criteria. We proved that the theoretical relationship between transition-based and data-flow-oriented criteria in our family is different from the relationship between the corresponding criteria for source code.

Chapter 4

Experiment

In this chapter, we present one experiment we conducted in order to answer the second research question of this dissertation — *what is the influence of test adequacy criteria for TIOSTS models on test suite reduction for model-based testing of real-time systems?*

We followed the guidelines on experimentation in software engineering given by Wohlin et al. [71], and we detail how we carried the planning, operation, results and analysis of the experiment in the next sections.

4.1 Planning

In this section, we detail the planning of the experiment.

The goal of the experiment was to analyze test adequacy criteria for TIOSTS models and test suite reduction techniques for the purpose of comparison with respect to mutation killing effectiveness and time execution cost of the reduced test suites to assess the influence of criteria and techniques on test suite reduction from the point of view of the tester in the context of model-based testing of real-time systems. We conducted the experiment in a research laboratory, thus it was an off-line study with a specific context.

4.1.1 Selection of the variables

The independent variable were the *test suite reduction technique* and the *test adequacy criterion*. For the techniques, we chose the four most evaluated techniques (Greedy, GE, GRE, and HGS) [26], and a random-based technique to act as a baseline. For the criteria, we chose 12 from our proposed family of criteria as described below.

A criterion was eligible for inclusion in the experiment if its requirements could be satisfied by a test suite generated with the goal of conformance (contract) testing, because this is the type of test the SYMBOLRT tool [10; 11] can generate. Additionally, a test case must satisfy the same set of requirements of a criterion regardless of how many times the test case is executed and the data used in those executions. For example, ALL-CLOCK-REGIONS was not eligible because it requires all clock regions to be covered, but every time a test case is executed probably a different set of clock regions will be covered. Moreover, in conformance testing, the coverage of any clock region that satisfies a time constraint is enough to continue the testing of the system, however it is not enough to satisfy the criterion ALL-CLOCK-REGIONS. Finally, we also excluded criteria that are difficult to check whether they are satisfied, e.g. the ALL-CLOCK-BOUNDS criterion. So there are 12 out of 19 criteria of the family that satisfy the eligibility requirements, all of them are path-based criteria: ALL-LOCATIONS, ALL-TRANSITIONS, ALL-TRANSITION-PAIRS, ALL-SIMPLE-PATHS, ALL-ONE-LOOP-PATHS, ALL-INPUT-ACTIONS, ALL-OUTPUT-ACTIONS, ALL-DEFS, ALL-USES, ALL-DU-PATHS, ALL-CLOCK-RESETS, and ALL-CLOCK-GUARDS. They include a mix of transition-based, data-flow-oriented, and real-time systems criteria.

We are interested in *size*, *cost* and *effectiveness* of reduced test suites selected by the techniques and criteria, so we used three surrogate measures as dependent variables. *Size* was measured as the *number of test cases* in the reduced test suite. Each symbolic test case was mapped to a concrete test case. A symbolic test case is test case represented by a TIOSTS (cf. Definition 5), while a concrete test case is the realization of a symbolic test case and can be run against actual implementations. In this chapter, concrete test cases were implemented in JAVA with the JUNIT testing framework[1], version 4. Therefore, the number of test cases was the number of concrete test cases of the test suite. *Cost* was measured in terms of *execution time* of the test suite. The test suite execution time was calculated as sum of the execution times of the test cases that compose the test suite. The execution time of a test case, in turn, was the average of the virtual time spent on executions of the test case (details in section 4.2.1). And the *effectiveness* was measured by *number of killed mutants*

[1]http://junit.org

by the test suite, i.e. The number of faulty implementations the test suite can identify as defective. [7].

We used the number of killed mutants instead of the mutation score because statistical analysis with mutation score would yield the same results with the number of killed mutants since our experiment considered mutants of just one experimental unit at a time. For the same reason, no normalization among experimental units was needed. By using the number of killed mutants, we also avoided the expensive manual process of classifying mutants as either equivalent or non-equivalent.

Thus questions we are looking to answer with the experiment are:

- *SQ1*: What test adequacy criterion yields the best cost-effectiveness: ALL-LOCATIONS, ALL-TRANSITIONS, ALL-TRANSITION-PAIRS, ALL-SIMPLE-PATHS, ALL-ONE-LOOP-PATHS, ALL-DEFS, ALL-USES, ALL-DU-PATHS, ALL-CLOCK-RESETS, ALL-CLOCK-GUARDS, ALL-INPUT-ACTIONS, or ALL-OUTPUT-ACIONS?
- *SQ2*: What test suite reduction technique yields the best cost-effectiveness: GREEDY, GE, GRE, HGSA, or RANDOM?
- *SQ3*: How much of the variation in the cost and effectiveness of reduced test suites is explained by criteria and techniques?

4.1.2 Choice of the type of design

The experiment has two factors of primary interest (test adequacy criterion and test suite reduction technique) and three dependent variables (number of test cases, execution time, and number of killed mutants). Consequently, the natural design for each dependent variable was a two-way factorial design [53; 71]. For each combination of technique and criterion, we selected 1,000 test suites that satisfied the criterion by using the technique.

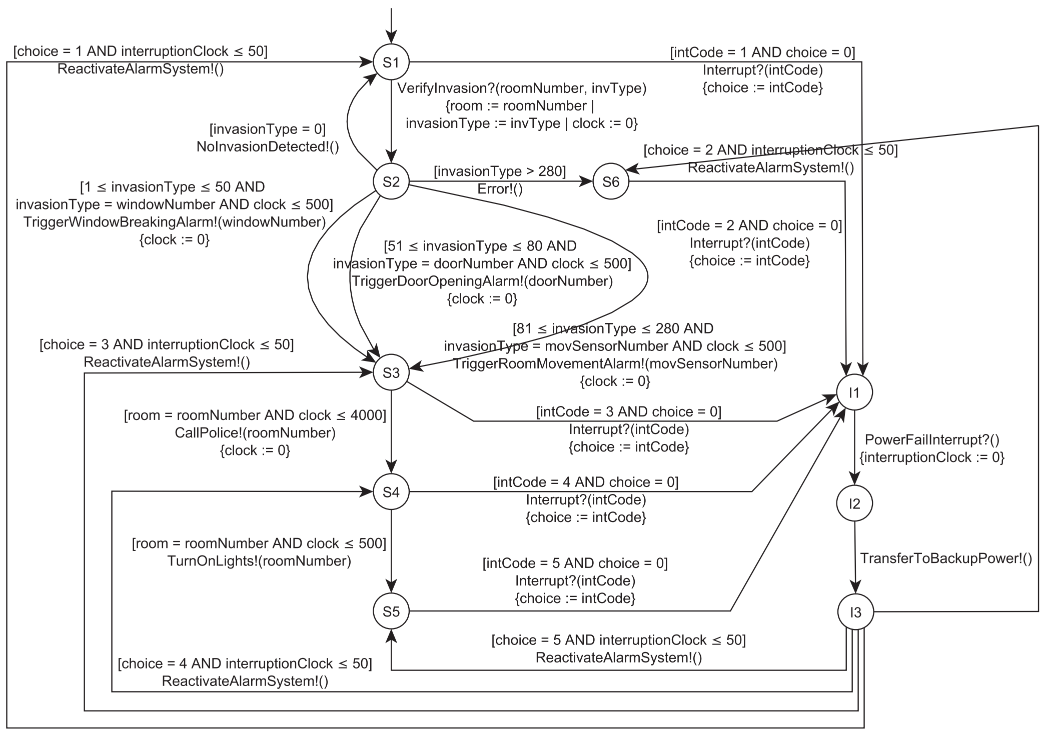
4.1.3 Selection of the experimental units

The experiment has three experimental units, and a distinct specification model for each one: a refilling machine for charging the subway card (SUB) [9], a burglar alarm system [10] and an automated car speed limiter [2; 49]. The refilling machine is a benchmark model which demonstrates the potential of the TIOSTS model and the SYMBOLRT tool. The burglar alarm system represents the class of systems with interruptions, which are hard to test. The car speed limiter is an industrial model that generates a large test suite. Therefore, each model explores a different aspect and challenge of model-based testing of real-time systems. It is also worth to note that the models seem small at a first glance due to reduced number of locations and transitions, however we recall they are compact just because of the expressiveness of the TIOSTS model, which symbolically abstracts both time and data. However, if we were to model these systems with a less expressive model, they would be much larger.

The first model was already introduced in Section 2.2. The second model is a Burglar Alarm System (BAS) [10]. Its purpose is to monitor sensors in order to detect the presence of intruders in a building. This system uses different kinds of sensors, including movement detectors in individual rooms, window sensors which detect the breaking of a window, and door sensors which detect the opening of doors. There are 50 window sensors, 30 door sensors, and 200 movement detectors. When a sensor indicates the presence of an intruder, the system automatically calls the police and switches on lights around the area with the activated sensors. The system is normally powered by the central power supply system of the house, but it is also equipped with a battery backup. The system switches automatically to backup power when a voltage drop is detected.

The TIOSTS specification model of the BAS system is illustrated in Figure 4.1. The specification is deterministic, and has nine locations and 20 transitions (7 transitions with input actions and 13 transitions with output actions). The specification has three variables (`room`, `invasionType`, and `choice`) and two clocks (`interruptionClock` and `clock`). The code of the area where a sensor has been activated is defined by `room`. Variable `invasionType` is used to identify which type of sensor has been activated. When the system detects a voltage drop, an interruption routine takes place to switch to backup power. After the switch to backup power is finished, the system must resume what it was doing before the interruption, and variable `choice` is used to keep track of the system location before the interruption in order to ensure the system resumes correctly. The clocks `interruptionClock` and `clock` are used to specify time requirements.

The last model is an Automated Speed Limiter (ASL) system inspired by an industrial



[choice = 1 AND interruptionClock ≤ 50]
ReactivateAlarmSystem!()
S1
[intCode = 1 AND choice = 0]
Interrupt?(intCode)
{choice := intCode}
VerifyInvasion?(roomNumber, invType)
{room := roomNumber |
invasionType := invType | clock := 0}
[invasionType = 0]
NoInvasionDetected!()
[choice = 2 AND interruptionClock ≤ 50]
ReactivateAlarmSystem!()
[invasionType > 280]
Error!()
S2
S6
[1 ≤ invasionType ≤ 50 AND
invasionType = windowNumber AND clock ≤ 500]
TriggerWindowBreakingAlarm!(windowNumber)
{clock := 0}
[intCode = 2 AND choice = 0]
Interrupt?(intCode)
{choice := intCode}
[51 ≤ invasionType ≤ 80 AND
invasionType = doorNumber AND clock ≤ 500]
TriggerDoorOpeningAlarm!(doorNumber)
{clock := 0}
[81 ≤ invasionType ≤ 280 AND
invasionType = movSensorNumber AND clock ≤ 500]
TriggerRoomMovementAlarm!(movSensorNumber)
{clock := 0}
[choice = 3 AND interruptionClock ≤ 50]
ReactivateAlarmSystem!()
S3
I1
[room = roomNumber AND clock ≤ 4000]
CallPolice!(roomNumber)
{clock := 0}
[intCode = 3 AND choice = 0]
Interrupt?(intCode)
{choice := intCode}
PowerFailInterrupt?()
{interruptionClock := 0}
S4
[intCode = 4 AND choice = 0]
Interrupt?(intCode)
{choice := intCode}
I2
[room = roomNumber AND clock ≤ 500]
TurnOnLights!(roomNumber)
[intCode = 5 AND choice = 0]
Interrupt?(intCode)
{choice := intCode}
TransferToBackupPower!()
S5
I3
[choice = 5 AND interruptionClock ≤ 50]
ReactivateAlarmSystem!()
[choice = 4 AND interruptionClock ≤ 50]
ReactivateAlarmSystem!()

Figure 4.1: TIOSTS specification model of the Burglar Alarm System.

case study provided by Volvo [2; 49]. The purpose of the system is to maintain the speed of a car according to a desired speed limit. The system can assume three states: off, limiting speed, or temporarily overriding speed limit. From the off state, the system can be activated and the car speed will be set to either a driver-defined speed limit or a manufacturer-preset speed limit. While in limiting speed mode, the speed limit can be increased and decreased manually, and a kickdown of the gas pedal overrides the speed limit for some time threshold before the system resumes control of car speed. At any time, the driver can turn off the system.

The TIOSTS specification model of the ASL system is illustrated in Figure 4.2. The specification is deterministic, and has 4 locations and 14 transitions (11 transitions with input actions and 3 transitions with output actions). The specification has four variables (`speed`, `limit`, `preset`, and `boost`) and one clock (`clock`). The current speed of the car is defined by `speed`, the target speed of the system by `limit`, and the fixed predefined target speed by `preset`. Variable `boost` is used to define the speed increase of a kickdown while the system is limiting the speed. The clock `clock` is used to specify time requirements.

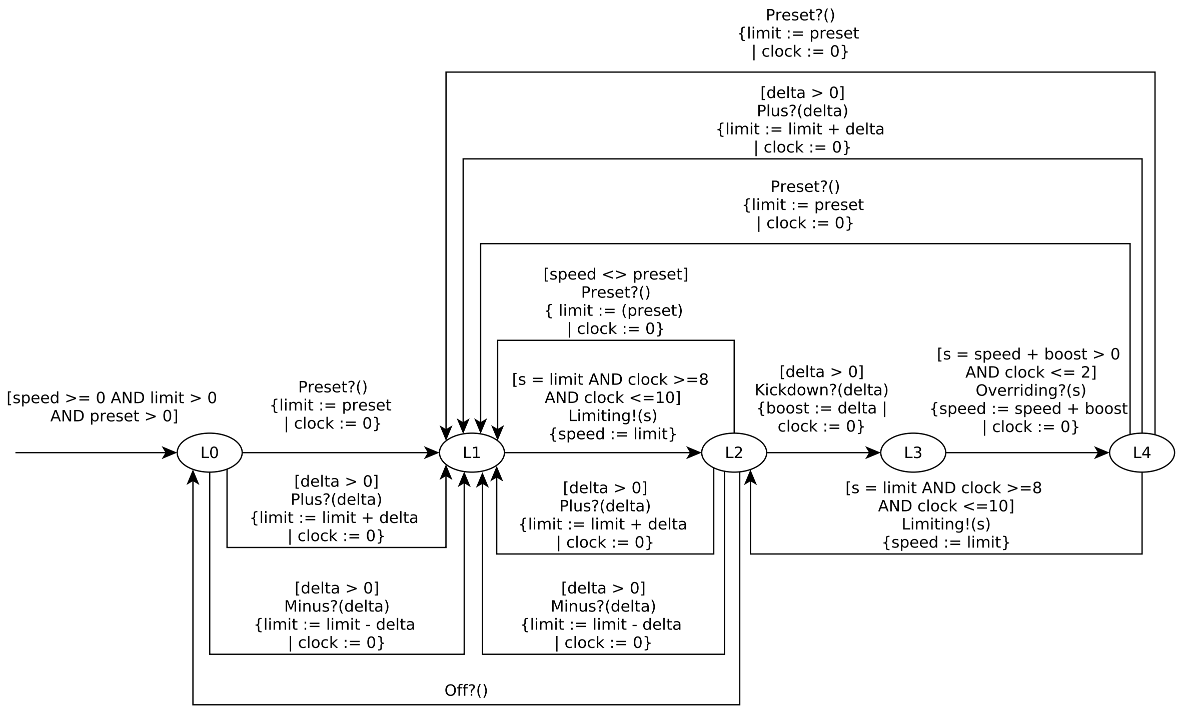


Figure 4.2: TIOSTS specification model of the Automated Speed Limiter.

4.2 Operation

In this section, we describe the steps to execute the experiment. The process is illustrated in Figure 4.3. The experiment was run in a computer equipped with DEBIAN GNU/LINUX, version Jessie, Intel® Core^{TM} i7-4510 CPU 2.00GHz, and 16 GB of memory RAM.

4.2.1 Preparation

Generation of the symbolic test suite

SYMBOLRT is a test case generator for TIOSTS models [9]. The tool receives as input a specification and a test purpose, and generates all the symbolic test cases that satisfy the test purpose. Specifications of real-time systems that run continuously have loops, which may generate infinite paths to traverse, as so the symbolic test cases that are generated from them. Hence, in order to generate concrete test cases with finite executions we must control how many times the loops are traversed.

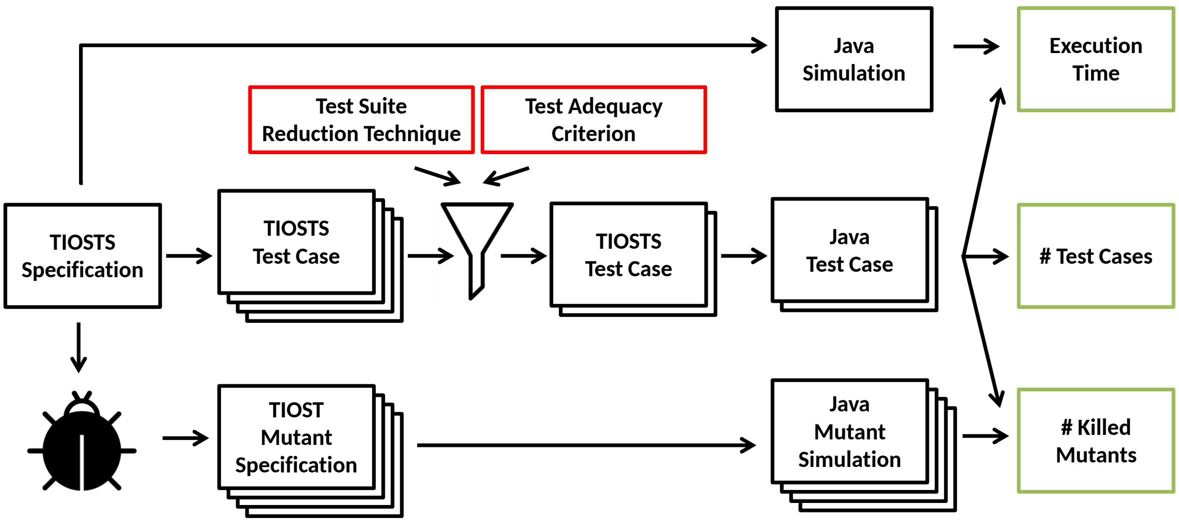


Figure 4.3: Experiment design steps.

In order to control the number of loop traversals of the specification in symbolic test cases, we extended SYMBOLRT to support the Parameterized Depth-First Search (PDFS) algorithm which unrolls a specification with a possibly infinite number of traversals due to loops into a specification with finite number of traversals [28]. The PDFS algorithm receives an input parameter which defines how many times each loop in the specification should be traversed. After applying the PDFS algorithm to a specification, the specification is transformed into a tree that represent traversals in the original specification and it is used to generate the symbolic test cases. Next, the tool traverses the transformed specification —from the initial location to the leaves— and all feasible paths are used as basis for the generation of symbolic test cases.

In the experiment, we applied the PDFS algorithm to the refilling machine and burglar alarm system specifications with parameter values zero, one and two, but with parameter values zero and one to the automated speed limiter specification. These parameter values generated a reasonable number of test cases for each specification model. When the algorithm runs with input parameter zero, it generates test cases with pass paths that makes no loop in the specification. When it runs with inputs one and two, it generates test cases with pass paths such that each loop in the specification is traversed one and two times respectively.

There were generated 3, 12, and 27 symbolic test cases for the refilling machine with algorithm parameter values zero, one and two respectively, totalizing 39 symbolic test cases.

For the burglar alarm system there were generated 12, 13, and 14 test cases respectively with values zero, one and two; also a total of 39 test cases. Finally, for the automated speed limiter there were generated 26 test cases with parameter value zero and 330 test cases with value one; a total of 356 test cases.

Generation of the concrete test suite

We developed a module in SYMBOLRT to generate test data for a symbolic test case. We used CVC3 [15] constraint solver to generate data for input action parameters, and we also managed clock valuations to make sure the symbolic test case could be executed from the initial location to the pass location. Finally, we translated the symbolic test case and its data into a concrete test case, i.e. a JUNIT test suite.

Since a JUNIT TEST CASE has only two possible verdicts (*Pass* and *Fail*) and a symbolic test case may have three verdicts (*Pass*, *Fail*, and *Inconclusive*), we mapped symbolic test case verdicts *Pass* and *Inconclusive* to JUNIT *Pass* verdict. We decided to map the inconclusive verdict of a symbolic test case to the pass verdict of JUNIT after observing a sample of concrete test case executions that reached the inconclusive verdict of the corresponding symbolic test cases. All analyzed concrete test case executions reached an inconclusive verdict of the symbolic test case because of a non-conformity with a time constraint. Strictly, this is not a failure because that behavior is specified but it is not desirable to be observed in the scenario being tested by the test case. Thus we opted to report inconclusive verdicts of symbolic test cases as a pass in concrete test case executions.

Implementations of the systems under test

We implemented a compiler which takes as input a TIOSTS specification model and outputs a simulation of the system in JAVA. The simulation has two components: the environment and the System Under Test (SUT). The environment component controls the SUT execution by sending two types of actions to the SUT: time-elapsing actions and input actions. In our simulation, time is virtual and discrete, and time-elapsing actions increase time by one unit. After sending an action to SUT, the environment waits for the SUT to react. The environment also records all output actions sent by the SUT as well as their time of occurrence.

The SUT component imitates the specification because it has the same variables, clocks, locations and transitions. The SUT keeps the current location, and stays there until the environment sends an action to the SUT. The SUT reacts upon the arrival of a new action. If the new action is an input, then the SUT reads the action name and its parameter values. Next, the SUT matches all outgoing transitions of its current location with that input action, and verifies whether a guard of any outgoing transition of the current location is enabled by the current SUT state plus the input action parameter values. Finally, the SUT clears the input buffer, even if no outgoing transition was enabled. If the new action is time-elapsing, then the SUT increases all clocks by one time unit, and proceeds to verify whether an outgoing transition of its current location is enabled by the new SUT state.

When an outgoing transition is enabled in the SUT, it updates its state: changes variable values, updates clocks, and moves to a new location according to the specification. If the enabled transition contains an output action, then the SUT sends it before updating its state. The SUT will continue this reacting loop until there is no enabled outgoing transition from its current location. At this point, the execution goes back to the environment. Thus, a single action from the environment may trigger a chain of actions of the SUT.

Transitions with output actions may have three deadlines: eager, delayable, and lazy. Transitions with an eager deadline are taken at the same time the transition is enabled. On the other hand, transitions with a delayable deadline are taken only when the output action is ready, but this may happen at any time within the specified clock guard. Finally, transitions with a lazy deadline are transitions without a clock guard. In order to avoid waiting too long for system responses in lazy transitions, we force the SUT to take lazy transitions by setting a clock constraint with a timeout specified by the environment, and we proceed as if they were delayable transitions.

In our simulation, we implemented a mechanism to determine when the output action of a delayable transition is ready. There are two modes of operation: earliest and latest time responses. In the earliest mode, a delayable transition behaves like an eager transition, i.e. the transition is taken at the same time it is enabled. On the other hand, in the latest mode, the transition is taken in the latest possible time of the specified clock guard. For example, let's consider the transition from location `I3` to location `S5` of the BAS system (Figure 4.1). According to the specification, the output action `ReactivateAlarmSystem!` must be taken at any time between 0 and 50 ms. Hence, in the earliest mode the output action would be triggered instantaneously after the transition is enabled, while in the latest mode that would happen only in the 50th ms.

Validation of the implementations of the SUTs

The implementations of the SUTs must pass all their test cases in order to assure the correctness of their implementations. To validate a SUT, each test case was ran two times, one time with the SUT in the earliest mode of operation and another time with the latest mode to calculate the average execution time. Also, by running some small experiments, we found out that these two modes of SUT operation are way more likely to detect time-related faults created by mutant operators. All the SUTs passed their test cases, then we assumed the implementation to be correct for the purposes of the experiment.

Generation and killing of the mutants

We extended SYMBOLRT to generate model mutants [3]. We designed first-order mutation operators [29] that changes one element at a time of a transition to implement a class of faults. We implemented 15 operators:

Change source location Change the source location of a transition for another location of the model;

Original
```
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Mutant
```
from Print
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Change action Changes the action of a transition for another action of the model such that they are both input or output actions, they have the same number of action parameters, and the action parameter names are equal;

```
Original
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync ReturnChange!(value)
    do
    delayable
to Receive
```

Change arithmetic operator in data guard Change an arithmetic operator in the data guard of a transition for another arithmetic operator;

```
Original
from Verify
    [balance < refillValue AND value = refillValue + balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [balance < refillValue AND value = refillValue * balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Change relational operator in data guard Change a relational operator in the data guard of a transition for another relational operator;

```
Original
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [balance <= refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Negate data guard Negate the data guard of a transition;

```
Original
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [NOT (balance < refillValue AND value = refillValue - balance) AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Increase integer constant in clock guard Increase by 1 the value of an integer constant in the clock guard of the transition;

```
Original
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 6]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Decrease integer constant in clock guard Decrease by 1 the value of an integer constant in the clock guard of the transition;

```
Original
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

```
Mutant
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 4]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Change arithmetic operator in assignment Change an arithmetic operator in the assignment of a transition for another arithmetic operator;

```
Original
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue| clock := 0
    delayable
to Print
```

```
Mutant
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance * refillValue| clock := 0
    delayable
to Print
```

Change relational operator in assignment Change an relational operator in the assignment of a transition for another relational operator;

```
Original
from L1
    sync input?(a)
    do  positive = a > 0
    delayable
to L2;
```

Mutant
```
from L1
    sync input?(a)
    do  positive = a < 0
    delayable
to L2;
```

Add integer to expression in assignment Add 1 to the assignment of a transition;

Original
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue
    delayable
to Print
```

Mutant
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance * refillValue + 1
    delayable
to Print
```

Subtract integer to expression in assignment Subtract 1 to the assignment of a transition;

Original
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue
    delayable
to Print
```

Mutant
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance * refillValue - 1
    delayable
to Print
```

Negate boolean expression in assignment Negate a boolean expression in the assignment of a transition;

Original
```
from L1
    sync input?(a)
    do  positive = a > 0
    delayable
to L2;
```

Mutant
```
from L1
    sync input?(a)
    do  positive = NOT(a > 0)
    delayable
to L2;
```

Add clock reset Add a clock reset to a transition;

Original
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue
    delayable
to Print
```

Mutant
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue  | clock := 0
    delayable
to Print
```

Remove clock reset Remove a clock reset from a transition;

Original
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue | clock := 0
    delayable
to Print
```

Mutant
```
from Verify
    [balance = refillValue AND clock < 5]
    sync RefillCard!()
    do cardBallance := cardBallance + refillValue
    delayable
to Print
```

Change target location Change the target location of a transition for another location of the model;

Original
```
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Receive
```

Mutant
```
from Verify
    [balance < refillValue AND value = refillValue - balance AND clock < 5]
    sync MissingValue!(value)
    do
    delayable
to Verify
```

After the model mutants were generated, we compiled the model mutants to JAVA simulations as we did with the SUT — we call them concrete mutants. Note that is not the same as generating mutants from the SUT with a mutation tool such as μJAVA [50]. Mutating the SUT instead of the model means that the framework code would be changed too and generated mutants could be meaningless. By generating a model mutant and then compiling it to our JAVA simulation framework, the faulty behavior of the JAVA implementation is modeled in the mutation operator and it has a well-defined meaning. Finally, for each concrete mutant we run the concrete test cases against it and we recorded whether a test case killed the mutant, i.e. a test case failed.

4.2.2 Execution

Reduction of the test suites

We used the four heuristics (G, GE, GRE, HGS) and a random algorithm (Algorithm 1) to reduce the test suites according to each criterion. The random algorithm randomly selects a test case from the complete test suite `TS` and adds it to `TS′` until this suite satisfies the criterion. To check satisfaction, it is computed the requirements that must be covered by the criterion (e.g. for ALL-TRANSITIONS, requirements are all transitions of the specification). Variable `old` contains the set of all test requirements already covered by `TS′`, and variable `new` contains the set of all test requirements covered by `TS'` and the additional test case `tc`. If any new requirement is covered by `tc`, then this test case is added to `TS'`.

Algorithm 1 Select a subset of a test suite that satisfy a criterion

```
Input: $criterion$, a test adequacy criterion
Input: $TS$, a test suite satisfying $criterion$
Output: $TS'$, a subset of $TS$ satisfying $criterion$
 1: $TS' \leftarrow \emptyset$
 2: while $\neg satisfied(criterion, TS')$ do
 3:     $tc \leftarrow$ randomly select a test case from $TS$
 4:     $old \leftarrow covered(criterion, TS')$
 5:     $new \leftarrow covered(criterion, TS' \cup \{tc\})$
 6:     if $|new| > |old|$ then
 7:         $TS' \leftarrow TS' \cup \{tc\}$
 8:     end if
 9: end while
10: return $TS'$
```

For each combination of technique and criterion evaluated, we reduced the test suite 1,000 times in order to obtain 1,000 test suites for the combination, because one thousand replications is enough to enable statistical testing [12; 70].

4.3 Results and Analysis

In this section we present the results and analyze the data for each experimental unit: Refilling Machine in Section 4.3.1, Burglar Alarm System in Section 4.3.2, and Automated Speed Limiter in Section 4.3.3. For each experimental unit, we compare size, cost and effectiveness of the criteria and techniques evaluated in this experiment in order to test the hypotheses in Table 4.1. All experiment materials and data can be downloaded from the website `https://www.alanmoraes.com`.

4.3.1 Refilling Machine

Figure 4.4 shows the boxplots of the number of test cases of the reduced test suites by test adequacy criterion and Figure 4.5 the boxplots by test suite reduction technique. Figure 4.6 shows the boxplots of the execution time by criterion and Figure 4.7 by technique. Figure 4.8 shows the boxplots of the killed mutants by criterion and Figure 4.9 by technique. Appendix A.1 shows the corresponding descriptive summary tables.

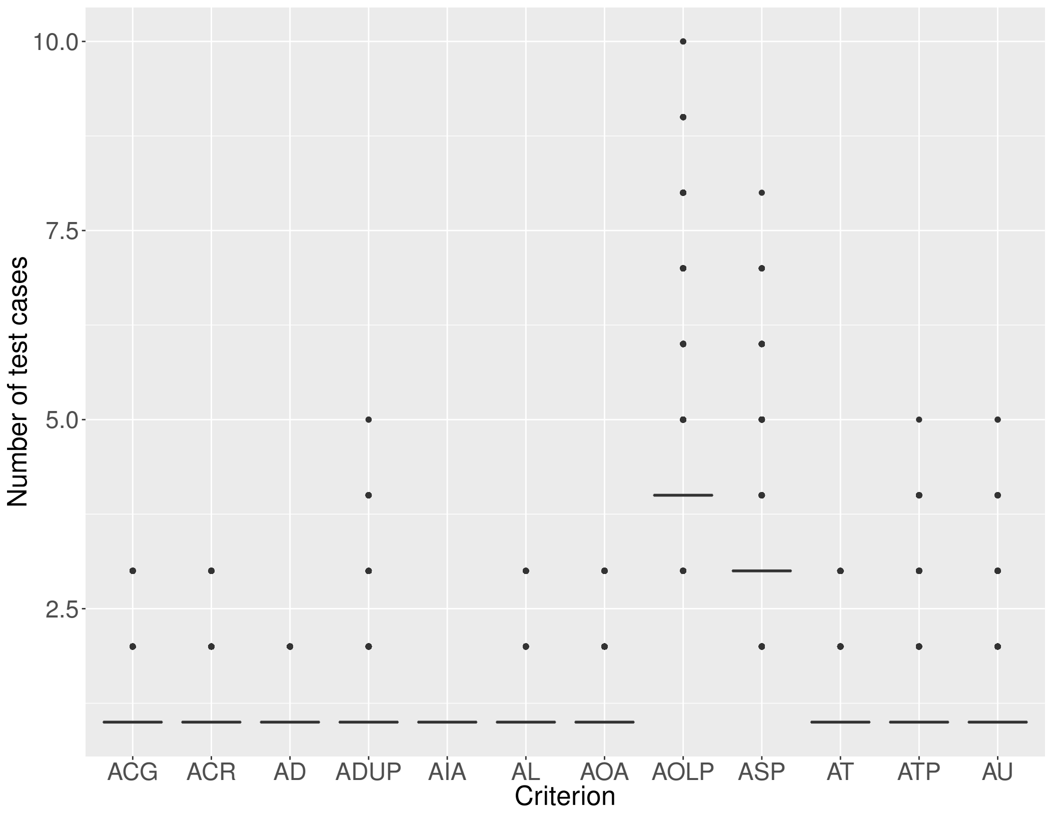


Figure 4.4: Refilling Machine — Boxplot of the number of test cases by criterion.

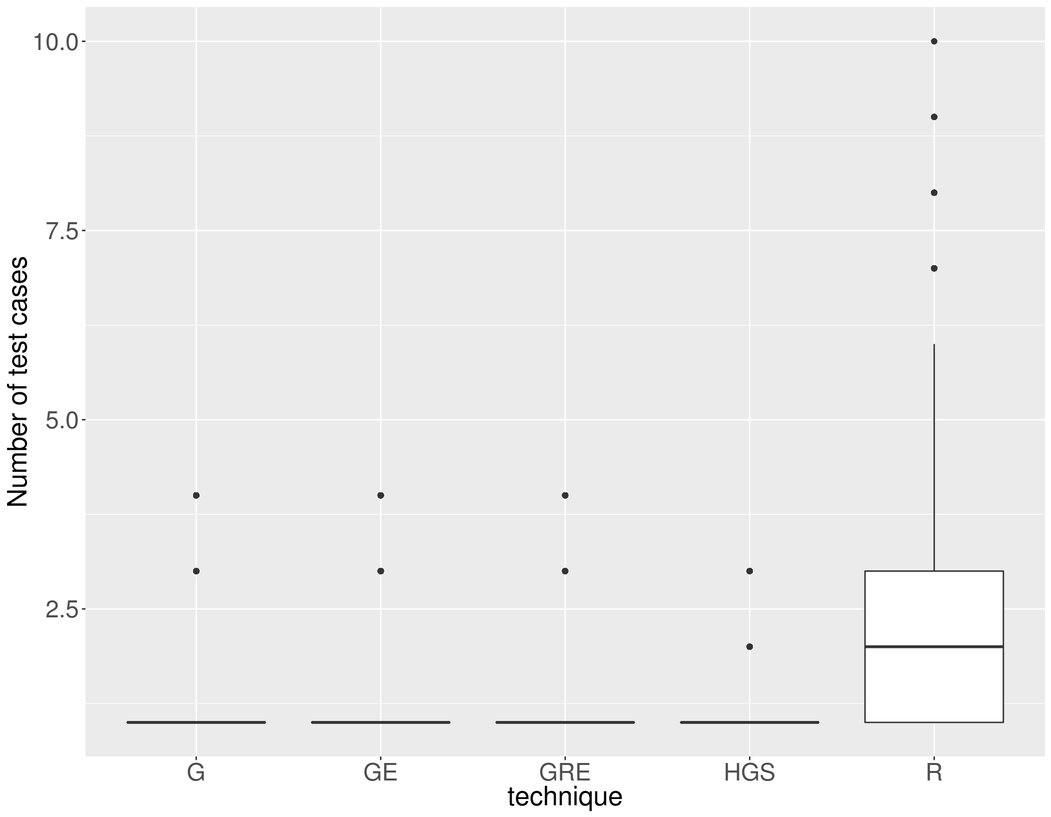


Figure 4.5: Refilling Machine — Boxplot of the number of test cases by technique.

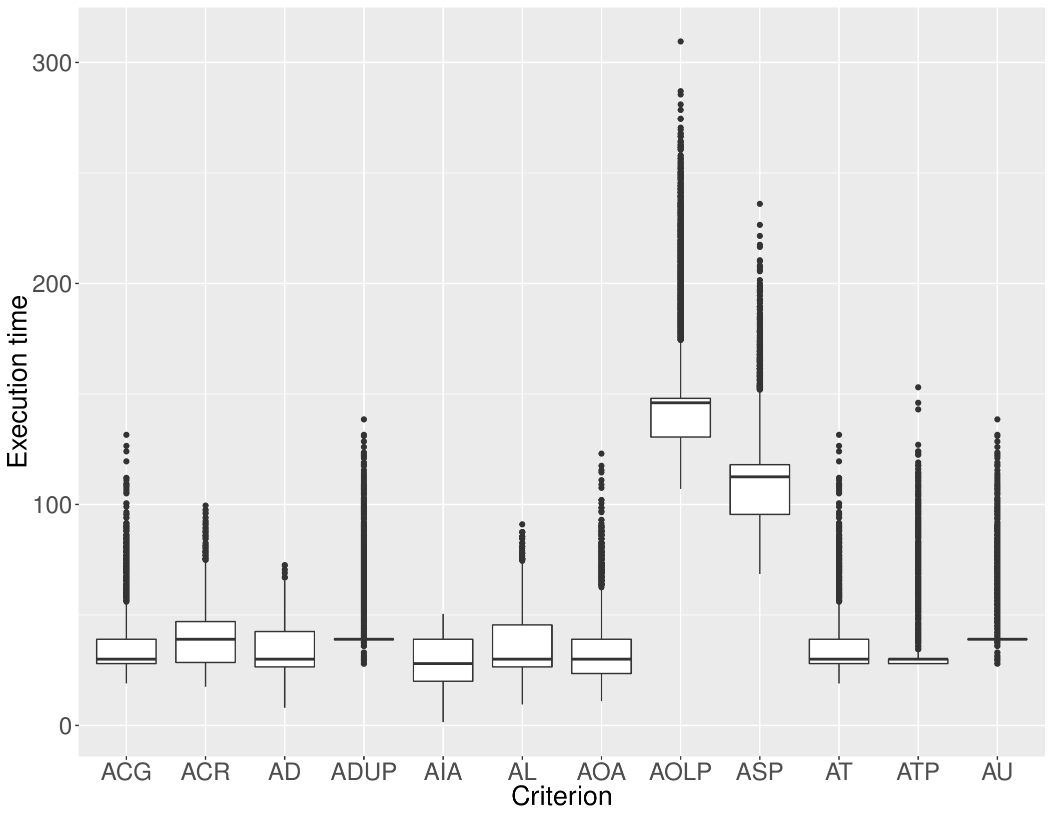


Figure 4.6: Refilling Machine — Boxplot of execution time by criterion.

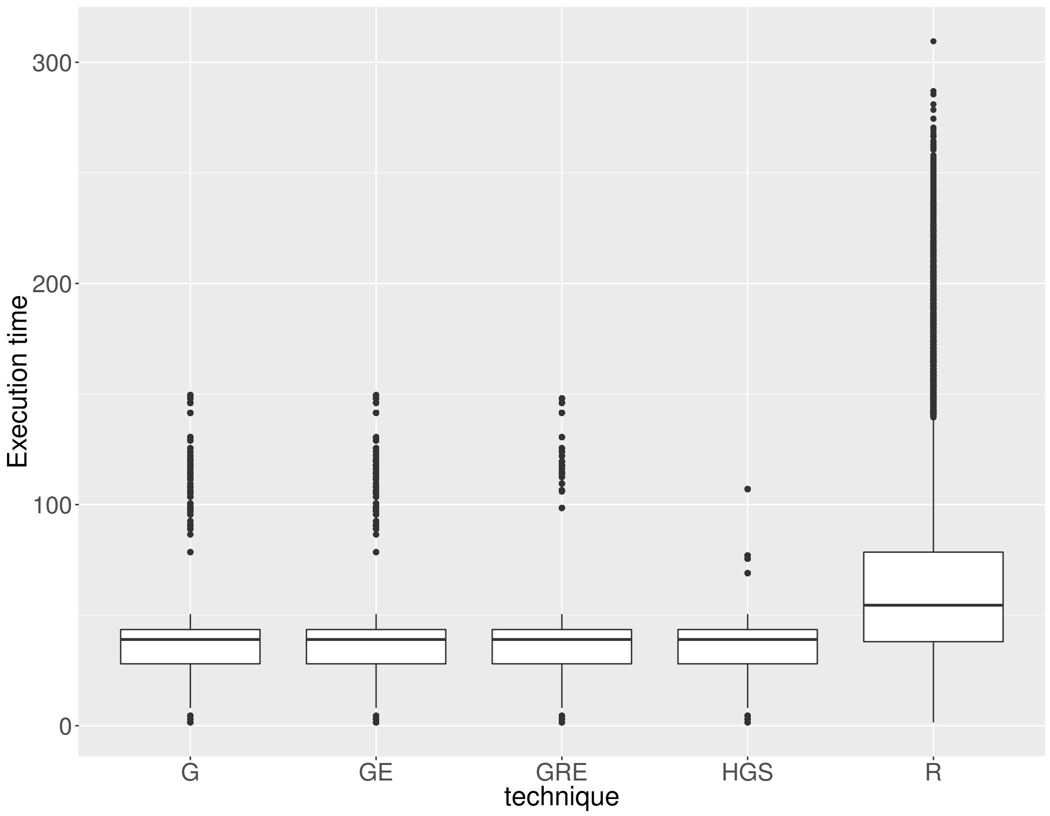


Figure 4.7: Refilling Machine — Boxplot of execution time by technique.

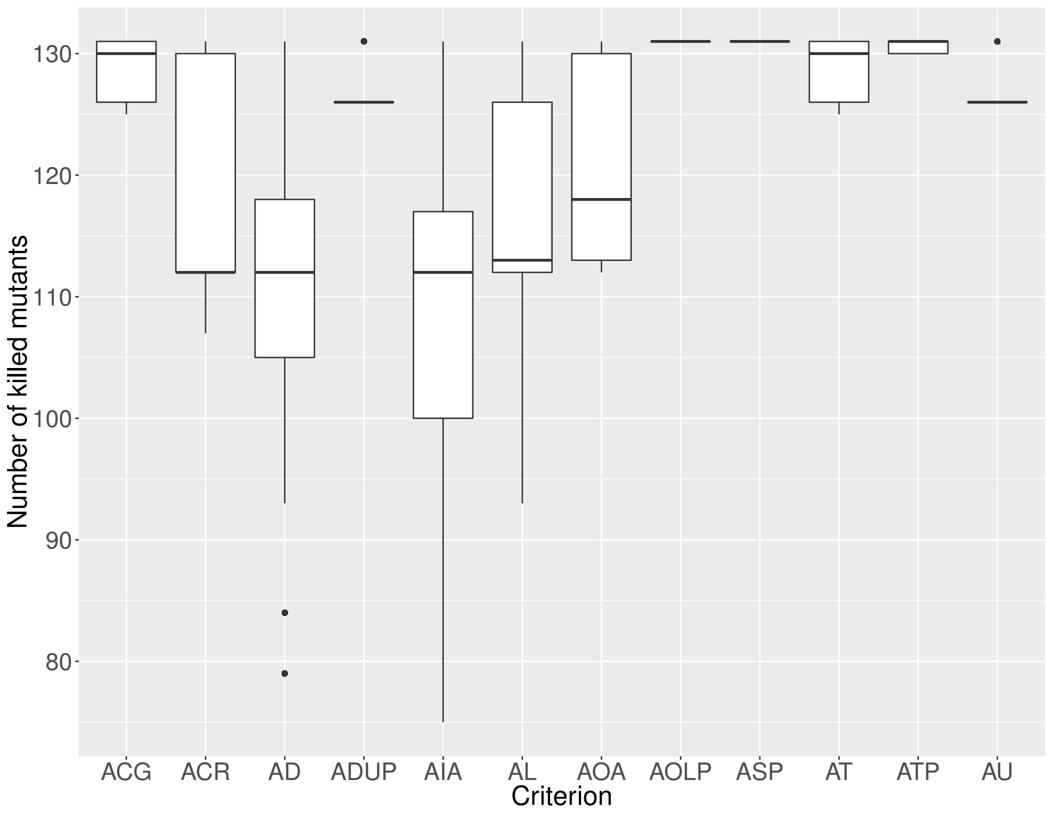


Figure 4.8: Refilling Machine — Boxplot of number of killed mutants by criterion.

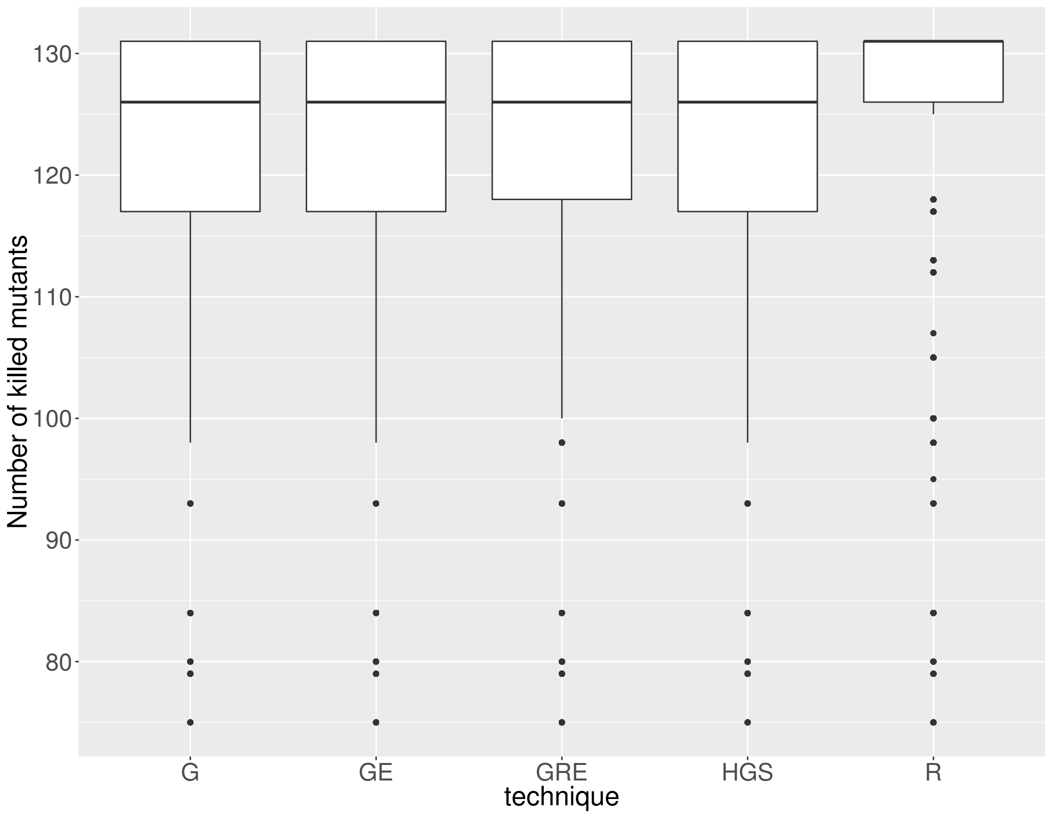


Figure 4.9: Refilling Machine — Boxplot of number of killed mutants by technique.

Table 4.1: Experiment hypotheses.

Null hypotheses	Alternative hypotheses
$H0_{\ test\_cases.criteria}$: there is no difference in the means of the criteria for the number of test cases of the reduced test suites.	$H1_{\ test\_cases.criteria}$: there is difference in the means of the criteria for the number of test cases of the reduced test suites.
$H0_{\ test\_cases.techniques}$: there is no difference in the means of the techniques for the number of test cases of the reduced test suites.	$H1_{\ test\_cases.techniques}$: there is difference in the means of the techniques for the number of test cases of the reduced test suites.
$H0_{\ time.criteria}$: there is no difference in the means of the criteria for the execution time of the reduced test suites.	$H1_{\ time.criteria}$: there is difference in the means of the criteria for the execution time of the reduced test suites.
$H0_{\ time.techniques}$: there is no difference in the means of the techniques for the execution time of the reduced test suites.	$H1_{\ time.techniques}$: there is difference in the means of the techniques for the execution time of the reduced test suites.
$H0_{\ mutants.criteria}$: there is no difference in the means of the criteria for the number of test cases of the reduced test suites.	$H1_{\ mutants.criteria}$: there is difference in the means of the criteria for the number of test cases of the reduced test suites.
$H0_{\ mutants.techniques}$: there is no difference in the means of the techniques for the number of test cases of the reduced test suites.	$H1_{\ mutants.techniques}$: there is difference in the means of the techniques for the number of test cases of the reduced test suites.

The data for the dependent variables number of test cases, execution time, and number of killed mutants were not normally distributed for the evaluated factors, as assessed by visual inspection of Normal Q-Q Plots. For all dependent variables, distributions were not similar for all groups as assessed by Levene's test for equality of variances, $p < 0.05$. Although the data was non-normal and distributions were not all similar, we decided to run two-way ANOVAs because the test is quite robust to violations of normality and homoscedasticity assumptions when group sizes are equal [53]. Thus we run a two-way ANOVA for each of the three dependent variables in order to determine whether there are differences in the means of the dependent variables. Figures 4.10, 4.11 and 4.12 present the results of the ANOVAs for the number of test cases, execution time and number of killed mutants respectively.

For the number of test cases (Tables A.1 and A.2 in Appendix), there was a statistically

Figure 4.10: Refilling Machine — Two-way ANOVA for number of test cases

```
Analysis of Variance Table

Response: test_cases
             Df Sum Sq Mean Sq F value    Pr(>F)
technique     4  10098  2524.5   14197 < 2.2e-16 ***
criterion    11  56635  5148.6   28955 < 2.2e-16 ***
Residuals 59984  10666     0.2
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.11: Refilling Machine — Two-way ANOVA for execution time

```
Analysis of Variance Table

Response: time
             Df   Sum Sq Mean Sq F value    Pr(>F)
technique     4  5058127 1264532  6368.7 < 2.2e-16 ***
criterion    11 73790557 6708232 33785.6 < 2.2e-16 ***
Residuals 59984 11909994     199
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.12: Refilling Machine — Two-way ANOVA number of killed mutants

```
Analysis of Variance Table

Response: mutants
             Df  Sum Sq Mean Sq F value    Pr(>F)
technique     4   45976   11494  213.17 < 2.2e-16 ***
criterion    11 3687107  335192 6216.70 < 2.2e-16 ***
Residuals 59984 3234214      54
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

significant main effect for the criteria, $F(11, 59984) = 28955.0$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 14197$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.technique}$.

For execution time (Tables A.3 and A.4 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59940) = 33785.6$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59940) = 6368.7$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.technique}$. ALL-ONE-LOOP-PATHS was the criterion that took longer to execute on average ($mean = 146.2$) and ALL-INPUT-ACTIONS was the fastest ($mean = 223.3$), while RANDOM was the technique that took longer to execute on average (mean = 69.31) and HGS was fastest (mean = 42.53). According to Tukey multiple comparisons of means post-hoc test, we sort the criteria by cost from cheapest to most expensive to execute as follows: ALL-INPUT-ACTIONS < ALL-OUTPUT-ACTIONS < ALL-DEFS < ALL-LOCATIONS < ALL-TRANSITION-PAIRS < (ALL-CLOCK-GUARDS = ALL-TRANSITIONS) < ALL-CLOCK-RESETS < (ALL-USES = ALL-DU-PATHS) < ALL-SIMPLE-PATHS < ALL-ONE-LOOP-PATHS. Likewise, we sort the techniques by cost from cheapest to most expensive as follows: HGS < (G = GE = GRE) < R.

For number of killed mutants (Tables A.5 and A.6 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59940) = 6216.70$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ mutants.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59940) = 213.1$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ mutants.technique}$. ALL-ONE-LOOP-PATHS and ALL-SIMPLE-PATHS were the most effective criterion ($mean = 131.0$) and ALL-INPUT-ACTIONS was the least ($mean = 107.3$), while R was the technique with best results (mean = 125.2) and the other techniques were tied. According to Tukey multiple comparisons of means post-hoc test, we sort effectiveness, from worst to best, as follows: ALL-INPUT-ACTIONS < ALL-DEFS < ALL-LOCATIONS < ALL-CLOCK-RESETS < ALL-OUTPUT-ACTIONS < (ALL-USES = ALL-DU-PATHS) < (ALL-CLOCK-GUARDS = ALL-TRANSITIONS) < ALL-TRANSITION-PAIRS < ALL-SIMPLE-PATHS < ALL-ONE-LOOP-PATHS. Likewise, we sort the techniques by effectiveness, from worst to most best, as follows: (HGS = G = GE = GRE) < R.

4.3.2 Burglar Alarm System

Figure 4.13 shows the boxplots of the number of test cases of the reduced test suites by test adequacy criterion and Figure 4.14 shows the boxplots by test suite reduction technique. Figure 4.15 shows the boxplots of the execution time by criterion and Figure 4.16 by technique. Figure 4.17 shows the boxplots of the killed mutants by criterion and Figure 4.18 by technique. Appendix A.2 shows the corresponding descriptive summary tables.

The data for the dependent variables number of test cases, execution time, and number of killed mutants were not normally distributed for the evaluated factors, as assessed by visual inspection of Normal Q-Q Plots. For all dependent variables, distributions were not similar for all groups, as assessed by Levene's test for equality of variances, $p < 0.05$. Although the data was non-normal and distributions were not all similar, we decided to run three two-way ANOVA to determine whether there are differences in the distributions of the three dependent variables for the two independent variables evaluated because ANOVA is quite robust to violations of normality when group sizes are equal [53]. Figures 4.19, 4.20 and 4.21 present the results for the number of test cases, execution time and number of killed mutants respectively.

For the number of test cases (Tables A.7 and A.8 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59984) = 258197.6$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 7177.2$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.technique}$. ALL-ONE-LOOP-PATHS was the criterion that selected the most test cases on average ($mean = 22.170$) and ALL-INPUT-ACTIONS selected the fewest ($mean = 1.021$), while RANDOM was the technique that selected the most ($mean = 8.065$) and all other techniques selected 6.667 test cases on average.

For execution time (Tables A.9 and A.10 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59984) = 155693$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 3378$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.technique}$. ALL-ONE-LOOP-PATHS was the criterion that took longer to execute on average ($mean = 60550$) and ALL-INPUT-ACTIONS was the fastest ($mean = 2559$), while

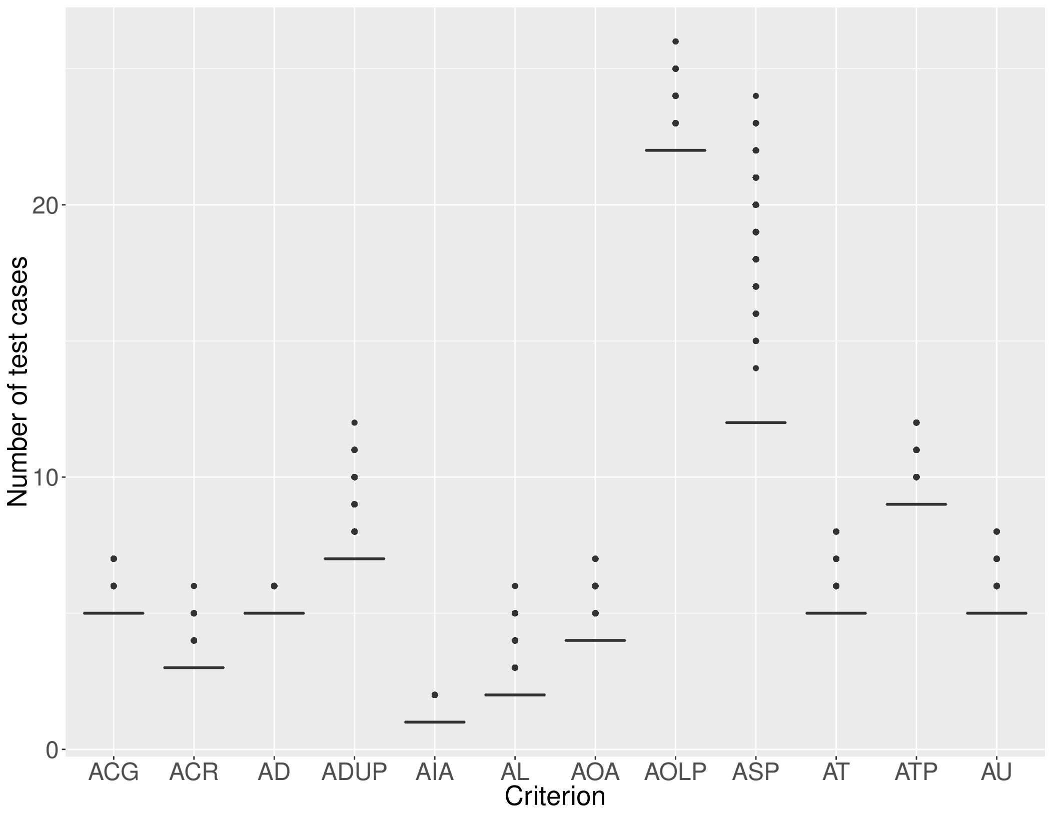


Figure 4.13: Burglar Alarm System — Boxplot of the number of test cases by criterion.

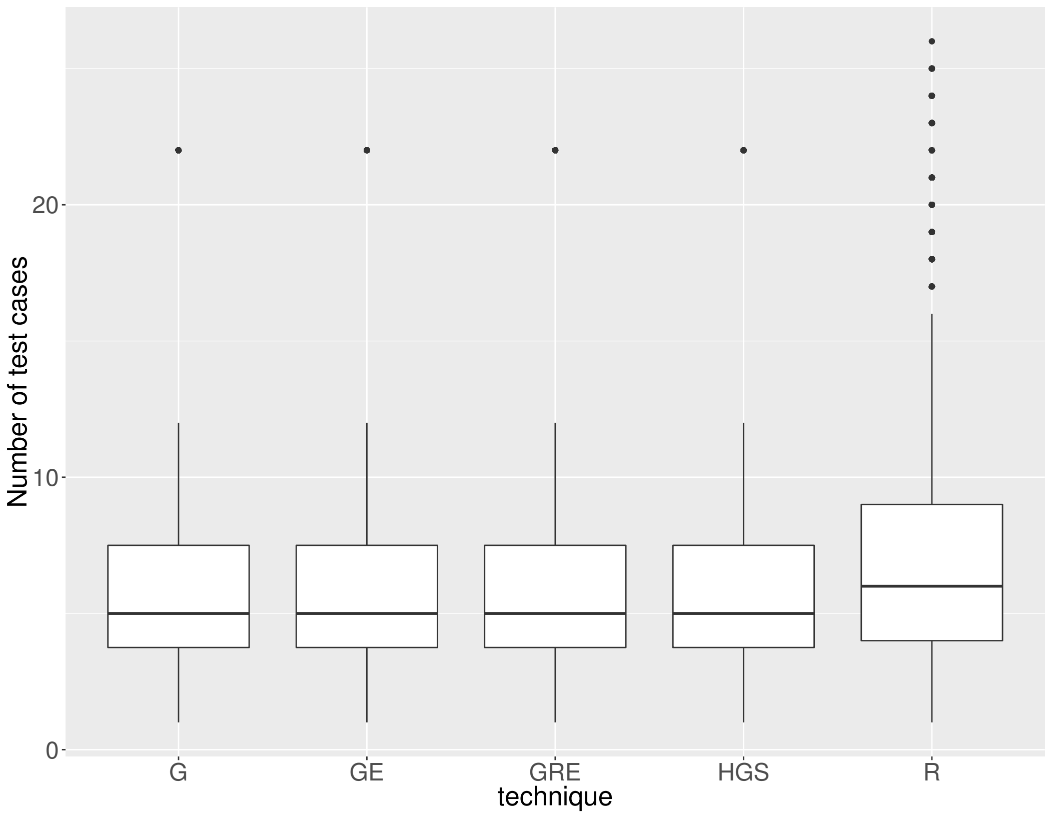


Figure 4.14: Burglar Alarm System — Boxplot of the number of test cases by technique.

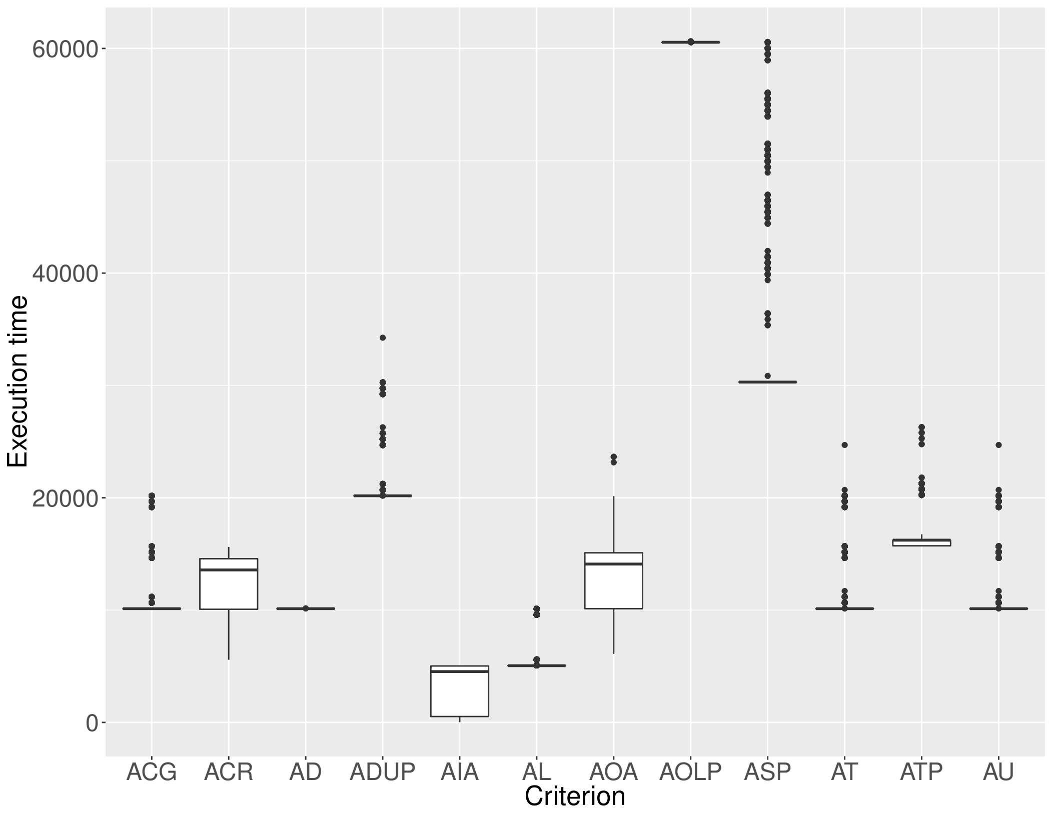


Figure 4.15: Burglar Alarm System — Boxplot of execution time by criterion.

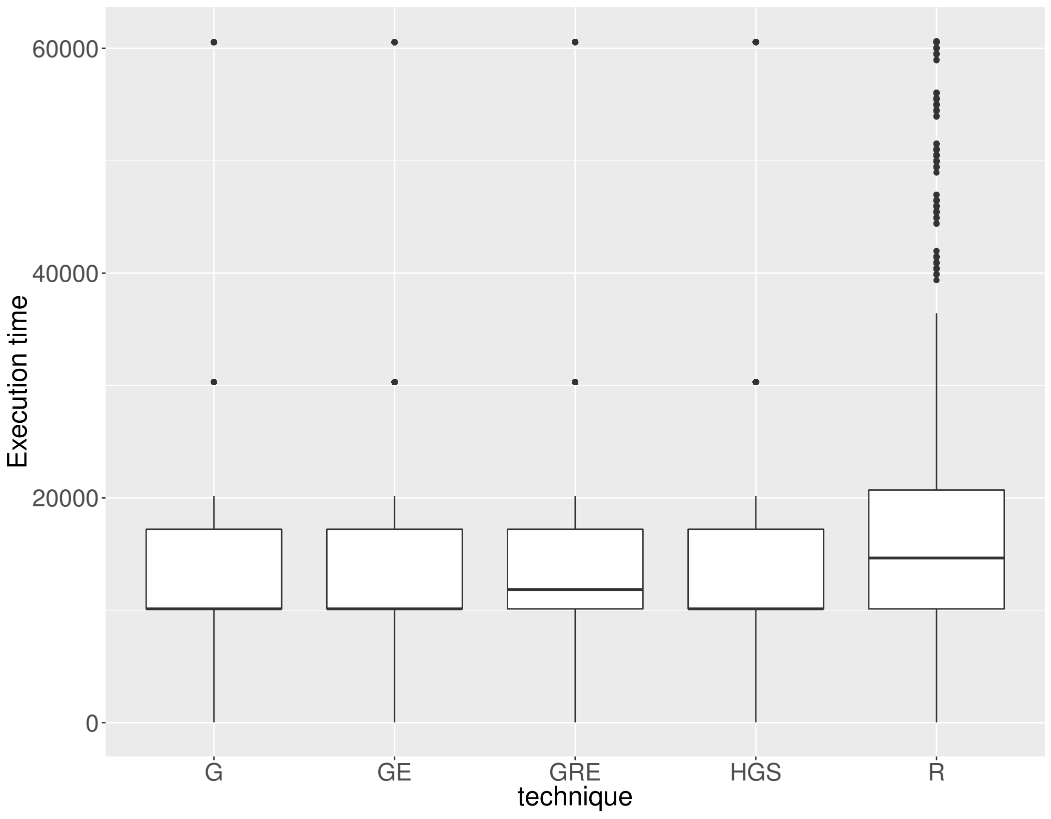


Figure 4.16: Burglar Alarm System — Boxplot of execution time by technique.

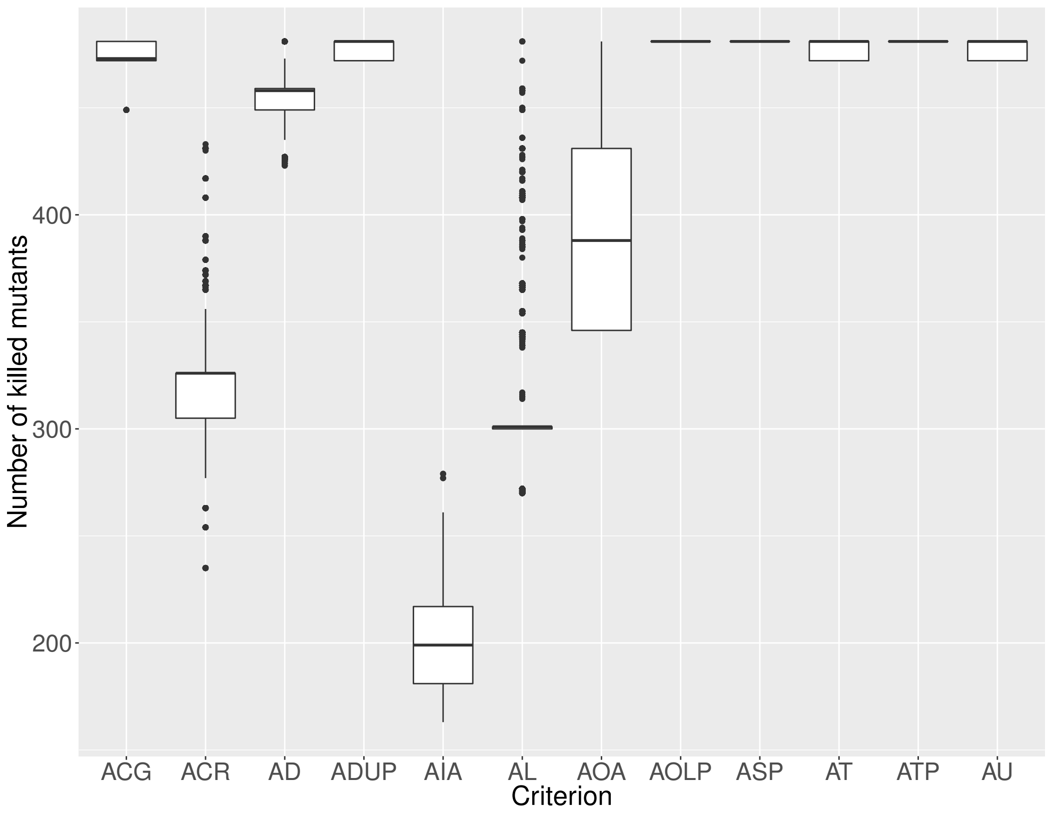


Figure 4.17: Burglar Alarm System — Boxplot of number of killed mutants by criterion.

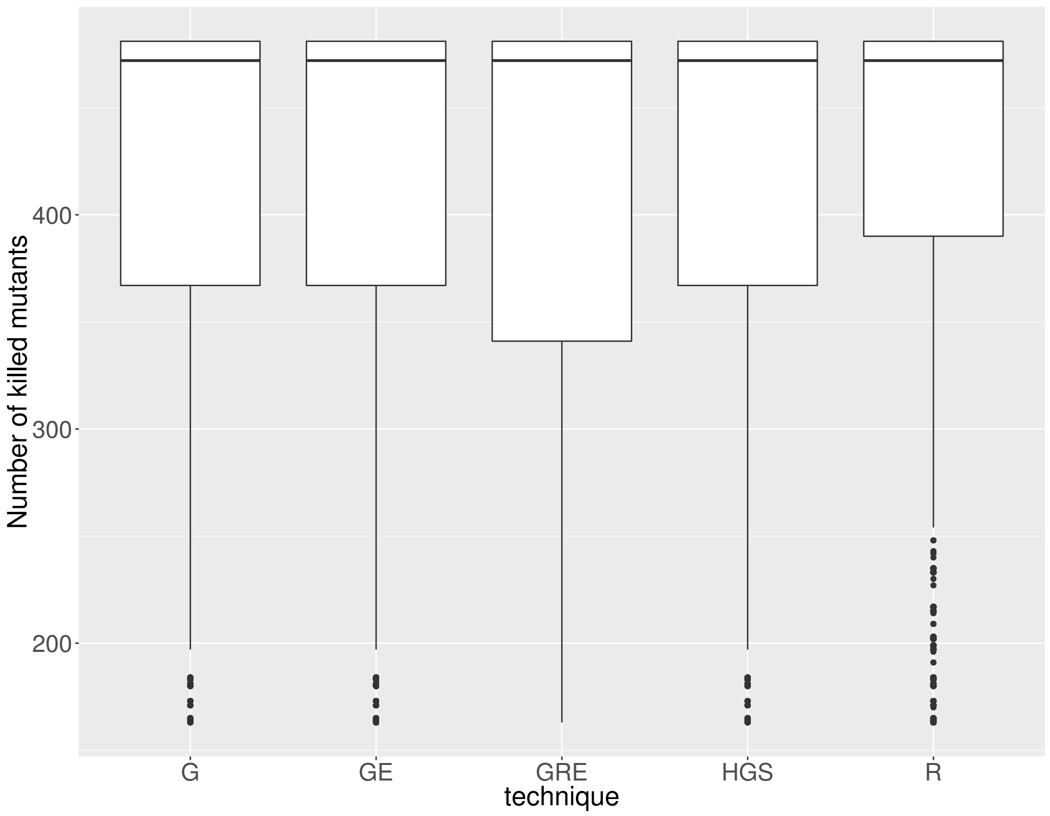


Figure 4.18: Burglar Alarm System — Boxplot of number of killed mutants by technique.

Figure 4.19: Burglar Alarm System — Two-way ANOVA for number of test cases

```
Analysis of Variance Table

Response: test_cases
             Df  Sum Sq Mean Sq  F value    Pr(>F)
technique     4   18773    4693   7177.2 < 2.2e-16 ***
criterion    11 1857260  168842 258197.6 < 2.2e-16 ***
Residuals 59984   39225       1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.20: Burglar Alarm System — Two-way ANOVA for execution time

```
Analysis of Variance Table

Response: time
             Df     Sum Sq    Mean Sq F value    Pr(>F)
technique     4 1.0870e+11 2.7175e+10    3378 < 2.2e-16 ***
criterion    11 1.3778e+13 1.2525e+12  155693 < 2.2e-16 ***
Residuals 59984 4.8256e+11 8.0448e+06
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.21: Burglar Alarm System — Two-way Anova for number of killed mutants

```
Analysis of Variance Table

Response: mutants
             Df    Sum Sq  Mean Sq   F value    Pr(>F)
technique     4   1085890   271473    903.82 < 2.2e-16 ***
criterion    11 479389109 43580828 145095.31 < 2.2e-16 ***
Residuals 59984  18016795      300
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

RANDOM was the technique that took longer to execute on average ($mean = 20000$), and G, GE and HGS were the fastest (mean = 16540). According to Tukey multiple comparisons of means post-hoc test, we sort the criteria by cost from cheapest to most expensive to execute as follows: ALL-INPUT-ACTIONS < ALL-LOCATIONS < ALL-DEFS < (ALL-TRANSITIONS = ALL-USES = ALL-CLOCK-GUARDS) < ALL-CLOCK-RESETS < ALL-OUTPUT-ACTIONS < ALL-TRANSITION-PAIRS < ALL-DU-PATHS < ALL-SIMPLE-PATHS < ALL-ONE-LOOP-PATHS. Likewise, we sort the techniques by cost from cheapest to most expensive as follows: (G = GE = HGS) < GRE < R.

For number of killed mutants (Tables A.11 and A.12 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59984) = 145095.3$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 903.8$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.technique}$. ALL-ONE-LOOP-PATHS, ALL-SIMPLE-PATHS, and ALL-TRANSITION-PAIRS were the most effective criterion ($mean = 481.0$) and ALL-INPUT-ACTIONS was the least ($mean = 200.2$), while RANDOM was the technique with best results (mean = 426.7) and the other techniques were tied. According to Tukey multiple comparisons of means post-hoc test, we sort effectiveness, from worst to best, as follows: ALL-INPUT-ACTIONS < ALL-LOCATIONS < ALL-CLOCK-RESETS < ALL-OUTPUT-ACTIONS < ALL-DEFS < (ALL-TRANSITIONS = ALL-USES = ALL-DU-PATHS = ALL-CLOCK-GUARDS) < (ALL-TRANSITION-PAIRS = ALL-SIMPLE-PATHS = ALL-ONE-LOOP-PATHS) Likewise, we sort the techniques by effectiveness, from worst to most best, as follows: (G = GE = GRE = HGS) < R.

4.3.3 Automated Speed Limiter

Figure 4.22 shows the boxplots of the number of test cases of the reduced test suites by test adequacy criterion and Figure 4.23 shows the boxplots by test suite reduction technique. Figure 4.26 shows the boxplots of the killed mutants by criterion and Figure 4.27 by technique. Figure 4.24 shows the boxplots of the execution time by criterion and Figure 4.25 by technique. Appendix A.3 shows the corresponding descriptive summary tables.

The data for the dependent variables number of test cases, execution time, and number of killed mutants were not normally distributed for the evaluated factors, as assessed by visual

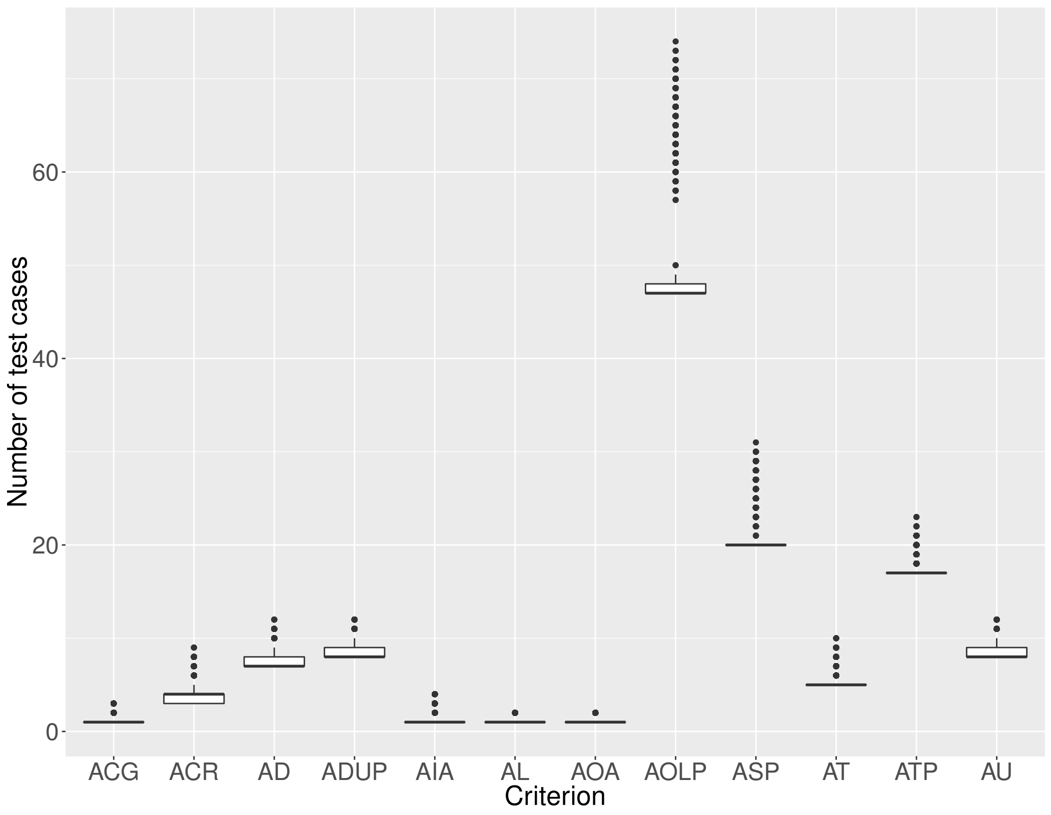


Figure 4.22: Automated Speed Limiter — Boxplot of the number of test cases by criterion.

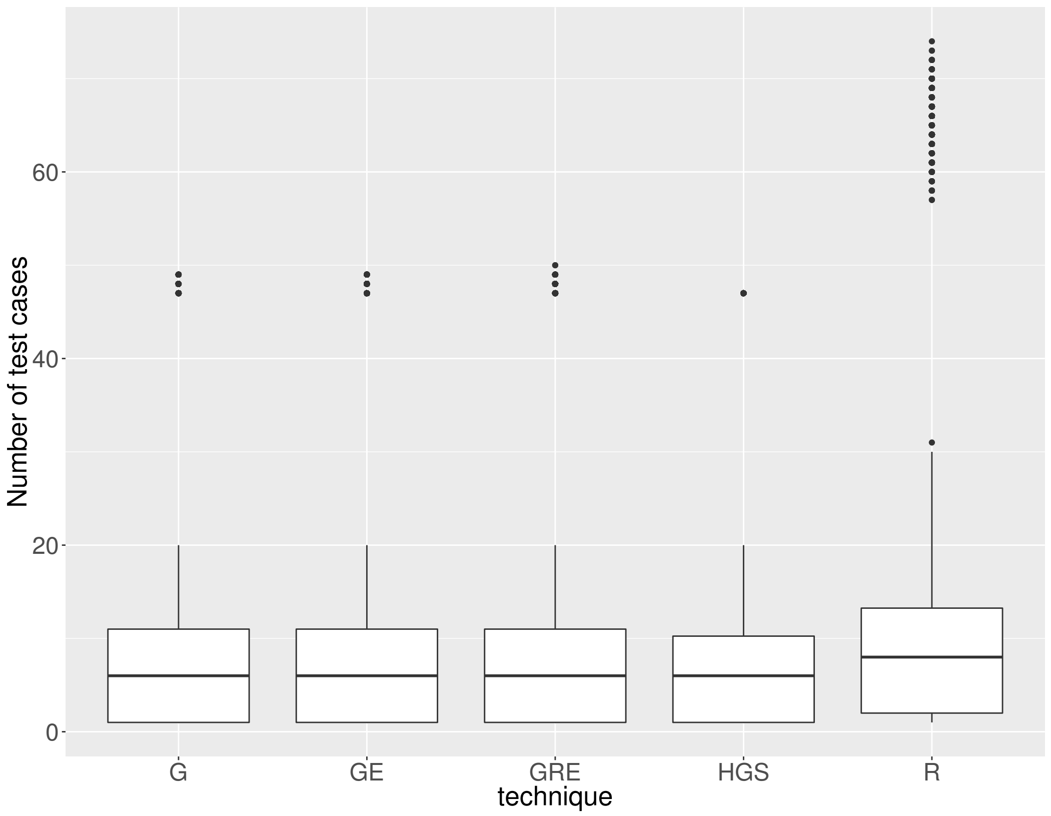


Figure 4.23: Automated Speed Limiter — Boxplot of the number of test cases by technique.

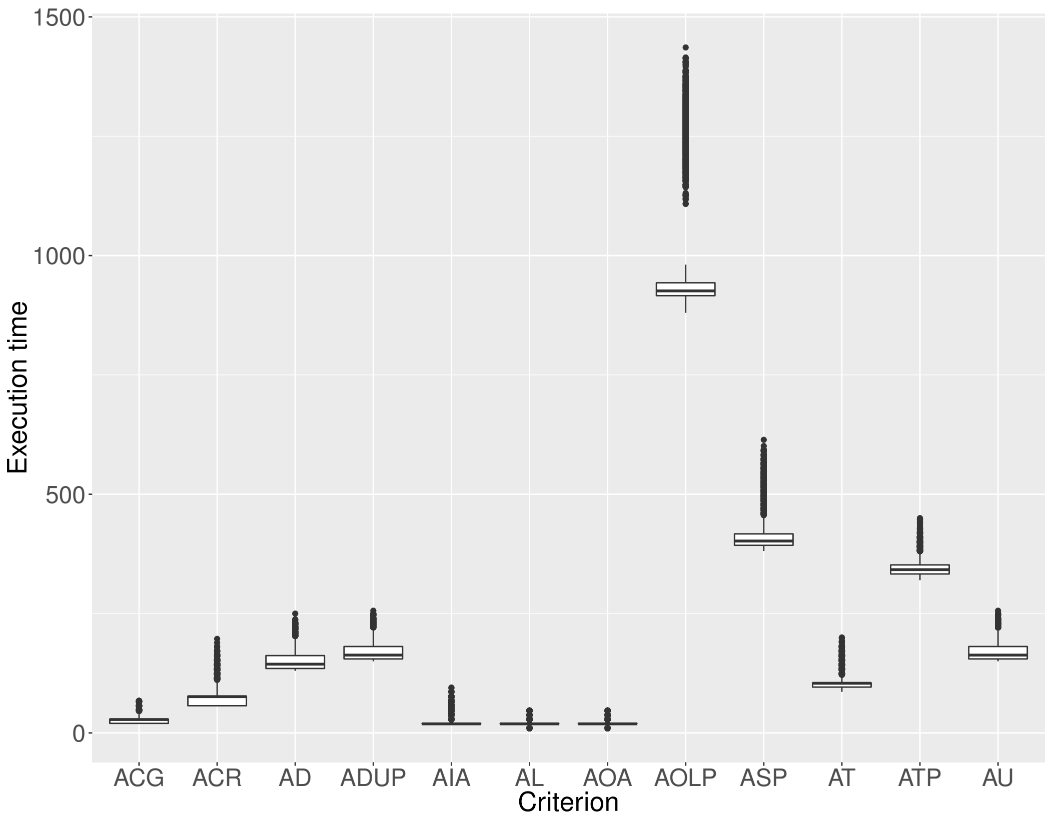


Figure 4.24: Automated Speed Limiter — Boxplot of execution time by criterion.

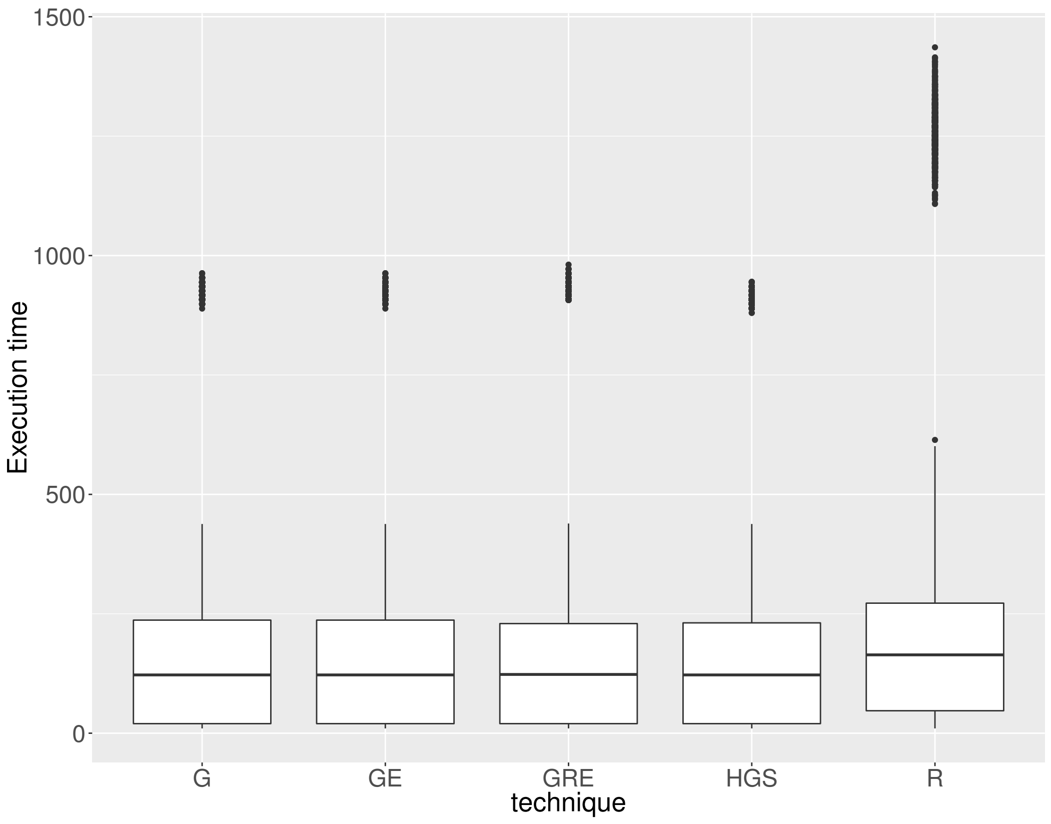


Figure 4.25: Automated Speed Limiter — Boxplot of execution time by technique.

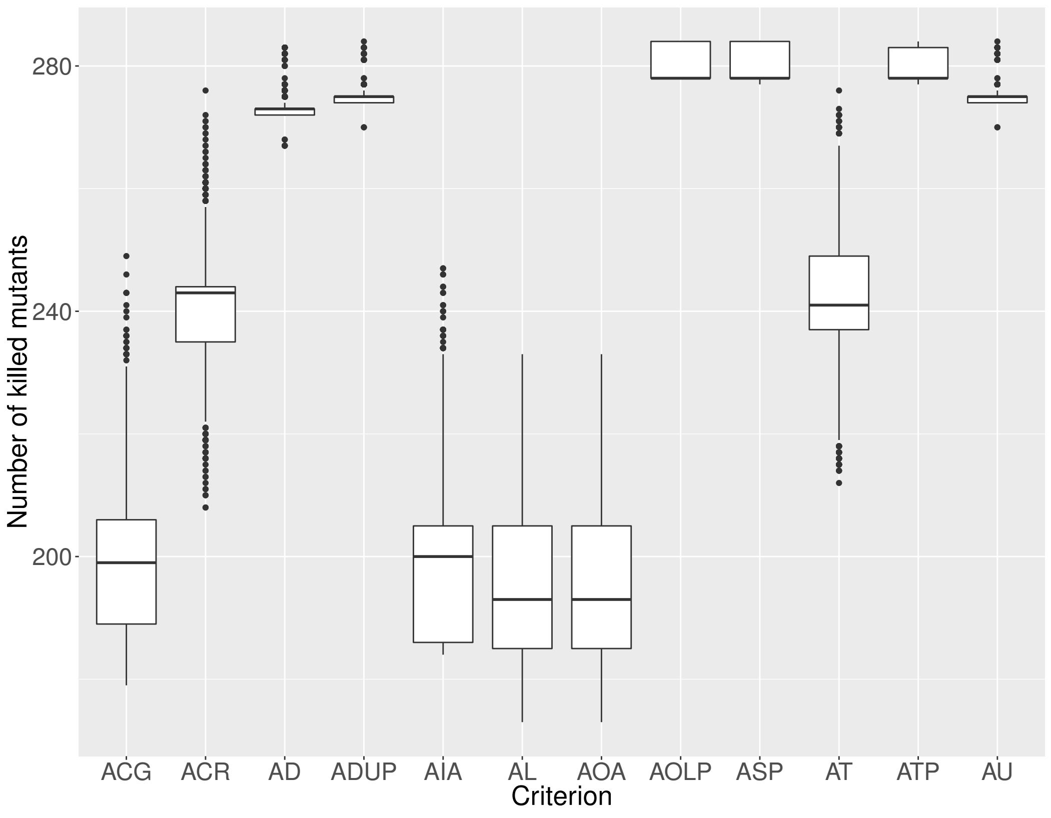


Figure 4.26: Automated Speed Limiter — Boxplot of number of killed mutants by criterion.

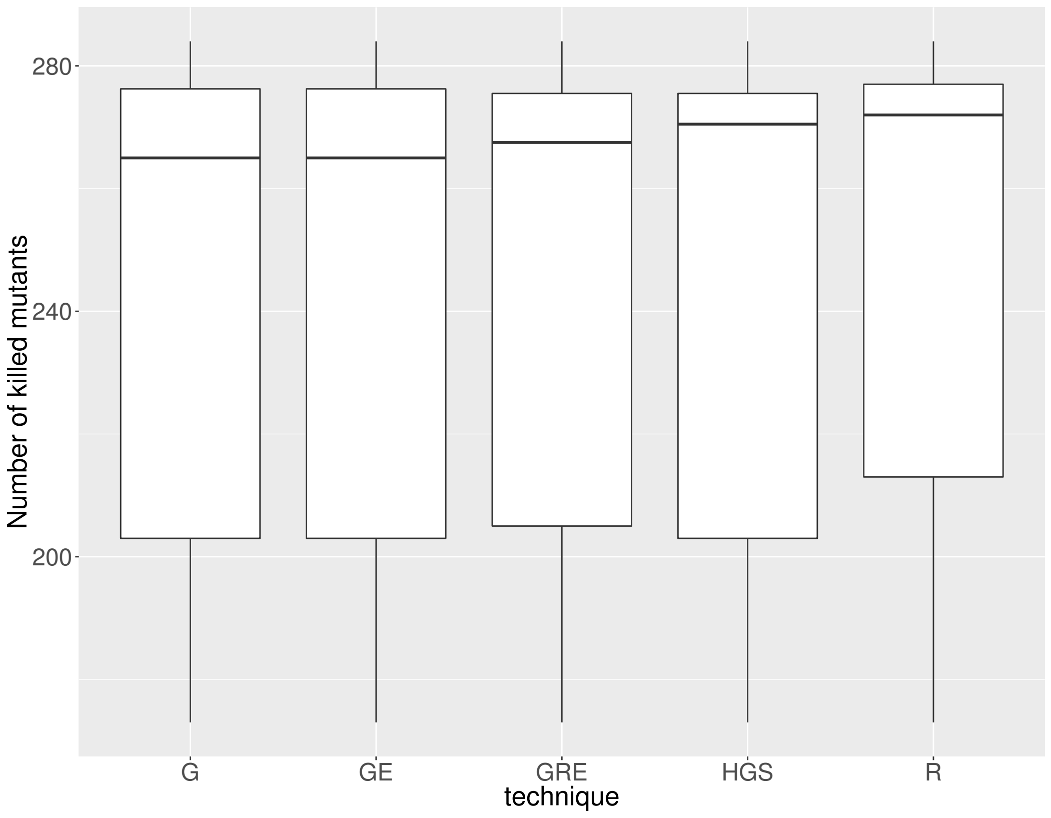


Figure 4.27: Automated Speed Limiter — Boxplot of number of killed mutants by technique.

inspection of Normal Q-Q Plots. For all dependent variables, distributions were not similar for all groups, as assessed by Levene's test for equality of variances, $p < 0.05$. Although the data was non-normal and distributions were not all similar, we decided to run a two-way ANOVA to determine whether there are differences in the distributions for each of the three dependent variables for the two independent variables evaluated because ANOVA is quite robust to violations of normality when group sizes are equal [53]. Figures 4.28, 4.29 and 4.30 present the results for the number of test cases, execution time and number of killed mutants respectively.

For the number of test cases (Tables A.13 and A.14 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59984) = 275140.9$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 6923.2$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ test\_cases.technique}$. ALL-ONE-LOOP-PATHS was the criterion that selected the most test cases on average ($mean = 50.840$), and ALL-OUTPUT-ACTIONS and ALL-LOCATIONS selected the fewest ($mean = 1.039$), while RANDOM was the technique that selected the most ($mean = 13.270$) and all other techniques tied.

For execution time (Tables A.15 and A.16 in Appendix), there was a statistically significant main effect for the criteria, $F(11, 59984) = 269750.9$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 6570.8$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ time.technique}$. ALL-ONE-LOOP-PATHS was the criterion that took longer to execute on average ($mean = 988.60$), and ALL-OUTPUT-ACTIONS and ALL-LOCATIONS were the shortest ($mean = 20.75$), while RANDOM was the technique that took longer to execute on average ($mean = 260.7$), and HGS was the fastest ($mean = 195.3$). According to Tukey multiple comparisons of means post-hoc test, we sort the criteria by cost from cheapest to most expensive to execute as follows: (ALL-OUTPUT-ACTIONS = ALL-LOCATIONS) < ALL-INPUT-ACTIONS < ALL-CLOCK-GUARDS < ALL-CLOCK-RESETS < ALL-TRANSITIONS < ALL-DEFS < (ALL-USES = ALL-DU-PATHS) < ALL-TRANSITION-PAIRS < ALL-SIMPLE-PATHS < ALL-ONE-LOOP-PATHS. Likewise, we sort the techniques by cost from cheapest to most expensive as follows: HGS < (G = GE = GRE) < R.

For number of killed mutants (Tables A.17 and A.18 in Appendix), there was a statis-

Figure 4.28: Automated Speed Limiter — Two-way ANOVA for number of test cases

```
Analysis of Variance Table

Response: test_cases
             Df   Sum Sq Mean Sq  F value    Pr(>F)
technique     4   101610   25403   6923.2 < 2.2e-16 ***
criterion    11 11104993 1009545 275140.9 < 2.2e-16 ***
Residuals 59984   220093       4
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.29: Automated Speed Limiter — Two-way ANOVA for execution time

```
Analysis of Variance Table

Response: time
             Df     Sum Sq   Mean Sq  F value    Pr(>F)
technique     4   37301652   9325413   6570.8 < 2.2e-16 ***
criterion    11 4211165764 382833251 269750.9 < 2.2e-16 ***
Residuals 59984   85129913      1419
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.30: Automated Speed Limiter — Two-way Anova for number of killed mutants

```
Analysis of Variance Table

Response: mutants
             Df   Sum Sq Mean Sq   F value    Pr(>F)
technique     4   206396   51599    953.44 < 2.2e-16 ***
criterion    11 78999049 7181732 132704.02 < 2.2e-16 ***
Residuals 59984  3246239      54
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

tically significant main effect for the criteria, $F(11, 59984) = 132704.0$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ mutants.criteria}$. The difference in the means is also statistically significant for the techniques, $F(4, 59984) = 953.4$, $p < 0.01$, hence *we reject the null hypothesis* $H_{0\ mutants.technique}$. ALL-ONE-LOOP-PATHS, was the most effective criterion ($mean = 280.4$), and ALL-INPUT-ACTIONS and ALL-LOCATIONS were the worst ($mean = 193.9$), while RANDOM was the technique with best results (mean = 247.8) and the other techniques were tied. According to Tukey multiple comparisons of means post-hoc test, we sort the criteria by cost from cheapest to most expensive to execute as follows: (ALL-OUTPUT-ACTIONS = ALL-LOCATIONS) < ALL-CLOCK-GUARDS < ALL-INPUT-ACTIONS < ALL-CLOCK-RESETS < ALL-TRANSITIONS < ALL-DEFS < (ALL-USES = ALL-DU-PATHS) < ALL-TRANSITION-PAIRS < ALL-SIMPLE-PATHS < ALL-ONE-LOOP-PATHS. Likewise, we sort the techniques by effectiveness, from worst to most best, as follows: (G = GE = GRE = HGS) < R.

4.4 Answers the Study Questions

4.4.1 SQ1: What test adequacy criterion has the best cost-effectiveness?

Empirical results showed that criteria closer to the top of the hierarchy selected larger test suites which were more effective but also more expensive to execute. However, some criteria were more cost-effective than others. Figures 4.31–4.33 shows scatterplots of cost vs. effectiveness for all evaluated criteria in the experiment. The sweet spot of cost-effectiveness is the top-left corner, i.e. a high number of killed mutants and low execution time.

In the SUB experiment (Figure 4.31), the best performing test adequacy criteria were ALL-ONE-LOOP-PATHS (AOLP) and ALL-SIMPLE-PATHS (ASP). All reduced test suites obtained by using AOLP and ASP were able to kill 100% of mutants that the full test suite was able to kill, but AOLP reduced the execution cost on average to 13.7% of the total time of the full test suite while ASP further reduced it to 10.2%. The next group is composed of ALL-TRANSITION-PAIRS (ATP), ALL-CLOCK-GUARDS (ACG), ALL-TRANSITIONS (AT), ALL-DU-PATHS (ADUP), and ALL-USES (AU) criteria. Although ATP, ACG and AT did not achieve 100% of mutation score on average (99.7%, 98.6%, and 98.6% respectively), they reduced even more the average execution time (3.3%, 3.5%, and 3.5% respectively). ADUP and AU obtained the same effectiveness and cost execution, 96.8% and 4.2% respectively, hence they had worse cost-effectiveness than previous criteria. Finally, ALL-OUTPUT-ACTIONS, ALL-CLOCK-RESETS, ALL-LOCATIONS, ALL-DEFS, and ALL-INPUT-ACTIONS form a group with worst performance, all of them with less than 93% of the effectiveness of the full test suite.

In the BAS experiment (Figure 4.32), the best performing group of test adequacy criteria is composed of ALL-ONE-LOOP-PATHS (AOLP), ALL-SIMPLE-PATHS (ASP), and ALL-TRANSITION-PAIRS (ATP). While all reduced test suites obtained by using these criteria were able to kill 100% of mutants that the full test suite was able to, reductions in execution times were greater in ATP (18.2% of total time) than in ASP (37.8%) and AOLP (66.6%). This group is closely followed by ALL-DU-PATHS (ADUP), ALL-USES, ALL-TRANSITIONS and ALL-CLOCK-GUARDS criteria, which are tied with mutation score of 99.0%, however ALL-DU-PATHS had almost double of the average execution time compared to other criteria in this group (23.0% vs. 12.0%). The worst performing group is composed of ALL-DEFS (AD), ALL-OUTPUT-ACTIONS (AOA), ALL-CLOCK-RESETS (ACR), ALL-LOCATIONS, and ALL-INPUT-ACTIONS criteria; their average mutation score was below 95%.

In the ASL experiment (Figure 4.33), the best performing group of test adequacy criteria is composed of ALL-ONE-LOOP-PATHS (AOLP), ALL-SIMPLE-PATHS (ASP), and ALL-TRANSITION-PAIRS (ATP) criteria with mutation scores of 98.7%, 98.4% and 98.4%, and execution times of 14.3%, 6.1% and 5.0% of the full test suite, respectively. The second group of criteria is composed of ALL-DU-PATHS (ADUP), ALL-USES (AU) and ALL-DEFS (AD). The first two criteria tied with mutation score of 96.8% and execution time of 2.5%, and the third criterion with 96.2% and 2.2% respectively. The worst performing group is composed of ALL-TRANSITIONS, ALL-CLOCK-RESETS, ALL-INPUT-ACTIONS, ALL-CLOCK-GUARDS, ALL-LOCATIONS, and ALL-OUTPUT-ACTIONS criteria; their average mutation score were less than 86%.

Therefore, evidence suggests that ALL-ONE-LOOP-PATHS outperforms the other evaluated criteria in terms of effectiveness but it is also the one that costs the most. Even so, it is a large reduction in cost execution compared to the full test suite — it costs on average between 13.7 and 66.6% of the execution time of the full test suite. If maximum effectiveness can be sacrificed, ALL-SIMPLE-PATHS is very close to ALL-ONE-LOOP-PATHS in terms of effectiveness but its execution cost is about the half of ALL-ONE-LOOP-PATHS. Further cost reduction may be achieved by using ALL-TRANSITION-PAIRS, however a little less effectiveness is expected. ALL-USES has better cost-effectiveness among the data-flow oriented criteria, even it is strictly included by ALL-DU-PATHS in the hierarchy of criteria. ALL-CLOCK-GUARDS is better than ALL-CLOCK-RESETS, however both real-time systems criteria are not on par with the best criteria.

4.4.2 SQ2: What test suite reduction technique has the best cost-effectiveness?

In the SUB experiment (Figure 4.34), the best performing test suite reduction technique was RANDOM, which on average killed 95.6% of mutants that the full test suite was able to and reduced execution time to 6.5% of the full test suite. While all other criteria obtained 93.9% of mutation score, they have got different average execution times: GRE 4.6%, G and GE 4.5%, and HGS 4.0%.

In the BAS experiment (Figure 4.35), the best performing test suite reduction technique was also RANDOM, which killed 88.7% of the mutants and reduced execution time to 22.0% of execution time of the full test suite on average. G, GE and HGS techniques killed 87.0% of the mutants and reduced execution time to 18.2% of the total time. The worse technique was GRE with mutation score of 86.0% and execution time of 18.7%.

In the ASL experiment (Figure 4.36), the best performing technique was RANDOM again, with mutation score of 87.2% and 3.8% of total execution time. It was followed by HGS with mutation score of 85.9%, GRE with 85.6%, and G and GE with 85.5%, but the differences were not statistically significant. With respect to execution time, there were statistically significant differences between HGS (2.8%) and G, GE and GRE (2.9%).

Empirical results show that the RANDOM technique was the most effective but also the most expensive to execute. The other evaluated techniques had the same effectiveness, however they showed differences in execution times. HGS had the lowest execution times, GRE had the second-highest execution times, and G and GE were between them.

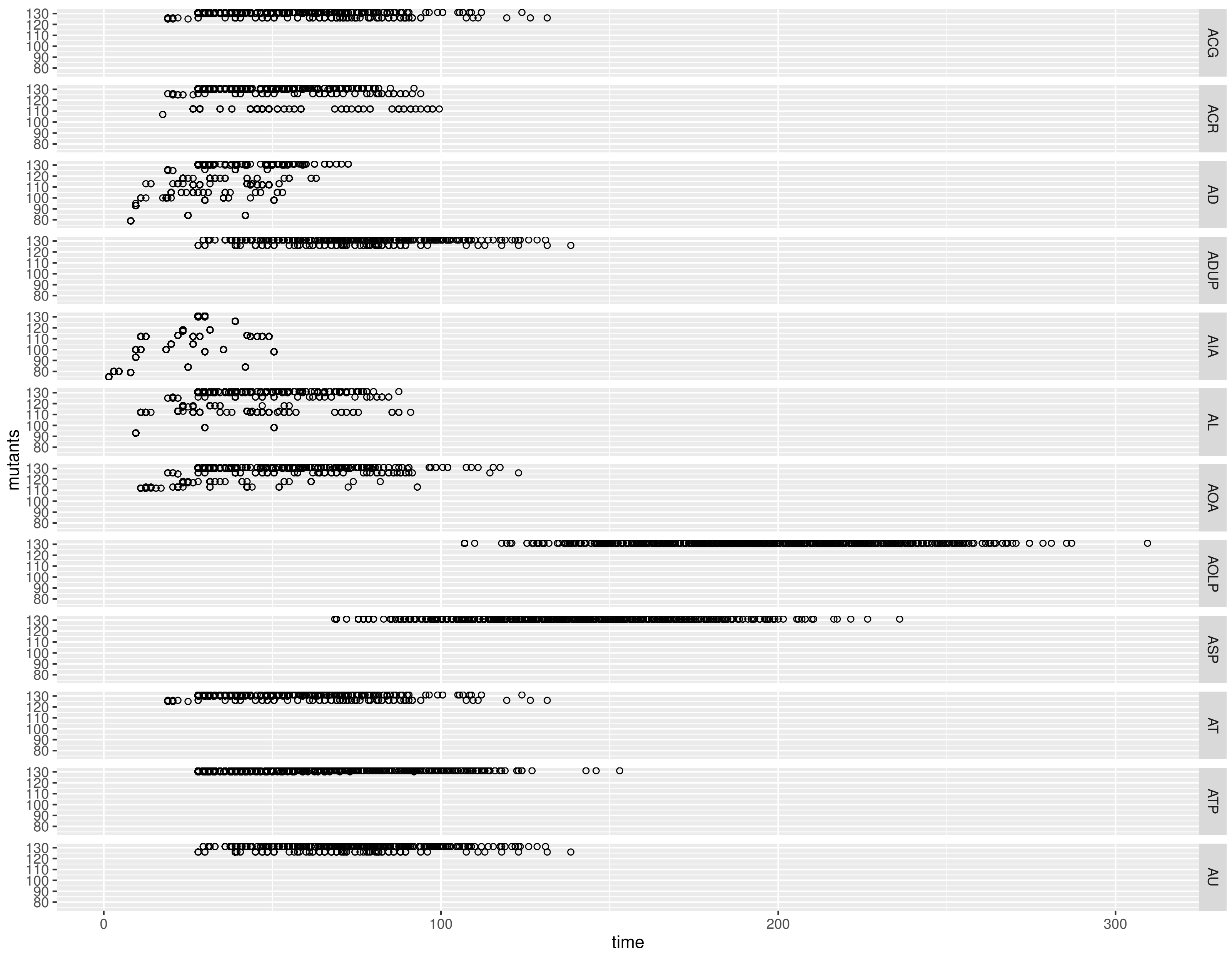


Figure 4.31: Refilling Machine: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test adequacy criterion.

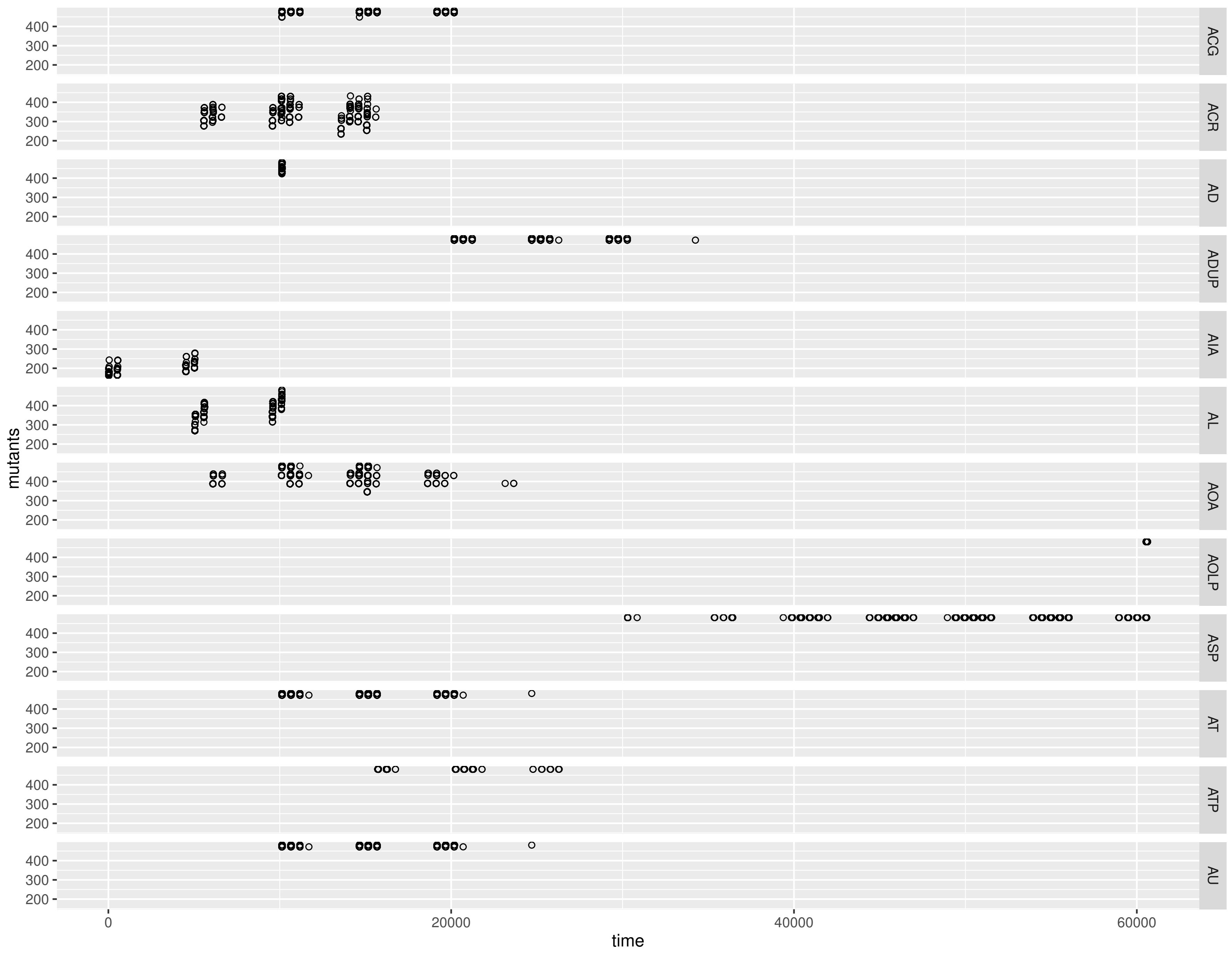


Figure 4.32: Burglar Alarm System: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test adequacy

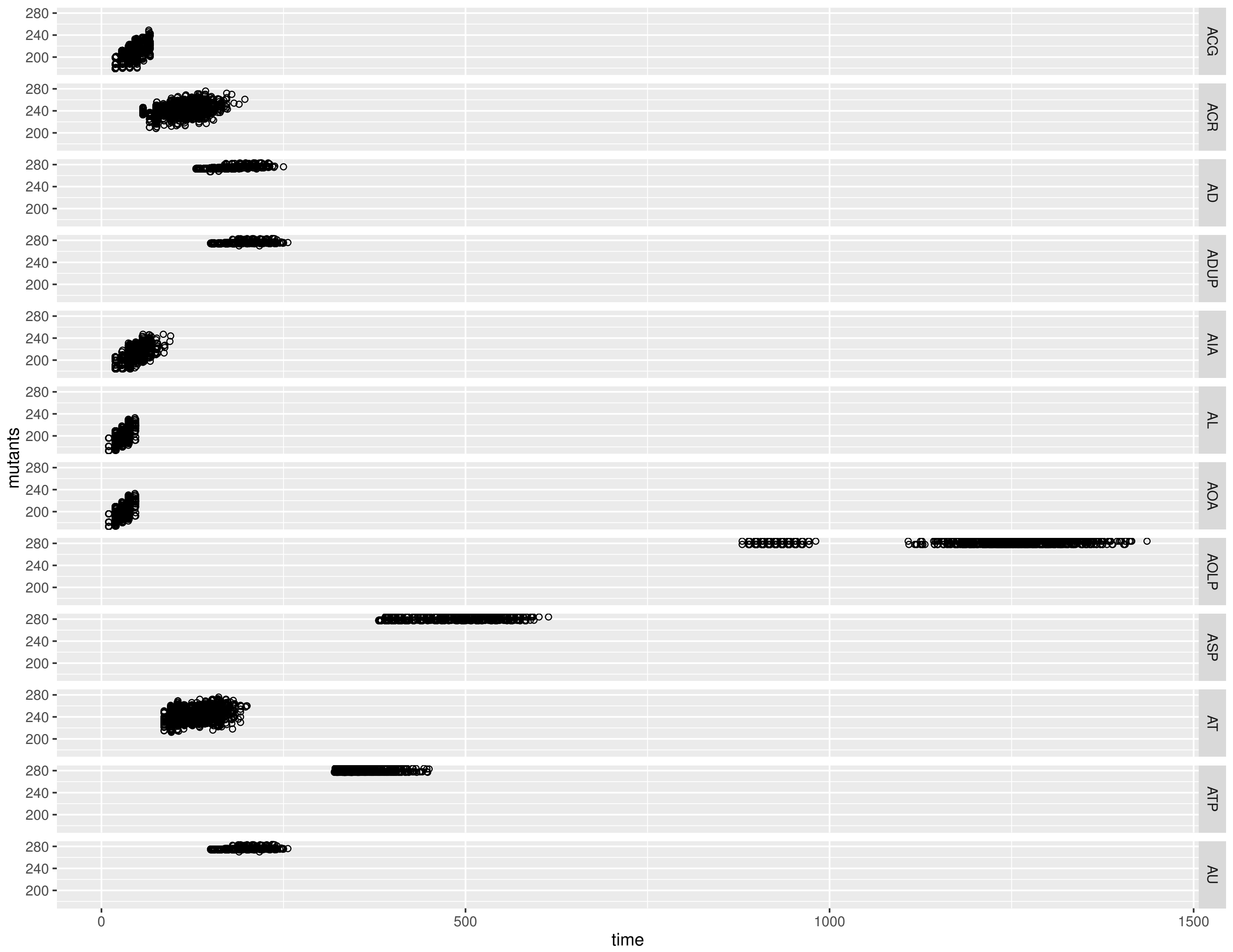


Figure 4.33: Automated Speed Limiter: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test adequacy

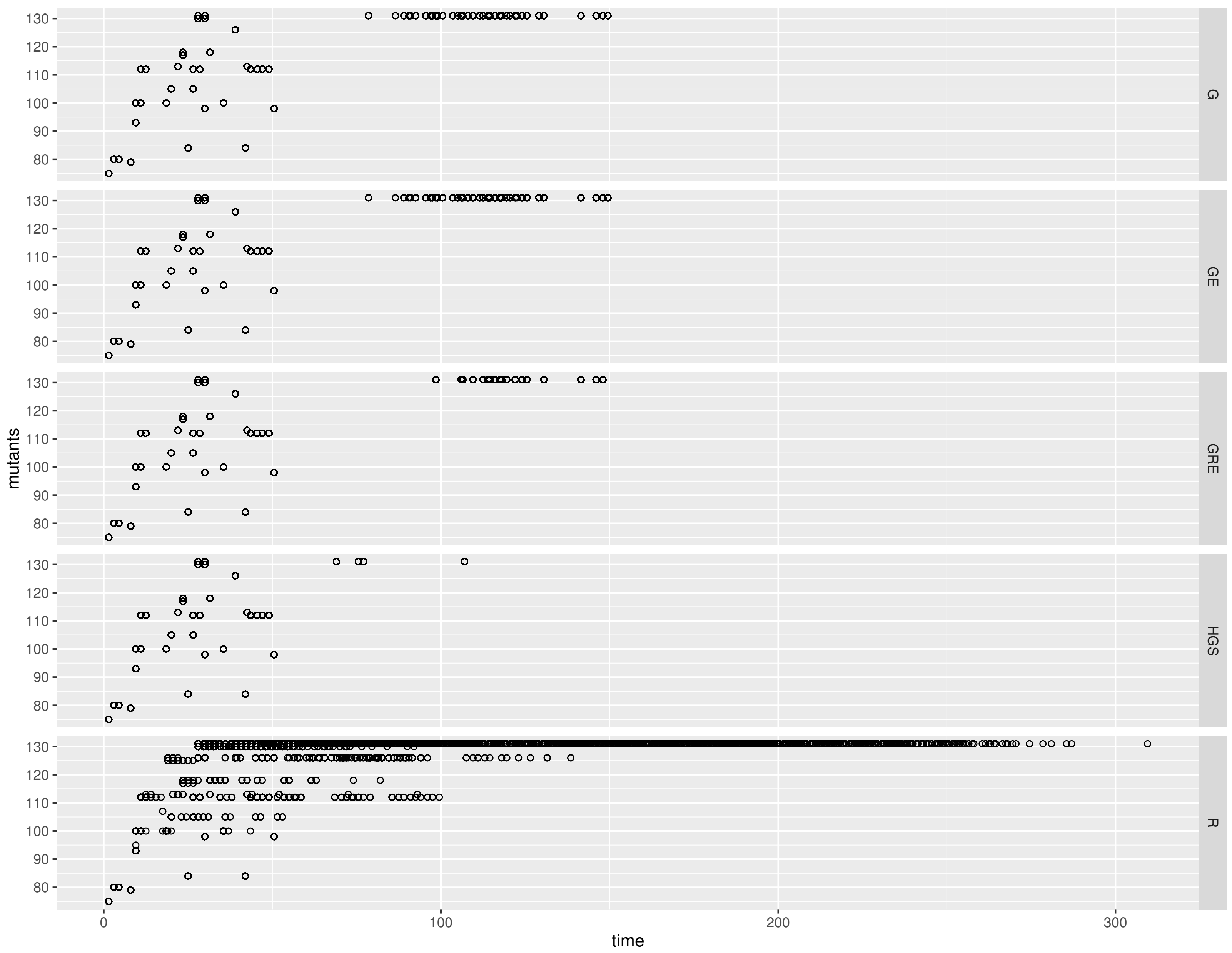


Figure 4.34: Refilling Machine: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test suite reduction

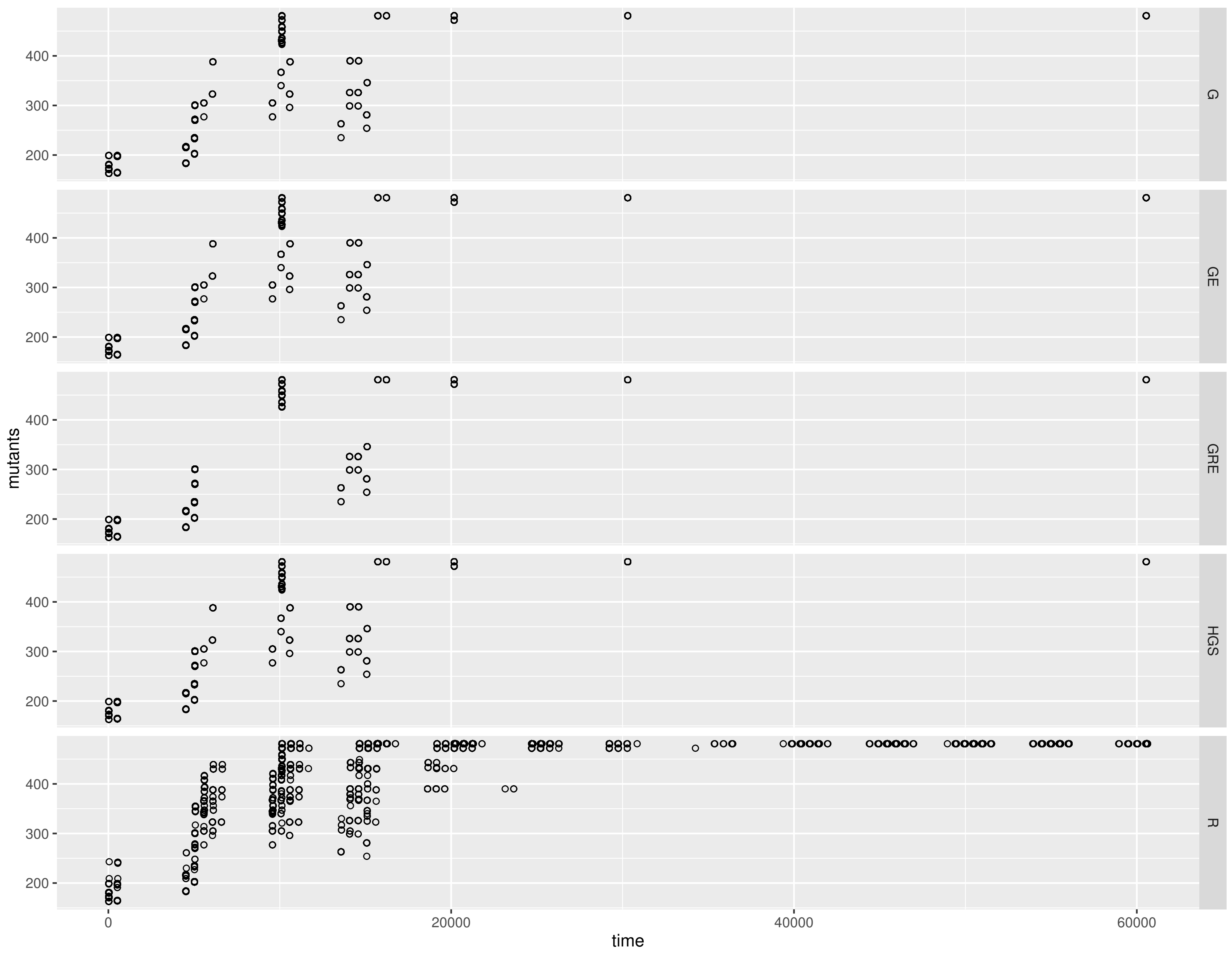


Figure 4.35: Burglar Alarm System: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test suite reduction

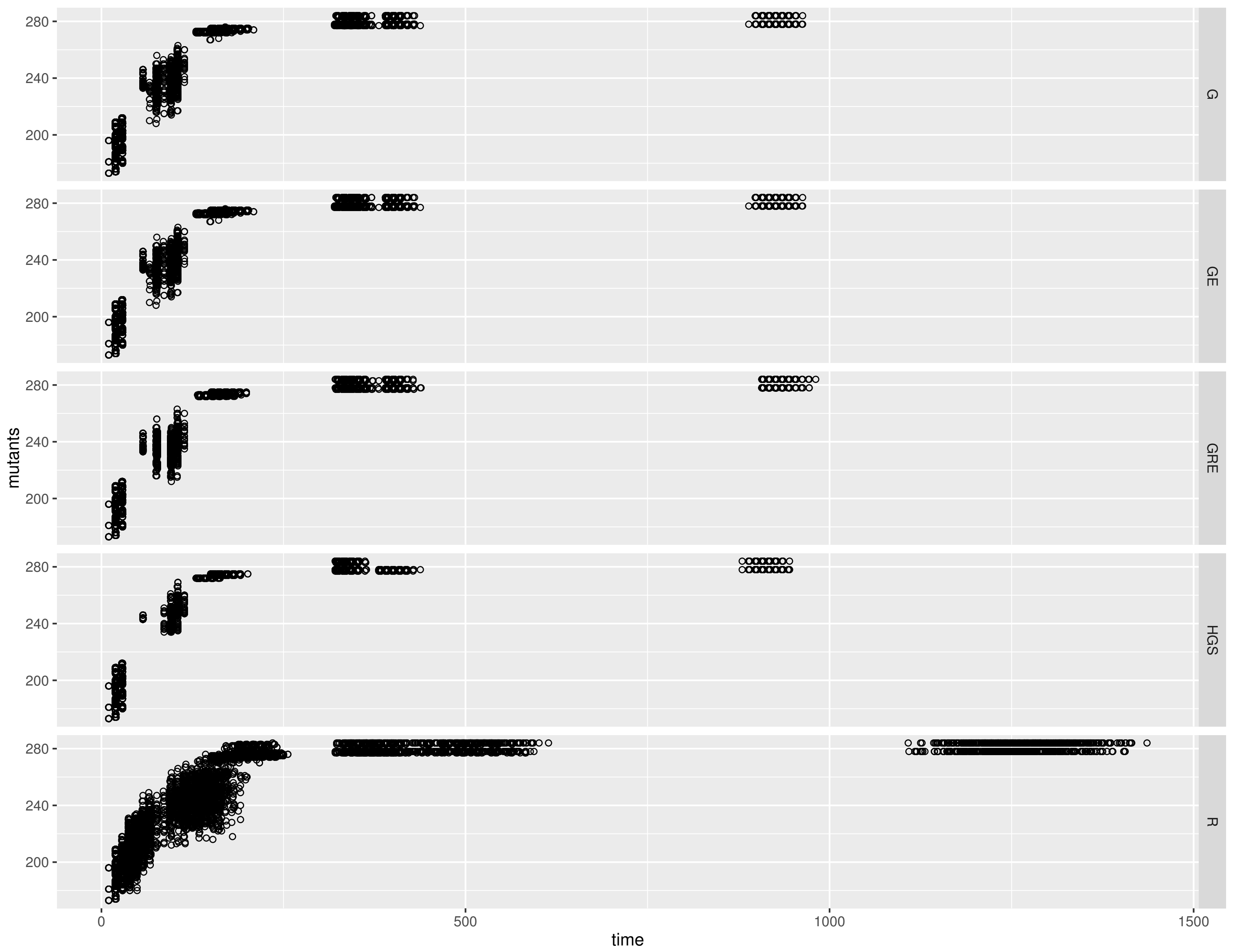


Figure 4.36: Automated Speed Limiter: Scatter plots of cost (execution time) vs. effectiveness (number of killed mutants) by test suite

In fact, G and GE produced the same results in the experiment. Since G algorithm is simpler than GE algorithm, then we classify G as better than GE. Therefore, test suite reduction techniques can be sorted by cost-effectiveness, from worst to best: GRE, GE, G, HGS, and RANDOM.

4.4.3 SQ3: How much of the variation in the cost and effectiveness of reduced test suites is explained by criteria and techniques?

In previous sections, we analyzed individual effects of test adequacy criteria and test suite reduction techniques in test suite reduction outcomes. Here we analyze whether one of the factors has larger effect than the other factor. For each dependent variable, we perform variance allocation between the factors as recommended by Montgomery [53].

For number of test cases, in the SUB experimental unit (Figure 4.10), the criteria explain 73.2% of the variance in the results and techniques explain 13.0%. In the BAS experimental unit (Figure 4.19), the criteria explain 97.0% of the variance in the results and techniques explain 0.1%. Finally, in the ASL experimental unit (Figure 4.28), the criteria explain 97.2% of the variance in the results and technique explains 0.9%.

For execution time, in the SUB experimental unit (Figure 4.11), the criteria explain 81.3% of the variance in the results and techniques explain 5.6%. In the BAS experimental unit (Figure 4.20), the criteria explain 95.9% of the variance in the results and techniques explain 0.8%. Finally, in the ASL experimental unit (Figure 4.29), the criteria explain 97.2% of the variance in the results and technique explains 0.9%.

For number of killed mutants, in the SUB experimental unit (Figure 4.12), the criteria explain 52.9% of the variance in the results and techniques explain 0.7%. In the BAS experimental unit (Figure 4.21), the criteria explain 96.2% of the variance in the results and techniques explain 0.2%. Finally, in the ASL experimental unit (Figure 4.30), the criteria explain 95.8% of the variance in the results and technique explains 0.3%.

The numerical results above are can be easily visualized in interaction graphs. Figures 4.37 and 4.38 show interaction plots of techniques and criteria for number of killed mutants, and Figures 4.39 and 4.40 show interaction plots for execution time for the Refilling Machine experiment. Figures 4.41 and 4.42 show interaction plots of techniques and criteria for number of killed mutants, and Figures 4.43 and 4.44 show interaction plots for execution time for the Burglar Alarm System experiment. Figures 4.45 and 4.46 show interaction plots of techniques and criteria for number of killed mutants, and Figures 4.47 and 4.48 show interaction plots for execution time for the Automated Speed Limiter experiment.

By examining the plots, we observe that a criterion change affects more the results than a technique change. For instance, in Figure 4.37 the lines of the criteria are clearly separated and their arrangement do not change significantly when the technique changes, except in the case of the RANDOM technique. On the other hand, in Figure 4.38 the lines of the techniques are much more grouped and they change drastically when the criterion is changed. The same pattern is observed in Figures 4.41 and 4.42, and in Figures 4.45 and 4.46. As with number of killed mutants, we observe the same pattern in interaction plots for execution time.

Therefore, empirical evidence suggests that criteria are the major driver of the outcomes in test suite reduction. The closer to the top of the hierarchy of criteria, the less the technique influences the outcomes of test suite reduction.

4.5 Discussion

First, we discuss the relationships between test adequacy criteria, test suite reduction techniques, test requirements, test cases, execution time and faults. By using this framework we explain the results of the experiment.

Given a model, a test adequacy criterion defines the test requirements a test suite must cover. A test requirement may be covered by more than one test case, and a test case may cover more than one test requirement. A test case has an associated execution time and can reveal some faults, but both information are not known upfront in test suite reduction. Given a test adequacy criterion, a test suite reduction technique produces a reduced test suite which satisfies all test requirements defined by the criterion. However, a potential side effect of the reduction is the decrease of effectiveness at detecting faults.

The challenge of the test suite reduction problem is to obtain a reduced test suite that maintains the fault detection capability of the complete test suite and minimizes its execution time without upfront knowledge of the fault detection of each test case. Otherwise, if fault detection of test cases is known in advance, the test suite reduction problem would be simply

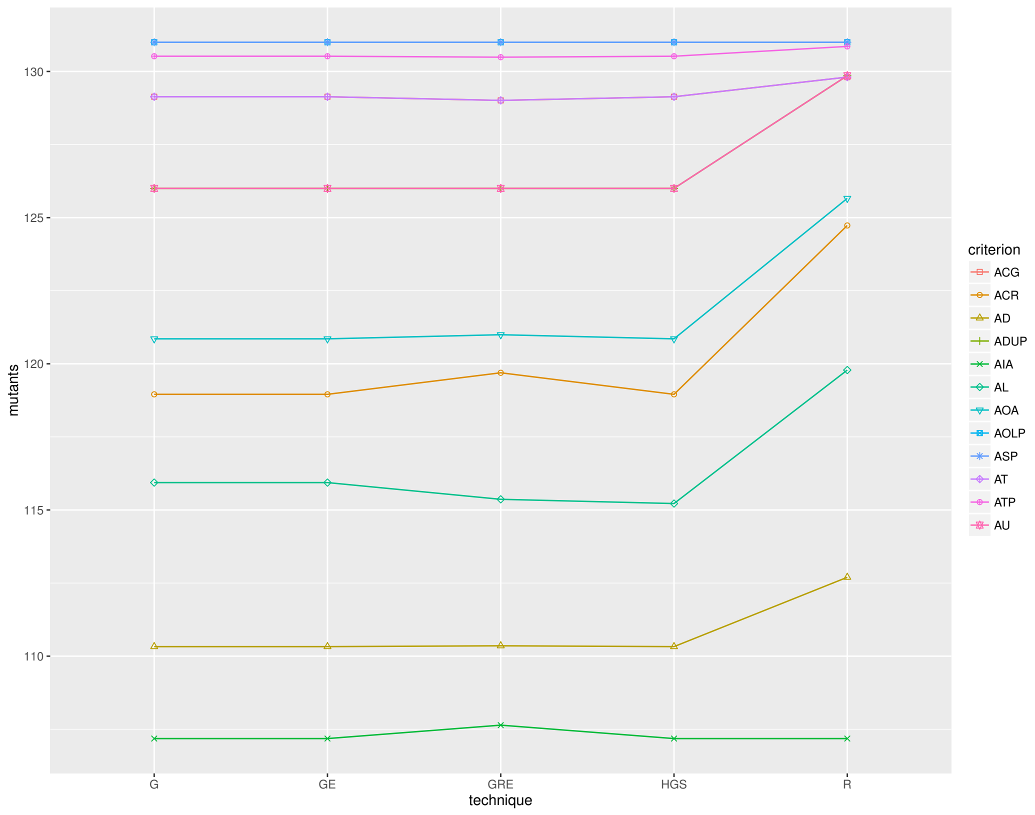


Figure 4.37: Refilling Machine: Interaction plot for number of killed mutants (criterion by technique).

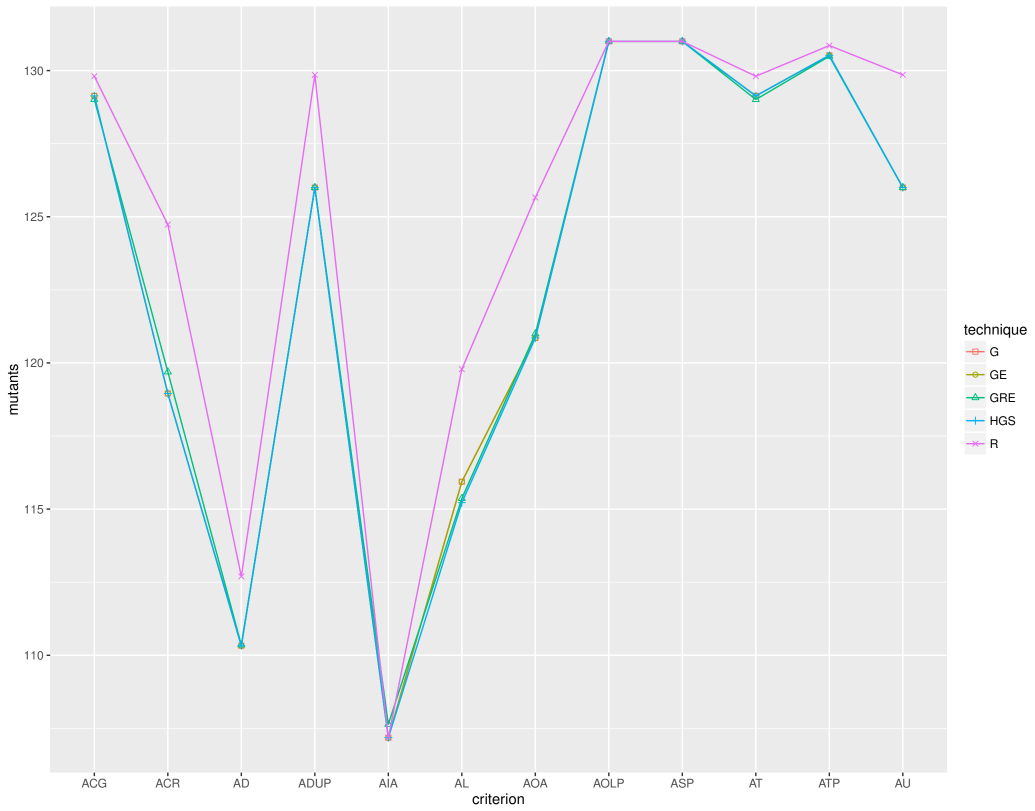


Figure 4.38: Refilling Machine: Interaction plot for number of killed mutants (technique by criterion).

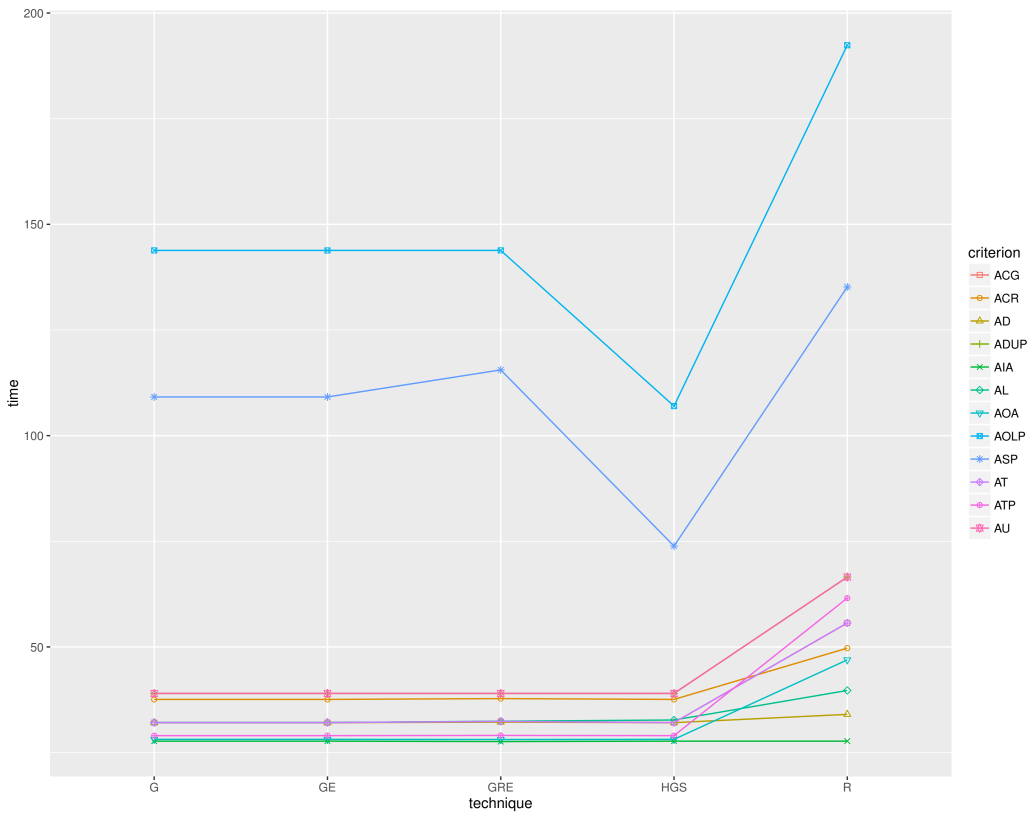


Figure 4.39: Refilling Machine: Interaction plot for execution time (criterion by technique).

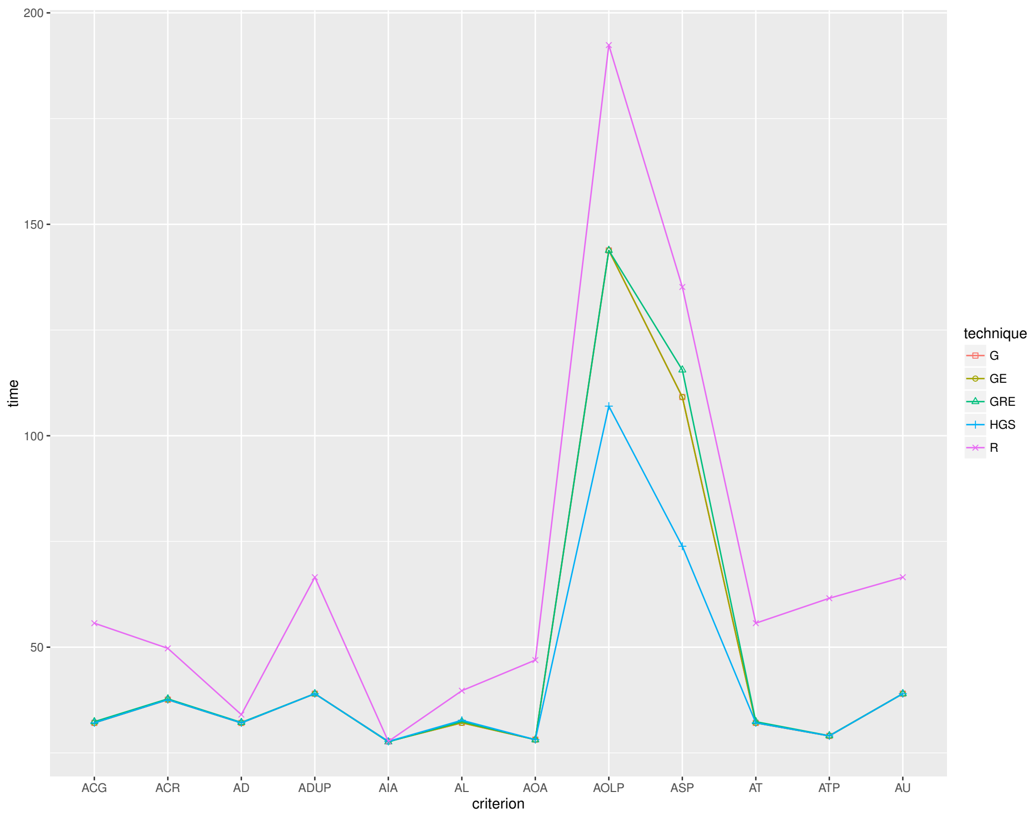


Figure 4.40: Refilling Machine: Interaction plot for execution time (technique by criterion).

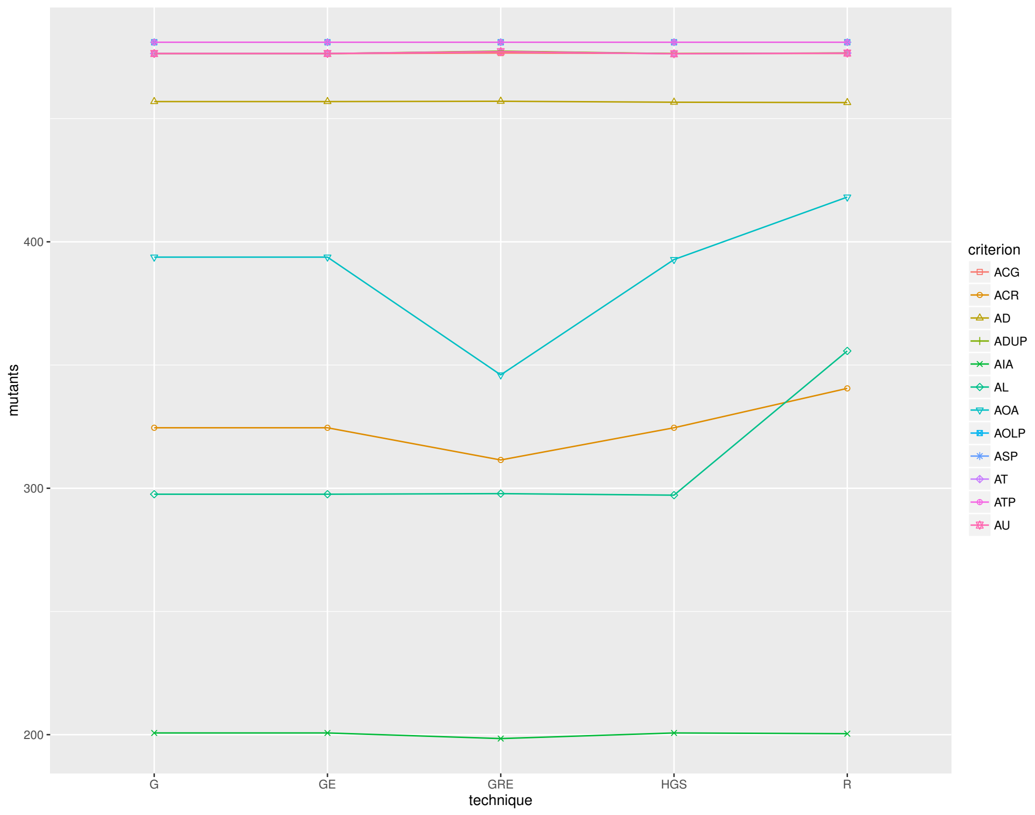


Figure 4.41: Burglar Alarm System: Interaction plot for number of killed mutants (criterion by technique).

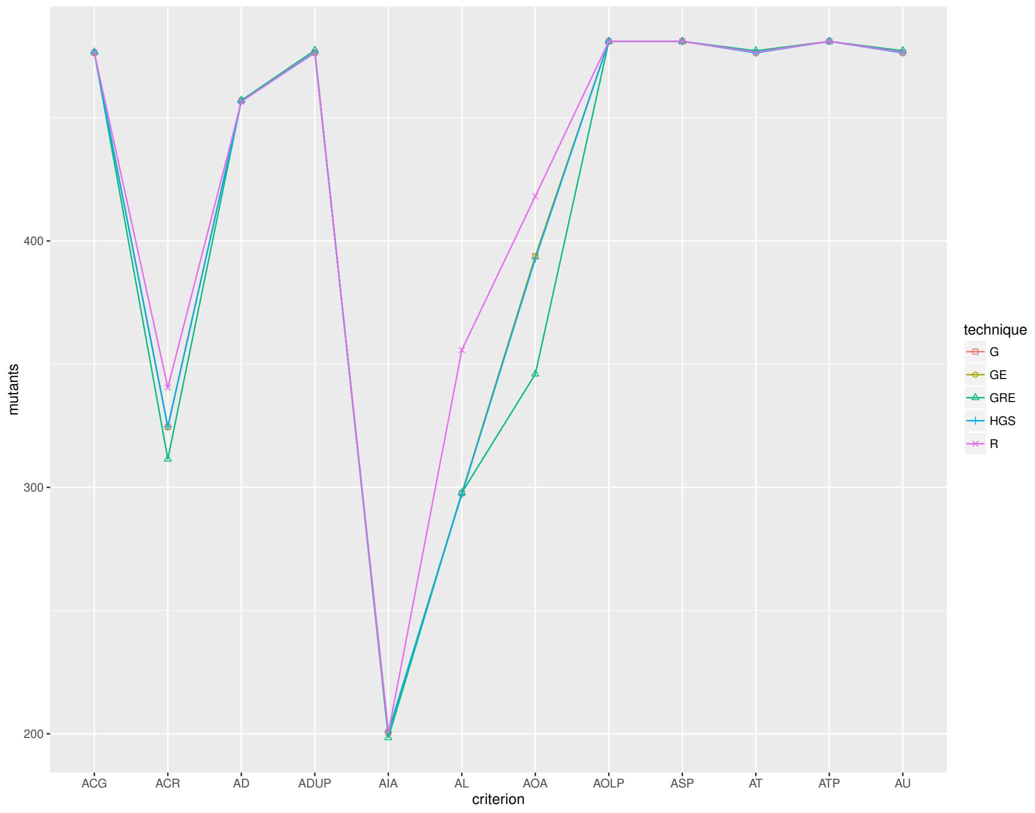


Figure 4.42: Burglar Alarm System: Interaction plot for number of killed mutants (technique by criterion).

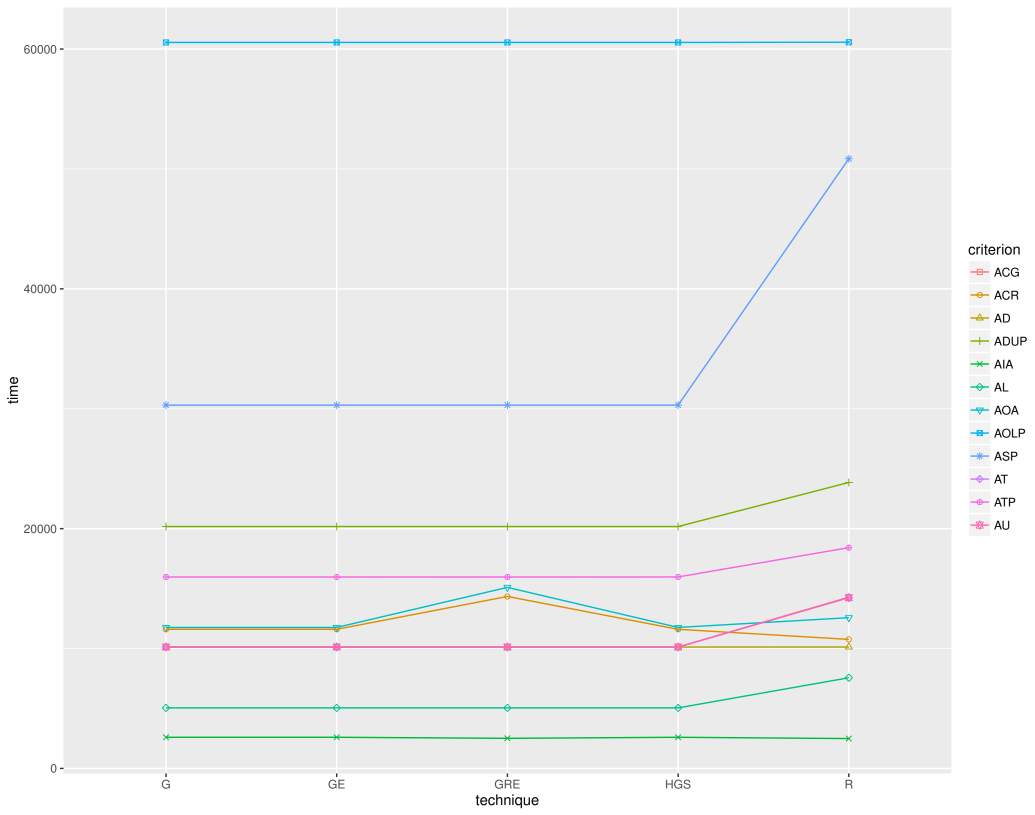


Figure 4.43: Burglar Alarm System: Interaction plot for execution time (criterion by technique).

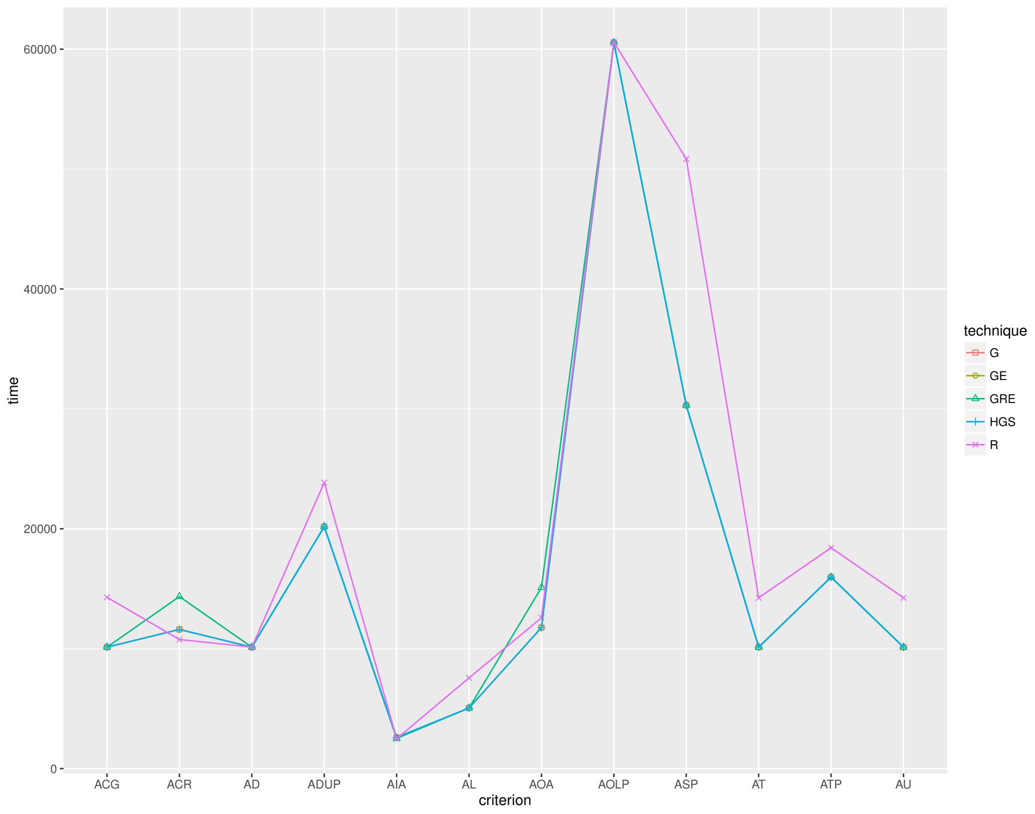


Figure 4.44: Burglar Alarm System: Interaction plot for execution time (technique by criterion).

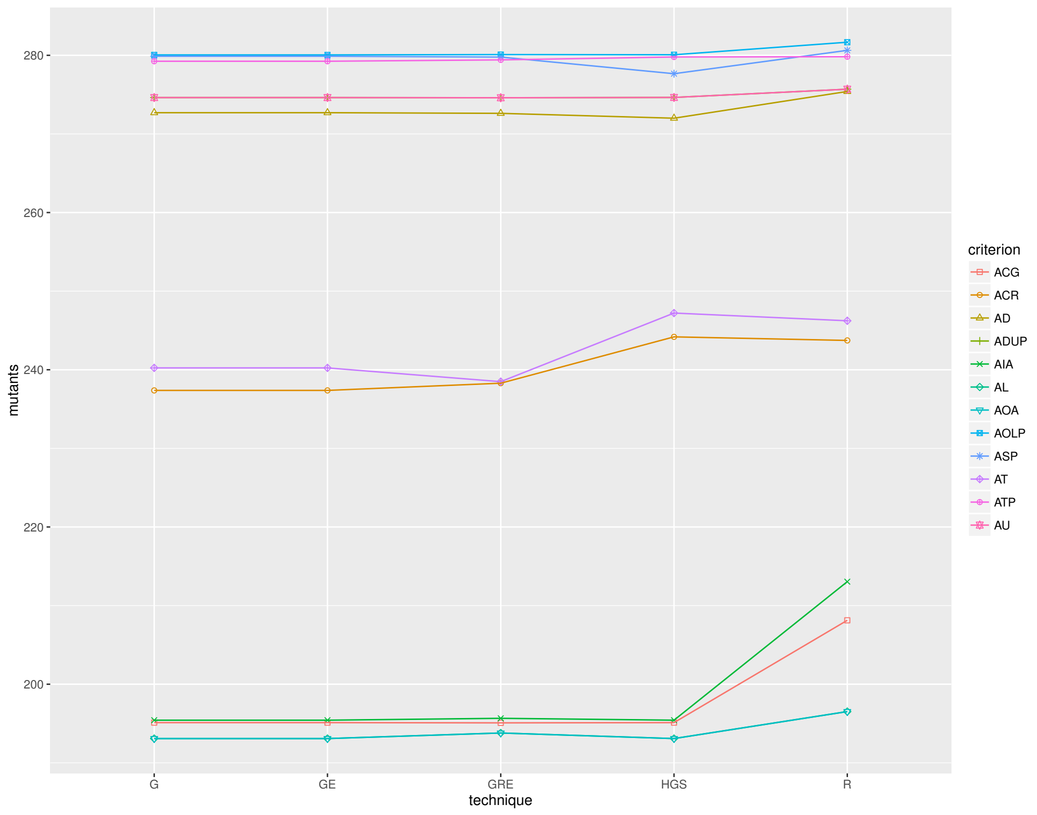


Figure 4.45: Automated Speed Limiter: Interaction plot for number of killed mutants (criterion by technique).

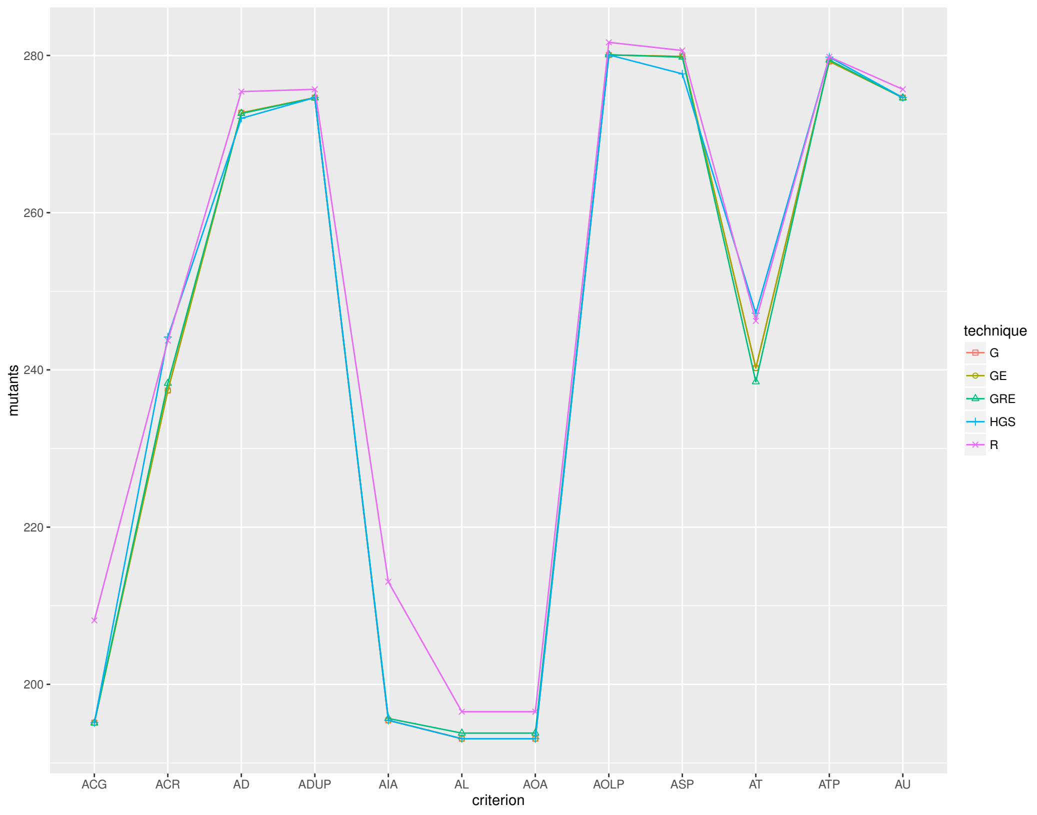


Figure 4.46: Automated Speed Limiter: Interaction plot for number of killed mutants (technique by criterion).

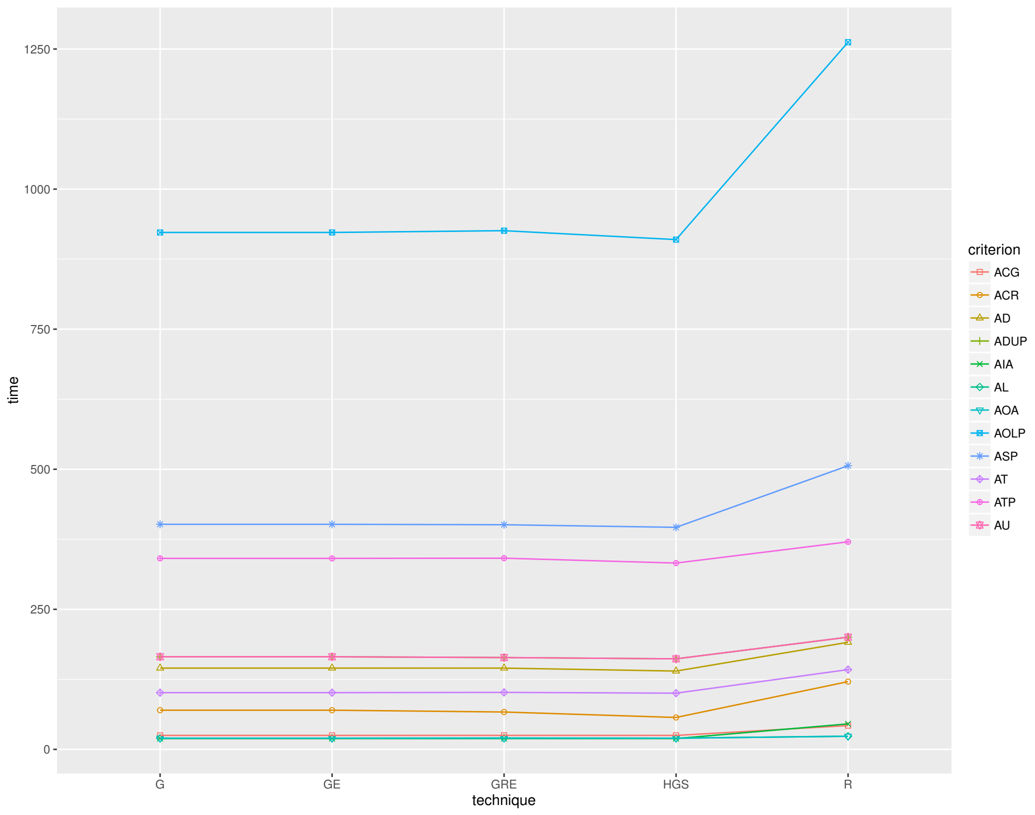


Figure 4.47: Automated Speed Limiter: Interaction plot for execution time (criterion by technique).

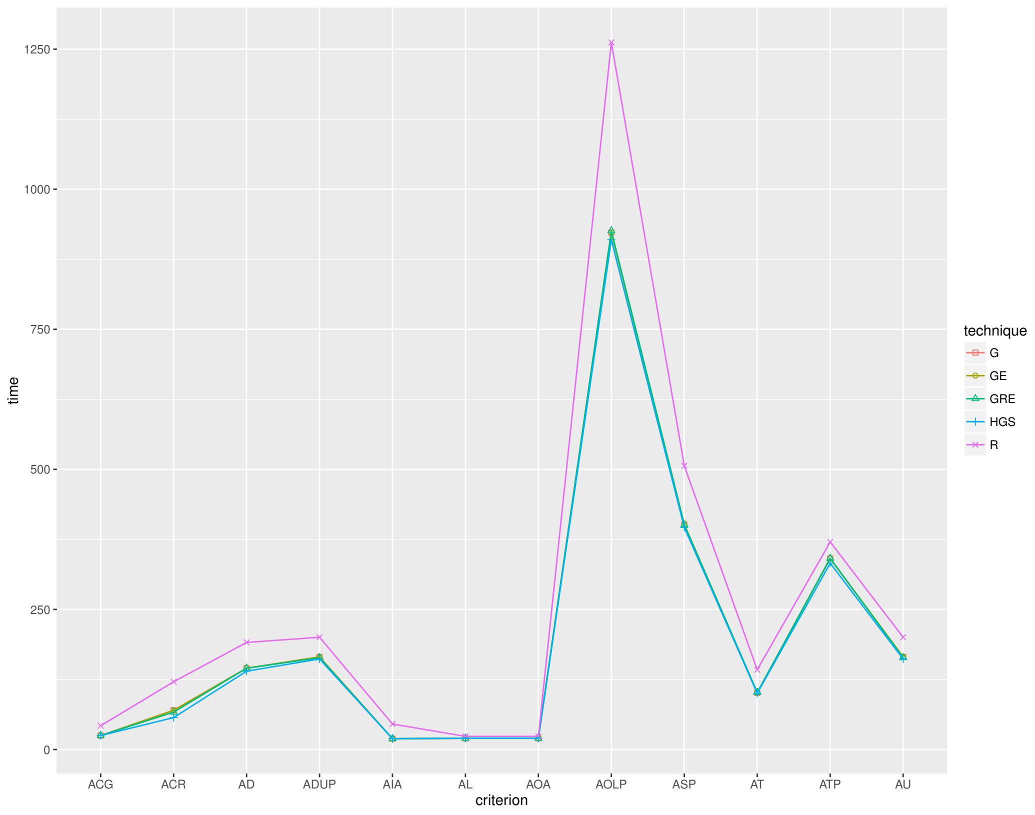


Figure 4.48: Automated Speed Limiter: Interaction plot for execution time (technique by criterion).

a direct application of the set covering problem (SCP) from test cases to faults (instead of test requirements), which have optimal solutions even for large SCP problems.

We recall that the test requirements of all criteria evaluated in the experiment are paths in the model. Criteria near the top of the hierarchy demand a greater number of test requirements, and these requirements also include longer paths to be covered. For example, ALL-TRANSITIONS requires all paths of length 1 in the model to be covered, while ALL-SIMPLE-PATHS requires all simple paths in the model to be covered, i.e. all simple paths of length 1, all simple paths of length 2, and so on.

In the experiment, we observed positive correlation between the number of test cases and execution time, and number of test cases and fault detection capability, i.e. larger test suites take more time to execute and detect more defects. Although there is no direct relationship between individual test requirements and faults, satisfying the set of test requirements of a "strong" criterion such as ALL-SIMPLE-PATHS requires more and longer test cases (with more transitions) in the reduced test suite, consequently it is more effective at detecting faults because the positive correlation between number of test cases and number of killed mutants. On the other hand, criteria near the bottom of the hierarchy are less demanding, thus less test cases are needed in the reduced test suite to satisfy the criterion and, consequently, it is less effective at detecting faults.

On the other side, the RANDOM technique produces more effective reduced test suites than other techniques because it does not produce near-optimal suites in terms of size, while G, GE, GRE and HGS heuristics produce smaller reduced test suites, which are in turn less effective at fault detection. Since there is positive correlation between number of test cases and number of killed mutants, worse techniques in the sense of the set covering problem —techniques that reduce less the size of the test suite— obtain reduced test suites that are more effective at detecting faults because they are larger.

We also observed that results of best performing criteria have less variance because they demand a greater number of test requirements which, in turn, limit the number of reduced test suites that satisfy the criteria. As we already discussed, the reduced test suites obtained from stronger criteria are larger and, consequently, they detect more faults. In other words, the stronger the criterion, the smaller the number of combinations of test cases that can satisfy the criterion. With less possible solutions, there is little room to techniques to play with, therefore the contribution of the technique to the results of reduction is further reduced, so that changing the technique will have little effect in the results compared to what can be obtained from changing the criterion to a stronger one. This claim is backed up from the ANOVA results, where the criterion explains 73 to 97% of the variation in the number of test cases, 81 to 97% in the execution time, and 52 to 96% in the number of killed mutants.

We conclude that criteria set an anchor for reduction outcomes such that the gain obtained by changing techniques while maintaining the criterion is marginal. In other words, no technique considerably increases the effectiveness if a "strong" criterion is used in first place. For example, results of the experiment points out that ALL-ONE-LOOP-PATHS criterion made the choice of technique unimportant. Finally, the key in test suite reduction problem is the design (and use) of test adequacy criteria that produce test requirements that match the faults, so, by covering the test requirements of a criterion, all faults would be detected and the test suite would still be reduced, regardless of the technique employed. In this sense, the criterion ALL-ONE-LOOP-PATHS is very close fully achieve this objective.

4.6 Threats to Validity

In this section we discuss what we consider to be threats to validity in our experiment. We followed the checklist provided by Wohlin et al. [71] and report here only the most significant threats in our experiment.

4.6.1 Internal validity

The first threat that may affect the conclusion about a possible causal relationship between treatment and outcome is the ambiguity of the direction of causal influence. We mitigated this threat by strictly controlling the experiment execution in laboratory, therefore the causal relation is unambiguous because the only factor that could affect the outcome is the choice of the test adequacy criterion. Moreover, the correctness of the implementation of the algorithms is critical to assess whether the results are reliable. Consequently, the validation was throughly performed, and the results of the algorithms were manually reviewed and compared with expected outputs. Finally, another threat is the fact that we used only one algorithm to generate test cases, hence the results could be biased towards the algorithm.

However, this is a common practice in other works presented in literature in order to provide more control over the experiment. Additionally, to mitigate this threat, we generated the test suite by using an algorithm that embeds a criterion stronger than the ones investigated in this paper; thus, we produced a large test suite with different kinds of test cases. From these suites, one hundred different test suites were selected for each criterion based on random choice, addressing the need for diversity of test suites. Nevertheless, we plan to conduct a study to investigate the influence of the test generation algorithm on cost-effectiveness of test adequacy criteria.

4.6.2 Construct validity

Some threats could concern generalization of the results of the experiment to the evaluated theory. Inadequate preoperational explication of constructs is not a threat because the constructs are clear and well understood by the community. The size, cost and effectiveness constructs were translated into the widely accepted measures of number of test cases, execution time, and number of defects found respectively. There is mono-operation bias regarding the single experimental unit. Although more experimental units are always desirable in empirical studies, we used a representative real-time system specification to conduct our experiment and we made it available to the community. So the practitioner can compare it with his systems, and decide whether our conclusions can be applied to his context.

4.6.3 Conclusion validity

Regarding threats to statistical conclusions, we discarded low statistical power as a threat because we used an adequate sample size. Although some ANOVA assumptions were not met, we minimized this threat by using a large number of replications for each experimental unit; also each combination of factors has the same number of replications. In order to avoid the "fishing" threat, we performed two-way analysis of variance with all levels of the treatments. When the null hypothesis was rejected, we performed post-hoc analysis with Tukey's HSD, in which the $p\text{-}values$ are adjusted to multiple comparisons. It is a simple and conservative method to control the family-wise error rate [53]. Reliability of measures is not a threat because the measures we used (number of test cases, execution time, and number of defects found) are objective and do not require human judgment. Finally, we did not detect any random irrelevancies in the experimental setting during the experiment operation that could affect the results.

4.6.4 External validity

The first threat that may limit our ability to generalize the results is the interaction of setting and treatment. This might be a threat because we used a simulation instead of an actual implementation, however by using a simulation we could control the experiment and isolate factors that might affect our interpretation of results. In our context, an implementation simulation of some model represents all actual implementations that are able to generate the same trace of the specification model. Since we just need the trace generated by an implementation in order to verify if it conforms to a specification for tioco [10; 67], the use of simulations did not affect the generality of our conclusions. Another threat may be the use of mutants instead of actual faults. Since we do not have actual implementations but simulations, it is not possible to have actual faults at all. So we mutated the specification models instead of mutating implementations, which gave us faulty models [1]. Additionally, we designed first-order mutation operators that changes every element of the TIOSTS model (locations, data guards, clock guards, assignments, and clock resets). Therefore, the combination of mutating specification models and simulating their implementations provided a robust basis for experimental evaluation. Finally, since we used just three experimental units, it is hard to generalize the results to all real-time systems. As we already discussed, we reduced this threat by using specifications that represents a common classes of real-time systems.

4.7 Concluding Remarks

In this chapter, we presented empirical studies with the goal of evaluating the effect of test suite reduction techniques and test adequacy criteria on test suite reduction outcomes. Empirical evidence suggests that test adequacy criteria have different cost-effectiveness as test suite reduction techniques also do, however the influence of criteria is larger than the influence of techniques in test suite reduction outcomes.

Chapter 5

Concluding Remarks

5.1 Conclusions

The main objective of this doctorate research is to evaluate the effect of the choice of the test adequacy criterion on the outcomes of test suite reduction for model-based testing of real-time systems. Considering the research questions defined in Section 1.2, the following results were achieved:

Research Question 1 *What is theoretical relationship among TIOSTS test adequacy criteria?*

We formalized a hierarchy of 19 test adequacy criteria partially ordered by strict inclusion. The family of criteria combines general-purpose transition-based and data-flow-oriented criteria with specific reactive and real-time systems criteria. We proved that the theoretical relationship between transition-based and data-flow-oriented criteria in our family is different from the relationship between the corresponding criteria for source code.

Research Question 2 *What is the influence of test adequacy criteria for TIOSTS models on test suite reduction for model-based testing of real-time systems?*

Empirical evidence suggests that test adequacy criteria have different cost-effectiveness as test suite reduction techniques also do, however the influence of criteria is larger than the influence of techniques in test suite reduction. This conclusion has direct implication for this research area because most of the effort currently is directed toward research of reduction techniques instead of the development of new test adequacy criteria, which should yield better practical results.

5.2 Future Work

There are several problems that need to be solved and improved in future works. Next, some work proposals are presented:

New test adequacy criteria The main conclusion of this doctorate research is that test adequacy criteria play a major role in test suite reduction, greater than the role of reduction techniques themselves. Since our family of criteria does not exhaust all possibilities of criteria, there is research opportunity to devise new criteria with the goal of improving cost-effectiveness. For instance, real-time test adequacy criteria from literature adapted to TIOSTS models did not perform well in our empirical studies, because their requirements are subsets of the requirements of ALL-TRANSITIONS criterion. However, new real-time-oriented criteria could be inspired in data-flow-oriented criteria, e.g. cover all paths from a transition that resets a clock to all transitions that use that clock in their clock guards. Another possibility is to create new criteria by composition of existing criteria.

More factors influencing test suite reduction Besides test suite reduction techniques and test adequacy criteria, there are other factors that may influence test suite reduction. One of them is the test suite itself because the number of test cases generated, the number of loop traversals, and the length of each test case may affect criteria satisfactory and, consequently, impact reduction results. The test suite generation algorithm could also be investigated as a factor in test suite reduction. In our research we opted to reduce full test suites generated by an algorithm which embeds its own stopping generation criteria, but the approach to just generate test cases until a criterion is satisfied should be considered. Certainly some cost saving may be obtained if test case generation is pruned, however how much of effectiveness is sacrificed?

Test data selection The effectiveness of a test suite depends on the test data used with its test cases, and having a more or less effective test suite may potentially change results of test suite reduction research. In this research we selected test data randomly but there are many techniques for test data selection such as equivalence class partitioning and combinatorial testing which deserve further investigation.

Bibliography

[1] AICHERNIG, B. K., BRANDL, H., JÖBSTL, E., KRENN, W., SCHLICK, R., AND TIRAN, S. Killing strategies for model-based mutation testing. *Software Testing, Verification and Reliability 25*, 8 (Dec. 2015), 716–748.

[2] AICHERNIG, B. K., HÖRMAIER, K., LORBER, F., NIČKOVIĆ, D., AND TIRAN, S. Require, test, and trace it. *International Journal on Software Tools for Technology Transfer 19* (2017), 409–426.

[3] AICHERNIG, B. K., LORBER, F., AND NICKOVIC, D. Time for mutants - model-based mutation testing with timed automata. In *Tests and Proofs - 7th International Conference, TAP 2013, Budapest, Hungary, June 16-20, 2013. Proceedings* (2013), M. Veanes and L. Viganò, Eds., vol. 7942 of *Lecture Notes in Computer Science*, Springer, pp. 20–38.

[4] ALAGAR, V. S., ORMANDJIEVA, O., AND ZHENG, M. Specification-based testing for real-time reactive systems. In *Proceedings of the 34th International Conference on Technology of Object-Oriented Languages and Systems* (2000), pp. 25–36.

[5] ALMEIDA, D. R., MORAES, A., ANDRADE, W. L., AND MACHADO, P. D. L. Towards a family of test selection criteria for symbolic models of real-time systems. In *Formal Methods: Foundations and Applications*, C. Braga and N. Martí-Oliet, Eds., vol. 8941 of *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 48–63.

[6] ALUR, R., AND DILL, D. L. A theory of timed automata. *Theoretical Computer Science 126*, 2 (1994), 183–235.

[7] AMMANN, P., AND OFFUTT, J. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 2008.

[8] ANAND, S., BURKE, E. K., CHEN, T. Y., CLARK, J., COHEN, M. B., GRIESKAMP, W., HARMAN, M., HARROLD, M. J., AND MCMINN, P. An orchestrated survey of methodologies for automated software test case generation. *The Journal of Systems and Software 86* (2013), 1978–2001.

[9] ANDRADE, W. L., ALMEIDA, D. R., CÂNDIDO, J. B., AND MACHADO, P. D. L. SYMBOLRT: A tool for symbolic model-based test case generation for real-time systems. In *Proceedings of the 3rd Brazilian Conference on Software: Theory and Practice* (Natal, Brazil, 2012), vol. 4 of *CBSoft*, pp. 31–37.

[10] ANDRADE, W. L., AND MACHADO, P. D. L. Generating test cases for real-time systems based on symbolic models. *IEEE Transactions on Software Engineering 39*, 9 (2013), 1216–1229.

[11] ANDRADE, W. L., MACHADO, P. D. L., JÉRON, T., AND MARCHAND, H. Abstracting time and data for conformance testing of real-time systems. In *Proceedings of the 8th Workshop on Advances in Model Based Testing* (2011), pp. 9–17.

[12] ARCURI, A., AND BRIAND, L. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability 24*, 3 (2014), 219–250.

[13] ARCURI, A., IQBAL, M., AND BRIAND, L. Random testing: Theoretical results and practical implications. *IEEE Transactions on Software Engineering, 38*, 2 (March 2012), 258–277.

[14] ARCURI, A., IQBAL, M. Z., AND BRIAND, L. Black-box system testing of real-time embedded systems using random and search-based testing. In *Proceedings of the 22nd International Conference on Testing Software and Systems* (2010), ICTSS 2010, pp. 95–110.

[15] BARRETT, C., AND TINELLI, C. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)* (July 2007), W. Damm and

H. Hermanns, Eds., vol. 4590 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 298–302. Berlin, Germany.

[16] BEASLEY, J. An algorithm for set covering problem. *European Journal of Operational Research 31*, 1 (jul 1987), 85–93.

[17] BINDER, R. V., LEGEARD, B., AND KRAMER, A. Model-based testing: Where does it stand? *Communications of the ACM 58* (2015), 52–56.

[18] BLACK, J., MELACHRINOUDIS, E., AND KAELI, D. Bi-criteria models for all-uses test suite reduction. In *Proceedings of the 26th International Conference on Software Engineering* (2004), ICSE 2004, pp. 106–115.

[19] BRIAND, L. C., LABICHE, Y., AND WANG, Y. Using simulation to empirically investigate test coverage criteria based on statechart. In *Proceedings of the 26th International Conference on Software Engineering* (2004), ICSE, pp. 86–95.

[20] CAPRARA, A., TOTH, P., AND FISCHETTI, M. Algorithms for the set covering problem. *Annals of Operations Research 98*, 1 (2000), 353–371.

[21] CARTAXO, E. G., MACHADO, P. D. L., AND OLIVEIRA, NETO, F. G. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability 21*, 2 (2011), 75–100.

[22] CHEN, T., AND LAU, M. A new heuristic for test suite reduction. *Information and Software Technology 40* (1998), 347–354.

[23] CHVATAL, V. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research 4*, 3 (1979), 233–235.

[24] CICHOS, H., AND HEINZE, T. S. Efficient test suite reduction by merging pairs of suitable test cases. In *Models in Software Engineering*, J. Dingel and A. Solberg, Eds., vol. 6627 of *LNCS*. Springer-Verlag Berlin Heidelberg, 2011, pp. 244–258.

[25] CORMEN, T. H., STEIN, C., RIVEST, R. L., AND LEISERSON, C. E. *Introduction to Algorithms*, 2 ed. McGraw-Hill Higher Education, 2001.

[26] COUTINHO, A. E. V. B. *Similarity-Based Test Suite Reduction in the Context of Model-Based Testing*. PhD thesis, Universidade Federal de Campina Grande, 2015.

[27] COUTINHO, A. E. V. B., CARTAXO, E. G., AND DE LIMA MACHADO, P. D. Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. *Software Quality Journal* (2014).

[28] DE ARAÚJO, J. D. S., CARTAXO, E. G., DE OLIVEIRA NETO, F. G., AND MACHADO, P. D. L. Controlando a diversidade e a quantidade de casos de teste na geração automática a partir de modelos com loop. In *Proceedings of the 6th Brazilian Workshop on Systematic and Automated Software Testing* (Natal, Brazil, 2012), SAST, pp. 63–72.

[29] DEMILLO, R. A., LIPTON, R. J., AND SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. *IEEE Computer 11*, 4 (Apr. 1978), 34–41.

[30] EL-FAR, I. K., AND WHITTAKER, J. A. Model-based software testing. In *Encyclopedia of Software Engineering*, J. J. Marciniak, Ed., vol. 1. John Wiley & Sons, Inc., 2002, pp. 825–837.

[31] EN-NOUAARY, A. Test selection criteria for real-time systems modeled as timed input-output automata. *International Journal of Web Information Systems 3*, 4 (2007), 279–292.

[32] FARAHANI, R. Z., ASGARI, N., HEIDARI, N., HOSSEININIA, M., AND GOH, M. Covering problems in facility location: A review. *Computers & Industrial Engineering 62*, 1 (2012), 368–407.

[33] FRASER, G., AND WOTAWA, F. Redundancy based test-suite reduction. In *Fundamental Approaches to Software Engineering*, M. B. Dwyer and A. Lopes, Eds., vol. 4422 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 291–305.

[34] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[35] GOODENOUGH, J. B., AND GERHART, S. L. Toward a theory of test data selection. *IEEE Transactions on Software Engineering 1*, 2 (June 1975), 156–172.

[36] GOTLIEB, A., AND MARIJAN, D. FLOWER: optimal test suite reduction as a network maximum flow. In *Proceedings of the 17th International Symposium on Software Testing and Analysis* (2014), ISSTA 2014, pp. 171–180.

[37] GRIESKAMP, W. Multi-paradigmatic model-based testing. In *Formal Approaches to Software Testing and Runtime Verification*, K. Havelund, M. N. nez, G. Roşu, and B. Wolff, Eds., vol. 4262 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 1–19.

[38] HARROLD, M. J., GUPTA, R., AND SOFFA, M. L. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology 2*, 3 (1993), 270–285.

[39] HEIMDAHL, M. P., AND GEORGE, D. Test-suite reduction for model based tests: Effects on test quality and implications for testing. In *Proceedings of the 19th International Conference on Automated Software Engineering* (2004), ASE, pp. 176–185.

[40] HESSEL, A. *Model-Based Test Case Selection and Generation for Real-Time Systems*. PhD thesis, Uppsala University, Uppsala, Sweden, 2007.

[41] JEANNET, B., JÉRON, T., RUSU, V., AND ZINOVIEVA, E. Symbolic test selection based on approximate analysis. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2005), pp. 349–364.

[42] JONES, J., AND HARROLD, M. Test-suite reduction and prioritization for Modified Condition/Decision Coverage. *IEEE Transactions on Software Engineering 29*, 3 (Mar. 2003), 195–209.

[43] JONES, M. What really happened on mars rover pathfinder. *The Risks Digest 19*, 49 (1997).

[44] JOURDAN, G.-V., RITTHIRUANGDECH, P., AND URAL, H. Test suite reduction based on dependence analysis. In *Computer and Information Sciences – ISCIS 2006*, A. Levi, E. Savaş, H. Yenigün, S. Balcısoy, and Y. Saygın, Eds., vol. 4263 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 1021–1030.

[45] KRICHEN, M., AND TRIPAKIS, S. Conformance testing for real-time systems. *Formal Methods in System Design 34*, 3 (2009), 238–304.

[46] LAN, G., DEPUY, G. W., AND WHITEHOUSE, G. E. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research 176*, 3 (2007), 1387–1403.

[47] LAPLANTE, P. A., AND OVASKA, S. J. *Real-time systems design and analysis: tools for the practitioner*, 4 ed. Wiley-IEEE Press, Hoboken, N.J., 2012.

[48] LIN, J.-W., HUANG, C.-Y., AND LIN, C.-T. Test suite reduction analysis with enhanced tie-breaking techniques. In *Proceedings of the 4th IEEE International Conference on Management of Innovation and Technology.* (2008), ICMIT 2008, pp. 1228–1233.

[49] LORBER, F., ROSENMANN, A., NIČKOVIĆ, D., AND AICHERNIG, B. K. Bounded determinization of timed automata with silent transitions. *Real-Time Systems 53* (2017), 291–326.

[50] MA, Y.-S., OFFUTT, J., AND KWON, Y. R. MuJava: An automated class mutation system. *Software Testing, Verification and Reliability 15* (2005), 97–133.

[51] MASTERINCOMPUTERSCIENCE.COM. 5 of the most catastrophic computer glitches in recent history. http://www.masterincomputerscience.com/articles/top-5-computer-glitches.html. Accessed June 6, 2015.

[52] MINKEL, J. The 2003 northeast blackout–five years later. http://www.scientificamerican.com/article/2003-blackout-five-years-later. Accessed June 6, 2015.

[53] MONTGOMERY, D. C. *Design and Analysis of Experiments.* John Wiley & Sons, Inc., Hoboken, NJ, 2013.

[54] MORAES, A., ANDRADE, W. L., AND MACHADO, P. D. A family of test selection criteria for timed input-output symbolic transition system models. *Science of Computer Programming 126* (2016), 52–72.

[55] NEUMANN, P. G. Cause of AT&T network failure. *The Risks Digest 9*, 62 (1990).

[56] OURIQUES, J. F. S., CARTAXO, E. G., AND MACHADO, P. D. L. Revealing influence of model structure and test case profile on the prioritization of test cases in the context of model-based testing. *Journal of Software Engineering Research and Development 3* (2015), 1–28.

[57] PANICHELLA, A., OLIVETO, R., PENTA, M. D., AND LUCIA, A. D. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering 41*, 4 (2015), 358–383.

[58] PELESKA, J. Industrial-strength model-based testing - state of the art and current challenges. In *Proceedings 8th Workshop on Model-Based Testing* (2010), pp. 3–28.

[59] PRENNINGER, W., EL-RAMLY, M., AND HORSTMANN, M. Case studies. In *Model-Based Testing of Reactive Systems*, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472 of *Lecture Notes In Computer Science*. Springer-Verlag Berlin Heidelberg, 2005, pp. 439–461.

[60] PRETSCHNER, A., SLOTOSCH, O., AIGLSTORFER, E., AND KRIEBEL, S. Model-based testing for real. *International Journal on Software Tools for Technology Transfer 5*, 2 (2004), 140–157.

[61] RAPPS, S., AND WEYUKER, E. J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering 11*, 4 (1985), 367–375.

[62] ROTHERMEL, G., HARROLD, M. J., OSTRIN, J., AND HONG, C. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings of the 6th International Conference on Software Maintenance* (1998), ICSM 1998, pp. 34–43.

[63] ROTHERMEL, G., HARROLD, M. J., VON RONNE, J., AND HONG, C. Empirical studies of test-suite reduction. *Software Testing, Verification & Reliability 12*, 4 (2002), 219–249.

[64] ROTHERMEL, G., UNTCH, R. H., CHU, C., AND HARROLD, M. J. Test case prioritization: an empirical study. In *Proceedings of 7th International Conference on Software Maintenance* (1999), ICSM, pp. 179–188.

[65] RUSU, V., DU BOUSQUET, L., AND JÉRON, T. An approach to symbolic test generation. In *Proceedings of the 2nd International Conference on Integrated Formal Methods* (2000), pp. 338–357.

[66] SIMÃO, A., PETRENKO, A., AND MALDONADO, J. C. Comparing finite state machine test. *IET Software 3*, 2 (apr 2009), 91–105.

[67] TRETMANS, J. Model based testing with labelled transition systems. In *Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers*, R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Springer, 2008, pp. 1–38.

[68] TRETMANS, J. Model-based testing and some steps towards test-based modelling. In *Proceedings of 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems* (2011), pp. 297–326.

[69] UTTING, M., AND LEGEARD, B. *Practical Model Based Testing: A Tools Approach.* Elsevier, San Francisco, CA, USA, 2007.

[70] WILCOX, R. R. *Fundamentals of Modern Statistical Methods: Substantially Improving Power and Accuracy.* Springer Verlag, 2010.

[71] WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M. C., REGNELL, B., AND WESSLÉN, A. *Experimentation in Software Engineering*. Springer, New York, NY, USA, 2012.

[72] WONG, W. E., HORGAN, J. R., LONDON, S., AND MATHUR, A. P. Effect of test set minimization on fault detection effectiveness. *Software: Practice and Experience 28*, 4 (1998), 347–369.

[73] WONG, W. E., HORGAN, J. R., MATHUR, A. P., AND PASQUINI, A. Test set size minimization and fault detection effectiveness: A case study in a space application. *Journal of Systems and Software 48*, 2 (1999), 79–89.

[74] YOO, S., AND HARMAN, M. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability 22*, 2 (2012), 67–120.

[75] ZHENG, M., ALAGAR, V., AND ORMANDJIEVA, O. Automated generation of test suites from formal specifications of real-time reactive systems. *Journal of Systems and Software 81*, 2 (feb 2008), 286–304.

[76] ZHU, H., HALL, P. A. V., AND MAY, J. H. R. Software unit test coverage and adequacy. *ACM Computing Surveys 29*, 4 (1997), 366–427.

Appendix A

Results

A.1 Refilling Machine

Table A.1: Refilling Machine — Descriptive statistics of number of test cases by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	3.0	4.0	4.0	4.3	4.0	10.0	1.2	0.0
ASP	2.0	3.0	3.0	3.1	3.0	8.0	0.8	0.0
ADUP	1.0	1.0	1.0	1.3	1.0	5.0	0.6	0.0
AU	1.0	1.0	1.0	1.3	1.0	5.0	0.6	0.0
ATP	1.0	1.0	1.0	1.3	1.0	5.0	0.6	0.0
ACG	1.0	1.0	1.0	1.2	1.0	3.0	0.5	0.0
AT	1.0	1.0	1.0	1.2	1.0	3.0	0.5	0.0
AOA	1.0	1.0	1.0	1.1	1.0	3.0	0.4	0.0
ACR	1.0	1.0	1.0	1.1	1.0	3.0	0.4	0.0
AL	1.0	1.0	1.0	1.1	1.0	3.0	0.3	0.0
AD	1.0	1.0	1.0	1.0	1.0	2.0	0.2	0.0
AIA	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.2: Refilling Machine — Descriptive statistics of number of test cases by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	1.0	1.0	2.0	2.4	3.0	10.0	1.6	2.0
G	1.0	1.0	1.0	1.4	1.0	4.0	1.0	0.0
GE	1.0	1.0	1.0	1.4	1.0	4.0	1.0	0.0
GRE	1.0	1.0	1.0	1.4	1.0	4.0	1.0	0.0
HGS	1.0	1.0	1.0	1.2	1.0	3.0	0.6	0.0

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.3: Refilling Machine — Descriptive statistics of execution time by criterion.

Criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	107.0	130.5	146.0	146.2	148.0	309.5	31.0	17.5
ASP	68.5	95.5	112.5	108.6	118.0	236.0	24.4	22.5
ADUP	28.0	39.0	39.0	44.5	39.0	138.5	13.9	0.0
AU	28.0	39.0	39.0	44.5	39.0	138.5	13.9	0.0
ACR	17.5	28.5	39.0	40.0	47.0	99.5	11.6	18.5
ACG	19.0	28.0	30.0	36.9	39.0	131.5	13.2	11.0
AT	19.0	28.0	30.0	36.9	39.0	131.5	13.2	11.0
ATP	28.0	28.0	30.0	35.5	30.0	153.0	16.5	2.0
AL	9.5	26.5	30.0	33.8	45.5	91.0	13.0	19.0
AD	8.0	26.5	30.0	32.5	42.5	72.5	11.5	16.0
AOA	11.0	23.5	30.0	31.9	39.0	123.0	14.6	15.5
AIA	1.5	20.0	28.0	27.7	39.0	50.5	13.6	19.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.4: Refilling Machine — Descriptive statistics of execution time by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	1.5	38.0	54.5	69.3	78.5	309.5	49.7	40.5
GRE	1.5	28.0	39.0	49.1	43.5	148.0	37.5	15.5
G	1.5	28.0	39.0	48.5	43.5	149.5	36.7	15.5
GE	1.5	28.0	39.0	48.5	43.5	149.5	36.7	15.5
HGS	1.5	28.0	39.0	42.5	43.5	107.0	23.9	15.5

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.5: Refilling Machine — Descriptive statistics of number of killed mutants by criterion.

Criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	131.0	131.0	131.0	131.0	131.0	131.0	0.0	0.0
ASP	131.0	131.0	131.0	131.0	131.0	131.0	0.0	0.0
ATP	130.0	130.0	131.0	130.6	131.0	131.0	0.5	1.0
ACG	125.0	126.0	130.0	129.2	131.0	131.0	2.1	5.0
AT	125.0	126.0	130.0	129.2	131.0	131.0	2.1	5.0
ADUP	126.0	126.0	126.0	126.8	126.0	131.0	1.8	0.0
AU	126.0	126.0	126.0	126.8	126.0	131.0	1.8	0.0
AOA	112.0	113.0	118.0	121.8	130.0	131.0	7.4	17.0
ACR	107.0	112.0	112.0	120.3	130.0	131.0	8.7	18.0
AL	93.0	112.0	113.0	116.4	126.0	131.0	9.9	14.0
AD	79.0	105.0	112.0	110.8	118.0	131.0	13.6	13.0
AIA	75.0	100.0	112.0	107.3	117.0	131.0	15.0	17.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.6: Refilling Machine — Descriptive statistics of number of killed mutants by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	75.0	126.0	131.0	125.2	131.0	131.0	10.6	5.0
G	75.0	117.0	126.0	123.0	131.0	131.0	10.8	14.0
GE	75.0	117.0	126.0	123.0	131.0	131.0	10.8	14.0
GRE	75.0	118.0	126.0	123.0	131.0	131.0	10.7	13.0
HGS	75.0	117.0	126.0	122.9	131.0	131.0	10.8	14.0

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

A.2 Burglar Alarm System

Table A.7: Burglar Alarm System — Descriptive statistics of number of test cases by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	22.0	22.0	22.0	22.2	22.0	26.0	0.5	0.0
ASP	12.0	12.0	12.0	13.4	12.0	24.0	3.0	0.0
ATP	9.0	9.0	9.0	9.2	9.0	12.0	0.5	0.0
ADUP	7.0	7.0	7.0	7.3	7.0	12.0	0.7	0.0
AT	5.0	5.0	5.0	5.3	5.0	8.0	0.6	0.0
AU	5.0	5.0	5.0	5.3	5.0	8.0	0.6	0.0
ACG	5.0	5.0	5.0	5.2	5.0	7.0	0.6	0.0
AD	5.0	5.0	5.0	5.0	5.0	6.0	0.1	0.0
AOA	4.0	4.0	4.0	4.2	4.0	7.0	0.5	0.0
ACR	3.0	3.0	3.0	3.1	3.0	6.0	0.3	0.0
AL	2.0	2.0	2.0	2.2	2.0	6.0	0.6	0.0
AIA	1.0	1.0	1.0	1.0	1.0	2.0	0.1	0.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.8: Burglar Alarm System — Descriptive statistics of number of test cases by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	1.0	4.0	6.0	8.1	9.0	26.0	6.3	5.0
G	1.0	3.8	5.0	6.7	7.5	22.0	5.4	3.8
GE	1.0	3.8	5.0	6.7	7.5	22.0	5.4	3.8
GRE	1.0	3.8	5.0	6.7	7.5	22.0	5.4	3.8
HGS	1.0	3.8	5.0	6.7	7.5	22.0	5.4	3.8

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.9: Burglar Alarm System — Descriptive statistics of execution time by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	60550.0	60550.0	60550.0	60550.0	60550.0	60620.0	11.2	0.0
ASP	30300.0	30300.0	30300.0	34410.0	30300.0	60580.0	8584.9	0.0
ADUP	20180.0	20180.0	20180.0	20910.0	20180.0	34250.0	2077.6	0.0
ATP	15720.0	15720.0	16220.0	16460.0	16220.0	26300.0	1504.6	500.0
AOA	6100.0	10120.0	14100.0	12590.0	15100.0	23650.0	2965.2	4975.0
ACR	5575.0	10080.0	13580.0	11980.0	14580.0	15620.0	2853.5	4500.0
ACG	10120.0	10120.0	10120.0	10960.0	10120.0	20180.0	2151.9	0.0
AT	10120.0	10120.0	10120.0	10950.0	10120.0	24700.0	2151.3	0.0
AU	10120.0	10120.0	10120.0	10950.0	10120.0	24700.0	2151.3	0.0
AD	10120.0	10120.0	10120.0	10130.0	10120.0	10150.0	0.5	0.0
AL	5050.0	5050.0	5050.0	5552.0	5050.0	10120.0	1432.4	0.0
AIA	25.0	525.0	4525.0	2559.0	5025.0	5050.0	2263.2	4500.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.10: Burglar Alarm System — Descriptive statistics of execution time by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	25.0	10120.0	14650.0	20000.0	20700.0	60620.0	17126.9	10575.0
GRE	25.0	10120.0	11850.0	17040.0	17210.0	60550.0	14839.9	7087.5
G	25.0	10120.0	10120.0	16540.0	17210.0	60550.0	14998.7	7087.5
GE	25.0	10120.0	10120.0	16540.0	17210.0	60550.0	14998.7	7087.5
HGS	25.0	10120.0	10120.0	16540.0	17210.0	60550.0	14999.3	7087.5

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.11: Burglar Alarm System — Descriptive statistics of number of killed mutants by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	481.0	481.0	481.0	481.0	481.0	481.0	0.0	0.0
ASP	481.0	481.0	481.0	481.0	481.0	481.0	0.0	0.0
ATP	481.0	481.0	481.0	481.0	481.0	481.0	0.0	0.0
ADUP	472.0	472.0	481.0	476.6	481.0	481.0	4.5	9.0
AT	472.0	472.0	481.0	476.6	481.0	481.0	4.5	9.0
AU	472.0	472.0	481.0	476.6	481.0	481.0	4.5	9.0
ACG	449.0	472.0	473.0	476.4	481.0	481.0	4.7	9.0
AD	423.0	449.0	458.0	456.8	459.0	481.0	13.6	10.0
AOA	346.0	346.0	388.0	388.9	431.0	481.0	33.1	85.0
ACR	235.0	305.0	326.0	325.1	326.0	433.0	30.3	21.0
AL	270.0	300.0	301.0	309.2	301.0	481.0	31.9	1.0
AIA	163.0	181.0	199.0	200.2	217.0	279.0	22.8	36.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.12: Burglar Alarm System — Descriptive statistics of number of killed mutants by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	163.0	390.0	472.0	426.7	481.0	481.0	85.9	91.0
G	163.0	367.0	472.0	418.5	481.0	481.0	91.4	114.0
GE	163.0	367.0	472.0	418.5	481.0	481.0	91.4	114.0
HGS	163.0	367.0	472.0	418.4	481.0	481.0	91.5	114.0
GRE	163.0	341.0	472.0	413.5	481.0	481.0	94.8	140.0

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

A.3 Automated Speed Limiter

Table A.13: Automated Speed Limiter — Descriptive statistics of number of test cases by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	47.0	47.0	47.0	50.8	48.0	74.0	7.2	1.0
ASP	20.0	20.0	20.0	21.1	20.0	31.0	2.3	0.0
ATP	17.0	17.0	17.0	17.4	17.0	23.0	0.9	0.0
ADUP	8.0	8.0	8.0	8.5	9.0	12.0	0.9	1.0
AU	8.0	8.0	8.0	8.5	9.0	12.0	0.9	1.0
AD	7.0	7.0	7.0	7.6	8.0	12.0	1.1	1.0
AT	5.0	5.0	5.0	5.5	5.0	10.0	1.0	0.0
ACR	3.0	3.0	4.0	4.0	4.0	9.0	1.3	1.0
AIA	1.0	1.0	1.0	1.3	1.0	4.0	0.6	0.0
ACG	1.0	1.0	1.0	1.2	1.0	3.0	0.5	0.0
AL	1.0	1.0	1.0	1.0	1.0	2.0	0.2	0.0
AOA	1.0	1.0	1.0	1.0	1.0	2.0	0.2	0.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.14: Automated Speed Limiter — Descriptive statistics of number of test cases by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	1.0	2.0	8.0	13.3	13.2	74.0	17.2	11.2
G	1.0	1.0	6.0	10.1	11.0	49.0	12.8	10.0
GE	1.0	1.0	6.0	10.1	11.0	49.0	12.8	10.0
GRE	1.0	1.0	6.0	10.0	11.0	50.0	12.8	10.0
HGS	1.0	1.0	6.0	9.9	10.2	47.0	12.7	9.2

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.15: Automated Speed Limiter — Descriptive statistics of execution time by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	880.0	916.0	926.0	988.6	943.0	1436.0	139.6	27.0
ASP	381.0	393.0	402.0	421.4	417.0	614.0	45.8	24.0
ATP	320.0	333.0	342.0	345.2	352.0	450.0	18.8	19.0
ADUP	150.0	155.0	163.0	171.3	181.0	256.0	19.2	26.0
AU	150.0	155.0	163.0	171.3	181.0	256.0	19.2	26.0
AD	130.0	135.0	144.0	153.2	162.0	250.0	22.6	27.0
AT	86.0	96.0	104.0	109.4	105.0	200.0	19.0	9.0
ACR	57.0	57.0	76.0	76.9	77.0	197.0	25.3	20.0
ACG	19.0	20.0	28.0	28.4	29.0	67.0	9.6	9.0
AIA	19.0	19.0	19.0	24.6	20.0	95.0	12.0	1.0
AL	10.0	19.0	19.0	20.8	20.0	47.0	5.2	1.0
AOA	10.0	19.0	19.0	20.8	20.0	47.0	5.2	1.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.16: Automated Speed Limiter — Descriptive statistics of execution time by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	10.0	47.0	164.0	260.7	272.2	1436.0	333.5	225.2
G	10.0	20.0	122.0	199.7	236.8	963.0	249.0	216.8
GE	10.0	20.0	122.0	199.7	236.8	963.0	249.0	216.8
GRE	10.0	20.0	123.0	199.5	229.5	981.0	249.8	209.5
HGS	10.0	20.0	122.0	195.3	231.0	945.0	245.9	211.0

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.

Table A.17: Automated Speed Limiter — Descriptive statistics of number of killed mutants by criterion.

criterion	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
AOLP	278.0	278.0	278.0	280.4	284.0	284.0	2.9	6.0
ASP	277.0	278.0	278.0	279.6	284.0	284.0	2.8	6.0
ATP	277.0	278.0	278.0	279.5	283.0	284.0	2.8	5.0
ADUP	270.0	274.0	275.0	274.8	275.0	284.0	1.0	1.0
AU	270.0	274.0	275.0	274.8	275.0	284.0	1.0	1.0
AD	267.0	272.0	273.0	273.1	273.0	283.0	1.5	1.0
AT	212.0	237.0	241.0	242.5	249.0	276.0	9.6	12.0
ACR	208.0	235.0	243.0	240.2	244.0	276.0	7.6	9.0
AIA	184.0	186.0	200.0	199.0	205.0	247.0	11.8	19.0
ACG	179.0	189.0	199.0	197.7	206.0	249.0	11.8	17.0
AL	173.0	185.0	193.0	193.9	205.0	233.0	10.8	20.0
AOA	173.0	185.0	193.0	193.9	205.0	233.0	10.8	20.0

Notes — Number of test suites for each criterion $N = 5,000$. Captions — ACG: All-Clock-Guards, ACR: All-Clock-Resets, AD: All-Defs, ADUP: All-DU-Paths, AIA: All-Input-Actions, AL: All-Locations, AOA: All-Output-Actions, AOLP: All-One-Loop-Paths, ASP: All-Simple-Paths, AT: All-Transitions, ATP: All-Transition-Pairs, AU: All-Uses.

Table A.18: Automated Speed Limiter — Descriptive statistics of number of killed mutants by technique.

technique	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. dev.	IQR
R	173.0	213.0	272.0	247.8	277.0	284.0	34.8	64.0
HGS	173.0	203.0	270.5	243.9	275.5	284.0	37.4	72.5
G	173.0	203.0	265.0	243.0	276.2	284.0	37.7	73.2
GE	173.0	203.0	265.0	243.0	276.2	284.0	37.7	73.2
GRE	173.0	205.0	267.5	243.0	275.5	284.0	37.5	70.5

Notes — Number of test suites for each technique $N = 12,000$. Captions — R: Random, G: Greedy, GE: Greedy-Essential, GRE: Greedy-Redundancy-Essential, HGS: Harrold, Gupta and Soffa.