An In-depth Study of Adaptive Random Testing for Testing Program with Complex Input Types

Arlinta Christy Barus

Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria 3122, Australia

A thesis submitted for the degree of PhD in 2010

Abstract

Adaptive random testing (ART) is a testing method aimed to improve the performance of random testing. The initial studies of ART merely focussed on programs with numeric input type. However, real life software involves more complex types of inputs. This thesis examines the application of ART on programs with complex input types.

Firstly, the application of a distance-based ART technique, namely FSCS-ART, on real programs with complex input types is conducted. This is an extension of two previous studies which developed new approaches to apply FSCS-ART on such programs. However, those studies have not applied the approaches to test any real programs. Here, they are applied to test four real programs with such input types.

An exclusion-based ART, namely Restricted Random Testing (RRT), has been demostrated effective to test programs with numeric input type. We are motivated to also apply RRT on programs with complex input types. Some new approaches are introduced to adjust the use of RRT on such programs.

Previous studies of ART have identified the regularity of failure causing inputs (FCIs), which are inputs that cause a program under test fails, in the numeric context. They have also observed how ART techniques behaved towards the identified patterns. This thesis investigates the extension of the study to more complex input types. Using the results of the relevant experiments in this thesis, we investigate the FCI patterns in a program with complex input types and analyse how the ART techniques behave towards the identified patterns.

When applying ART, the existence of a test oracle is required to identify the detection of failures. Test oracle is more difficult to obtain when the program under test has more complex input types. Therefore, in this thesis we propose the use of Metamorphic Testing (MT) as the test oracle for applying ART on a program with complex input types. We examine how effective is MT compared to a test oracle which has been commonly used in previous ART studies.

At last, an application of ART on other area of testing is conducted. We apply ART as the source test case selection strategy of MT. We investigate how effective ART compared to RT in selecting the source test cases of the implemented metamorphic relations.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Tsong Yueh Chen for the excellent support, guidance, and patience given to me during the completion of my PhD study. I would also like to thank Dr. Robert Merkel as my associate supervisor for his great support during my PhD study. I am also grateful to Dr. Diana Kuo for her great assistance and care given to me. Furthermore, I would like to thank all of my colleagues in Centre for Software Analysis and Testing for showing care and giving encouragement to me.

Then, I would like to give my best gratitude to Bapak General Luhut B. Pandjaitan and Ibu for their great support and attention given to me. I also thank Bapak Ir. Patuan Simatupang, MCRP and Ibu, Bapak Prof. Saswinadi Sasmojo, and Ibu Dr. Inggriani for the great guidance, help, patience, and encouragement given to me during my PhD study. I would like to thank all staffs and lectures in Yayasan Del and Politeknik Informatika Del for also giving me a lot of support.

Finally, I would like to thank my family members: my beloved husband and my lovely son for always giving the great understanding, prayers, and love during my study; my beloved parents, brothers, and sisters-in-law for continuing to pray and support me which I believe contributing so much for the completion of my study.

Declaration

I declare that the work presented in this thesis is original and my own work, and that it has not been submitted for any other degree. To the best of my knowledge, this contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Signed:

Dated:

Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1

Introduction

1.1 What is Software Testing?

In software development, the role of the testing phase is to check whether the software meets the needs of the customers and users as written in its requirement specifications [1]. In commercial organizations, the checking process – better known as software testing – is viewed as a process to deliver higher quality software, more satisfied users and lower maintenance costs. In the scientific area, it is a process to produce more accurate and reliable results. Failure to conduct effective testing gives the opposite results such as low quality products, unsatisfied users, and increased maintenance costs [2]. Hence, even testing is an expensive activity, omitting this phase could lead to much higher cost. This is obvious in the critical systems which involve human safety where not conducting intensive testing before launching such systems may endanger human lives. It also applies in commercial systems. The earlier the errors are discovered, the earlier the program is fixed. Consequently, the lower the cost will be. Errors discovered when the system has been put into operation can be extremely damaging.

As testing activity is expensive, a testing activity must expose the quality or the reliability of the program effectively. The quality or the reliability is best improved by finding the errors in the program and enabling these errors to be fixed. In other words, testers should not test programs to show that they work; rather, they should test them to reveal errors inside the programs.

The argument has been also presented by Parrington and Roper in their book [2]. They argued that software testing should not be identified as a process to demonstrate that errors were not present in the software. They agreed with Myers [3] who stated that software testing was a process to execute a program under test with a purpose to find as many errors as possible. This was applied with a belief that the program under test contained errors, unless it had been mathematically proved that

it was correct.

Parrington and Roper [2] also called software testing as an active process. They argued that if no errors had been revealed after running a test, we should not passively conclude that no errors existed. It merely showed that we had failed to find errors which we believed that the software contained.

Myers [3] discussed the psychological importance of appropriate goals when testing a program. With a goal to show that a program has no errors, we tend to find *test cases* – elements of the input domain which are selected for program execution – with a low probability of revealing errors. On the other hand, with a goal to demonstrate that the program contains errors, we would be more selective to choose test cases having a higher probability of finding errors. In other words, software testing is more appropriately viewed as the destructive process of attempting to find the errors in a program. Therefore, test case selection strategy has a significant role in software testing.

1.2 Problems in Software Testing

There are two problems of software testing discussed in this thesis. The first is the *reliable test set problem*. Given a set of test inputs T which is a subset of the input domain D to test program P, T is reliable if it reveals that P contains an error whenever P is incorrect. However, the size of D is large most of the time. Therefore, it is difficult to find manageable size of T. This situation is known as the reliable test set problem [4, 5, 6]. Many testing strategies have been developed to deliver a reliable T. However, Howden proved that there was no such effective testing strategy to find T which was reliable for all programs [4]. To achieve more reliable T, it is common to combine more than one testing strategy as each is designed for different testing objective.

The second problem is the *oracle problem*. Suppose t is an element of the input domain D of P, the program under test. The *test oracle* verifies the correctness of P by comparing the actual output of P, which is P(t), to the expected outcome of P. Here, the expected outcome refers to the correct output of P based on its requirement specification. We conclude that P contains an error if the actual and the expected outputs are different. However, situations exist where oracles are not readily available. For example, it is difficult to verify the results of a program implementing a heuristic method as this method does not guarantee to deliver optimal solution. Hence, it is very difficult to verify the correctness of its implementation [7]. This difficulty is known as the oracle problem. Weyuker [8] explained that the oracle problem occurs if : (1) there does not exist an oracle or (2) it is theoretically possible, but practically too difficult to determine the correct output. *Metamorphic*

Testing (MT) is one of testing strategies aimed to alleviate the oracle problem [9].

1.3 Test Case Selection Strategy

As mentioned, test case selection strategy has a very important role in the software testing process. A good strategy reveals errors of the program under test, as the goal of software testing, therefore providing partial solution to the reliable test set problem. There are many approaches in performing test case selection. However, in general, they can be classified into two major groups: *black-box testing* [3] and *white-box testing* [10].

Black-box testing (also known as data-driven or input/output-driven technique) is a testing method which selects the test cases fully based on the program specifications. It ignores any knowledge of the internal behaviour of the program under test and the structure behaviour of the program code. Rather, the technique only considers the input and the output of the program to identify whether the program behaves according to its specifications. Some examples of black-box testing include *random testing*, *equivalence partitioning* and *boundary-value analysis* [3].

On the other hand, white-box testing (also known as logic-driven testing) is a testing method where the test cases are selected based on the examination of the program code or the logic and the internal behaviour of the program. This approach often ignores the program specifications. Testing techniques with code coverage criteria such as *statement coverage* and *decision/condition coverage* are examples of white-box testing [11].

1.3.1 Random Testing (RT)

Random testing (RT) is a testing strategy which selects test cases randomly from all possible input values. According to Myers [3], this is the poorest technique of all. He argued that a collection of randomly selected test cases had little chance of being optimal, or close to optimal, in terms of the probability of detecting the most errors. He advised that tester should select test cases more intelligently than RT.

On the other hand, compared to other techniques, the automated implementation of RT is much more cost effective because it is computationally inexpensive. Based on this advantage, Duran and Wiorkowski [12] were interested to investigate RT further. They conducted a study to compare the effectiveness of RT and path-coverage testing in the context of analysing software reliability. They found a surprising result which is RT could be sometimes better. Based on the study in [12] , Duran and Nafos [13] conducted an empirical study of the fault-finding capabilities of RT. They showed that RT could find errors a reasonably high proportion of the

time. The further study conducted by Duran and Nafos [14] also showed that the effectiveness of RT was not much different from partition testing — a testing strategy which divides the input domain into several partitions and assigns test cases amongst them — whilst the latter technique was much more expensive than RT. Yoshikawa et al. [15] also reported that RT had been successful to test Java virtual machines as well as Slutz [16] in testing SQL implementation. Therefore, studies involving the comparison of RT and other testing strategies still attract great interest.

1.3.2 Adaptive Random Testing (ART)

Adaptive Random Testing (ART) developed by Chen *et al.* [17] is a testing method aimed to improve the performance of RT. The approach is based on the intuition that two test cases located close to each other are more likely to have the same failure behaviour than two test cases located far from each other. Accordingly, ART techniques attempt to select test cases as evenly spread as possible across the program's input domain. Hence, by choosing a more even spread of test cases, they believed that fewer test cases would be required to identify the first failure.

Based on the approaches, ART techniques can be classified at least into the following groups: best of all, exclusion, and partitioning approach [18]. Best of all methods select a best candidate from potential test cases to be executed as the next test case. The selection attempts to find one with the largest distance to all executed test cases. Thus, this group is also known as ART by distance. The most commonly used ART, Fixed Size Candidate Set (FSCS-ART) [17] belongs to this group. The exclusion approach excludes test cases located inside exclusion regions. These regions are defined prior to each selection by allocating same sizes and shapes of areas around all executed test cases. One typical example of this group is ART by restriction [19]. In the partitioning approach, the input domain of the program under test is defined into some subdomains based on a certain rule. Then the next test case is chosen from one of the subdomains. Random Partitioning [20] is an example of the this group.

ART methods have been a subject of considerable research interest recently. Liu and Zhu [21] conducted some experimental work to evaluate of the reliability of the existing ART methods. They discovered that ART significantly improved the reliability of fault detecting capability. They found that there were two factors of the program under test that affected the reliability of ART: the failure rate and the regularity of the failure domain. They concluded that the increase of these two factors would increase the the reliability of ART.

Tappenden and Miller [22] applied ART on their search algorithm. They introduced an ART-based search algorithm, called evolutionary Adaptive Random

Testing (eART). Following the basic purpose of ART, it is aimed to increase the effectiveness of an RT-based search algorithm. Their proposed algorithm adopted a well defined evolutionary algorithm based on the natural selection process of Darwin, called the genetic algorithm. They conducted an extensive simulation study to compare the performance of eART and existing strategies.

Tse *et al.* [23] extended the use of ART in the regression testing area. As regression testing is about rearranging the execution order of test cases to improve their effectiveness, they use an ART-based approach to do the test case rearrangement. Their new techniques could be classified into a new family of coverage-based ART techniques. Their related experimental work showed that their new techniques were statistically superior to the RT-based technique in detecting faults and even one of them is consistently comparable to some of the best existing coverage-based prioritization techniques whilst imposing less selection overhead.

1.4 Contribution of this Thesis

The use of ART on programs with complex input types is the main theme of this thesis. The initial studies of ART have only used programs with numeric input type as the subjects [24, 25, 26]. However, real life software involves more complex types of inputs. Here, complex types refer to any input types which are not purely numeric. In this thesis, such types are mostly represented by text or collections of any characters which are not number. This thesis studies the extension of existing ART strategies to programs having such input types. The previously relevant studies of FSCS-ART conducted by Merkel [27] and Kuo [18] are enhanced and verified against real life programs.

Next, some new ideas of how to extend the use of Restricted Random Testing (RRT) – an exclusion-based ART technique – to test program with complex input types are introduced. ART by exclusion has been demostrated effective for testing programs with numeric inputs. Motivated by this, we attempt to apply RRT on programs with complex input types, as another alternative to ART by distance. To simplify the comparison of the performances of FSCS-ART and RRT, in this study we use the program that has been used in FSCS-ART experiments.

The next contribution of this thesis is the identification of failure causing input patterns in programs with complex input types. In the numeric context, Chan *et al.* [28] observed the variety of inputs that commonly cause programs fail, or referred to as failure causing inputs (FCIs). They identified some regularities of these inputs. Based on their observation, the FCI patterns are classified into three groups: the block pattern, the strip pattern, and the point pattern. The first pattern represents a single compact and contiguous region of FCI in the program under test's input

domain. The second pattern is for a thin and long contiguous region, and the last one is for many small regions where each region contains one or a very few FCI. For the numeric context, ART techniques perform well in the first and second pattern as it supports the basic intuition of ART that test cases located close to each other are more likely to have similar failure behaviours than test cases located far from each other. In this thesis, we investigate the regularity of FCI in programs with complex input types. Accordingly, we observe how our ART techniques behave towards each of the identified FCI patterns.

In previous studies of ART, the existence of a test oracle has been assumed and usually implemented by defining an unmodified program as the test oracle and a defined mutated version as the program under test. By using a test case from the input domain as the input, if the output of the program is different to the output of the test oracle, then the input is identified as a failure causing input. Such approach is only feasible for evaluation of a testing method. In the reality, we need a real test oracle. However, the more complex the program is, the more difficult it can be to find a test oracle. Therefore, in this thesis we propose the use of metamorphic testing (MT) as an alternative test oracle for applying ART on programs with complex input types. We investigate the effectiveness of MT as a test oracle compared to the "test oracle" commonly used for that purpose.

As Tse *et al.* [23] showed that ART was useful in the test case prioritization area, in this thesis we also present the use of ART techniques in other area of testing. Accordingly, we apply ART techniques as the source test case selection strategies in conducting MT, particularly on testing programs with complex input types.As the experimental setting, the testing stops whenever the implemented metamorphic relation (MR) is violated. The number of test cases executed before the violation of the MR is used to evaluate the effectiveness of ART techniques. We investigate how effective is ART compared to the random selection, which was commonly used in the previous studies of MT to select the source test cases from the input domain. This measurement is intuitively similar to the *F-Measure* which has been widely used in ART studies.

1.5 Organization of the Thesis

This thesis has following structure. Chapter 2 discusses the theoretical frameworks, basic concepts, and methodologies used in the following chapters. This chapter describes the main concept of ART, MT, and evaluation methods used as the metrics in this thesis. Then Chapter 3 describes the application of FSCS-ART on `grep`, a widely used line-search program in Unix and Linux environment. Details of techniques, relevant experiments, the results and the findings are included in this

chapter. Chapter 4 presents the application of FSCS-ART on three of SIEMENS programs. SIEMENS programs are simple text processing utilities which were originally assembled by researchers at SIEMENS Corporate Research for control-flow and data-flow test adequacy criteria [29]. They can be acessed freely in SIR [30]. The experiment design, which is different to the experiment in Chapter 3, together with the experiment results and findings are discussed in this chapter. Next, Chapter 5 presents the implementation of ART by exclusion to test `grep`. This chapter discusses the new approach in detail along with the experimental results and findings. Chapter 6 discusses failure causing input patterns in programs with complex input types. Chapter 7 presents the effectiveness of using MT as the test oracle when applying ART on programs with complex input types. The investigation about how our ART techniques behave against each identified pattern is also presented in this chapter. In Chapter 8, the application of ART for complex input types on Metamorphic Testing (MT) is presented. Finally, Chapter 9 summarises all contributions and findings as the conclusions of this thesis. Suggestions for related future work are also presented in this chapter.

Chapter 2

Theoretical Framework

2.1 FSCS-ART

As previously stated, to improve the performance of random testing (RT), Chen *et al.* [17] proposed an enhanced method, known as adaptive random testing (ART). Their approach was based on the intuition that two test cases close to each other were more likely to have the same failure behavior than two test cases that were widely separated. The intuition was supported by a number of studies [31, 32] which found evidences that faults tend to cause erroneous behaviour to occur across contiguous regions of the input domain. Hence, they believed that a method that spread test cases more evenly would identify failures using fewer test cases.

As mentioned, there are many possible ways to apply the principle of ART. One that has been widely used is distance-based ART, also known as Fixed-Size Candidate Set ART (FSCS-ART) [17]. It makes use of the Euclidean distance as a distance measure to ensure test cases are spread out evenly. At first, two types of sets are defined: *candidate set* and *executed set*. The candidate set contains a pre-defined number n candidate inputs, whereas the later set stores all test cases that have been executed. Both sets are initially empty. Then n elements of candidate set are selected randomly from the existing input domain. For the first round, a random test case is selected, executed, and if the execution does not reveal any failures, it would be added to the executed set whilst the candidate set is reset as empty. For subsequent iterations, n test cases are randomly selected and placed again in the candidate set. Then, the minimum distance of each candidate test case to the executed test cases is calculated, that is the nearest executed test case. The candidate having maximum minimum distance is then selected as the next test case. This is known as the *max-min* criterion. The testing stops whenever an executed test case reveals a failure. Then, the number of test cases required to reveal the first failure is used as the metric to measure the effectiveness of the technique. This

metric is known as the *F-Measure* [28]. Currently, ART has been widely used for programs with numeric inputs. The previous experiments showed that ART could be more effective than RT with an F-Measure up to 50% smaller than RT [20, 33, 34].

2.1.1 FSCS-ART for Complex Input Types

In the beginning, ART studies examined the testing of programs with exclusively numeric inputs [24, 25, 26]. Since in the real world we deal with a lot of programs accepting complex types of inputs, for wide application of ART methods, supporting these varieties of inputs is clearly necessary. The basic issue of the extension of ART techniques is the distance measure used. The original ART techniques use the Euclidean distance as it is an effective distance measure to support smooth continuous functions in numeric input domain. Suppose that there are two points close to each other, as measured by the Euclidean distance, it is likely that the execution patterns of that software will be very similar. Thus, the failure behaviour will also be similar - either both inputs reveal failures or both will not. However, the Euclidean distance measure cannot be directly applied to programs with complex input types. We need to find a distance measure showing the similarity of two inputs' execution patterns as reflected by the Euclidean distance.

Recently, several studies have been conducted to extend ART to wider input types. Kuo [18] and Merkel [27] studied the application of FSCS-ART on non-numeric input types. They introduced a category-partition based distance measure to replace the Euclidean distance used by the original ART in measuring the distance between two numeric inputs.

Ciupa *et al.* [35] examined programs with non-numeric inputs which particularly focussed on object-oriented applications. They developed an FSCS-ART-based tool, called ARTOO to test object-oriented programs. Their method is also based on the intuition to select test cases spread out evenly across input domain. Instead of the Euclidean distance, they introduced new distance measures to calculate the difference between two objects. They classified the distance measures into two major groups: *elementary distance* and *composite object distance*. As explained by its name, elementary distance dealt with elementary values appearing in fields of objects such as integer and boolean, whilst composite object distance defined distance between two composite objects. For the composite objects, there were three properties taken into account, which were *type distance*, *field distance*, and *recursive distance*. The type distance measured the difference between object types, the field distance measured the difference between matching fields, whereas the recursive distance measured the difference between unequal references of corresponding objects. They used the maximum of sum of the differences instead of max-min criterion as

the selection criterion.

Jaygarl and Chang [36] developed an enhancement of ARTOO developed by Ciupa *et al.* [35]. They argued that ARTOO can improve RT by requiring less number of test cases in revealing the first failure; however ARTOO needs 160% longer time than RT for revealing the first failure. They attempted to enhance ARTOO by introducing four extended techniques: Distance-based input selection, Type-based input selection, Open-access selection and Array Generation. From the experiments, they found that their technique can improve ARTOO in term of the time requirement and the test coverage. They also tested the technique on larger scale software such as: Java collections, Apache Ant, and ASM.

Lin *et al.* [37] proposed a new approach in generating the test pool containing divergent objects as well as boundary values of the input space that would be used by the ART methods to test Java programs. The test pool generation approach is called divergence-oriented approach as it tries to generate a test pool of divergent test cases where each test case has a significant difference from the others. Once the test pool is generated, the ART technique - adopted from ARTOO [35] - is used to test the Java programs. They developed a tool called ARTGen to support this testing approach. They conducted some experiments to apply ART on some Java programs using three different test pools. Two test pools were generated using two random test case generators and the other one was using ARTGen. They compared the performance of ART by using the three different test pools. They found that ARTGen-test pool generated higher quality test cases. The test cases had significant differences. Thus, the F-Measures of ART using this test pool were significantly smaller than using the other test pools.

This thesis contributes to the extension of FSCS-ART which was commenced by Kuo [18] and Merkel [27]. As mentioned, instead of using the Euclidean distance, they presented an alternative difference measure to ensure that the test case selection was performed evenly spread across input domain. The difference measure used the concepts of category and choices from the category-choices method [38]. The method is a specification-based method that defines a number of categories which are major aspects of input parameters or operational environment. All possible values of a category are divided into some disjoint groups named choices. Choices are assigned to form a basis to define the similarity or difference of non-numeric input values. Given two program inputs, we need to determine categories and choices of those two inputs. Then we use this information to calculate the distance between them determined by the total number of different choices. In other words, given two program inputs x and y, our distance measure is a count of the number of categories in which x and y have different choices. A greater distance represents the situation where the two inputs are more dissimilar or more evenly spread in the numeric input

context.

As mentioned in Section 1.3.2, the original FSCS-ART has usually been applied using the *max-min* selection criterion to ensure that executed test cases are evenly spread, which is a fundamental concept of ART. The max-min criterion specifies that for each candidate, the smallest distance to a previously executed test case is calculated. The candidate with the largest such distance is chosen as the next test case. However, for the category-choices based distance measure, the max-min criterion cannot guarantee to yield evenly spread test cases. As most programs with complex input types will not have a very large number of categories, the possible values of distances are only integers ranging from 0 to the number of categories that have been defined. If more test cases are executed, the range of values for the minimum distance of the following candidates will become smaller and smaller, and many candidates eventually have the same minimum distance. It means that the the distance measure will diminish its discriminatory power along with the testing process. Hence, the method will degrade to be a computationally expensive variation of RT.

Due to this issue, Kuo [18] made some modifications of the max-min selection criterion. Firstly, in addition to the minimum distance, she also used the sum of distance - called the *max-sum* criterion. Instead of calculating the smallest distance to a previously executed test case, the distance to all previous test case are accumulated for each candidate. Then the test case having maximum total distances will be selected. The sum of distances obviously results a larger range of values than max-min criterion. Secondly, she adopted an orthogonal approach which considered only the n most recently executed test cases. This approach was firstly introduced in [39] and known as the *aging* criterion. This approach restricts the number of distance comparisons so that the candidate test cases can still be discriminated. Then all existing approaches were combined together. In other words, Kuo [18] presented four selection criteria for applying FSCS-ART on complex input types: *max-min without aging* (that is original FSCS- ART), *max-min with aging*, *max-sum without aging*, and *max-sum with aging*.

2.2 Restricted Random Testing (RRT)

Another variant of ART was devised by Chan *et al.* [19], known as restricted random testing (RRT). This technique selects test cases located outside exclusion regions. Before a new test is generated, an exclusion region of a specified radius is placed around each previously executed test case. A candidate input is accepted as the next test case if the candidate lies outside the exclusion regions. Otherwise, the generation of candidates continues until the candidate satisfying the criterion is

found.

Firstly, we set the target exclusion ratio, R, which is a ratio of the total area of the exclusion regions to the area of the input domain. For each execution test case, each test case will have its exclusion region. If N test cases have been executed, there will be N exclusion regions in equal size where the total area of the regions is $R * the\ area$ of the input domain, say A. For the larger N, the exclusion region will be smaller to keep the total exclusion ratio constant to R. However, the exclusion regions can be overlapped. Hence, the actual total area of the exclusion regions will be almost always less than A.

2.2.1 RRT for Complex Input Types

For applying R-ART to programs with complex input types, we also adopt the category-choices based distance measure as used in FSCS-ART for complex input types. However, the selection criteria will be based on the intuition used in the original R-ART. Details of this method would be discussed in Chapter 5.

2.3 Metamorphic Testing (MT)

A test oracle is a mechanism that can be used by testers to verify the correctness of computed outputs of a program [8]. As mentioned, we encounter the test oracle problem when (i) there is no such oracle or (ii) the application of such oracle becomes too expensive. To alleviate this problem, Chen *et al.* [9] have developed the metamorphic testing (MT) approach which has been successfully applied in various application domains ([40], [41], [42], [43], [44], [45]).

The key idea of MT is to use a pair of relationships: the *test case relation* and the *test result relation*. These relationships are identified from the properties of the program under test. Thus, MT is known as a property-based testing method. The test case relation is used to generate the *follow-up test cases* from the original test cases, referred to as the *source test cases*. The test result relation is then used to be complied with the relationship between the outputs of the source and the follow-up test cases. If the relation is false, then this is indicative of an error in the program under test.

Let us use the *sine* function to illustrate the idea of MT. Suppose P is a program that computes the *sine* function. We assume that we do not have an oracle for this problem – that is, we do not know exactly what is the value of the *sine* of an arbitrary input. Let 0.49 (radians) be a source test case of P. After executing P with 0.49, the corresponding output is P(0.49). Due to the lack of an oracle, P(0.49) may be correct but we have no way to verify it. One property of the *sine* function is

that $sin(0.49) = sin(2\pi + 0.49)$. The test case relation of this property is selecting $2\pi + 0.49$ as the follow-up test case. The test result relation is the *sine* function ought to yield the same value for $2\pi + 0.49$ as for 0.49. After executing P with $2\pi + 0.49$, we can then check whether the following equalities hold: P(0.49) = P(2π + 0.49). If the equality relation does not hold, we know that P contains an error.

The success of MT relies on the existence of a metamorphic relation (MR) which comprises of the two interrelated relations: the test case relation and the test result relation. Once an MR is defined, the generation of the follow-up test cases from the source test case and the verification of the test result relationship can be automated.

The following are formal definitions of MR and the procedure of MT [46]:

Definition of MR Suppose a function f has inputs, $\mathrm{I}_1 = \{x_1, x_2, ..., x_i\}$ where $i \geq 1$ and let $\mathrm{O}_1 = \{f(x_1), f(x_2), ..., f(x_i)\}$ be the corresponding outputs. Let S= $\{f(x_{s_1}), f(x_{s_2}), ..., f(x_{s_k})\}$ denote a subset of O_1 where S may be empty. Let $I_2 = \{x_{i+1}, x_{i+2}, ..., x_j\}$ be other inputs to f where $j \geq i+1$ and $O_2 = \{f(x_{i+1}), f(x_{i+2}), ..., f(x_j)\}$ be the corresponding outputs. Suppose there exists a relation R_1 among I_1, S and I_2, and another relation R_2 among I_1, I_2, O_1 and O_2 such that R_2 must be satisfied whenever R_1 is satisfied. Then, a metamorphic relation MR can be defined as:

MR =$\{ (x_1, x_2, ..., x_j, f(x_1), f(x_2), ..., f(x_j))| \mathrm{R}_1(x_1, x_2, ..., x_i, f(x_{s_1}), f(x_{s_2}), ..., f(x_{s_k}), x_{i+1}, x_{i+2}, ..., x_j) \rightarrow \mathrm{R}_2(x_1, x_2, ..., x_j, f(x_1), f(x_2), ..., f(x_j))\}$

Elements of I_1 and I_2 are referred to as source test cases and follow-up test cases, respectively. Relations R_1 and R_2 are referred to as the test case relation and the test result relation, respectively.

Procedure MT Suppose the function f is implemented by a program P. The procedure of MT using the MR described in the above definition consists of the following steps:

1. Run P using a series of test cases I_1 as source test cases and get the corresponding outputs O_1.
2. Use R_1, I_1, and O_1 to generate follow-up test cases I_2.
3. Run P using I_2 as inputs to get the corresponding outputs O_2.
4. Check the relation R_2: if R_2 does not hold then a failure is revealed.

As program failures may be sensitive to different MRs, it is recommended to identify more than one MR when applying MT. For our *sine* function example, other possible MR is as follows. For any inputs x_1 and x_2 where $\pi/2 < x_1 < x_2 < 3\pi/2$, $\sin(x_1)$ must be greater than $\sin(x_2)$. Formally speaking, $\mathrm{MR}_{\sin_2}$: $\pi/2 < x_1 < x_2 < 3\pi/2 \rightarrow \sin(x_1) > \sin(x_2)$

Chapter 3

Applying FSCS-ART on `grep`

In this chapter, we apply FSCS-ART on programs with complex input types. As mentioned, the initial studies of ART merely used programs with numeric input type as the testing objects [24, 25, 26]. Here, we extend the application of the technique on programs with complex input types. Among many ART techniques, we choose FSCS-ART as it has been most extensively studied in numerical programs, and has repeatedly shown itself to be an effective method. We expand on the initial work of Merkel [27] and Kuo [18]. This chapter develops enhancements of the ideas and applies them to a real life program which has not been attempted in those previous studies. We apply FSCS-ART on a real program with complex input types, called `grep`. `grep` is a standard string-searching utility distributed with most unix-like operating systems [47]. Detailed experiments and results are presented in this chapter.

3.1 Requirements for Applying FSCS-ART to Programs with Complex Input Types

As mentioned, the basic concept of ART is to select test cases evenly spread across the program's input domain. Specific to FSCS-ART, the technique makes use of the idea of the *candidate set* and the *executed set* in selecting the next test cases as described in details in Section 2.1.1. In summary, there are two key points of applying the FSCS-ART:

- the *distance measure*. This measure is used to determine the distance between every candidate in the candidate set and all elements in the executed set. For the original FSCS-ART, we use the Euclidean distance.
- the *selection criterion*. Once we get the distance between pair of candidate and executed test cases, we need to select the best candidate test case for the

next execution based on a certain selection criterion. The selection criterion attempts to addres the "widely spread" goal of ART. For the original FSCS-ART, we use the *max-min* criterion. This criterion selects a candidate having the maximum smallest distance to executed test cases.

Due to different characteristics of complex input types, the first key point above cannot be used using the same approach as for the numeric input types. Most programs with numeric input types consists of smooth continuous functions. Hence, the execution patterns of two numeric inputs close to each other – measured by the Euclidean distance – are likely to be very similar. Thus, the failure behaviour will also be similar - either both inputs reveal failures or both will not. Therefore, in complex input types, we need to find a distance measure that is able to show the similarity of two inputs' execution patterns as reflected by the Euclidean distance.

Accordingly, Merkel [27] and Kuo [18] have studied preliminary ideas to adapt FSCS-ART on such programs. Instead of using the Euclidean distance, they presented an alternative difference using the concept of category and choices from the category-choices method [38]. The method is a specification-based method that defines a number of categories which are major aspects of input parameters or operational environment. All possible values of a category are divided into some disjoint groups named choices. Choices are assigned to form a basis to define the similarity or difference of non-numeric input values. Suppose, we are given two program inputs x and y. First, we need to determine categories and choices of these two inputs. Then, we count the number of categories in which x and y have different choices. A greater distance represents the situation where the two inputs are more dissimilar or more evenly spread in the numeric input context. The maximum difference or distance is the total number of categories defined for the program under test.

For the second key point of FSCS-ART above, we may not be able to simply use the max-min criterion as the selection criterion. As we apply the category-choices based distance measure, this criterion cannot guarantee to select test cases evenly spread. Most of programs with complex input types will not have a very large number of categories and the possible values of distances are only integers ranging from 0 to the number of categories the has been defined. If more test cases are executed, the range of values for the minimum distance of the following candidates will become smaller, and many candidates eventually have the same minimum distance. In this case, the the distance measure will diminish its discriminatory power along the testing process. Hence, the method will degrade to a very expensive version of RT. Due to this issue, some modifications of the max-min selection criterion are introduced with detailed explanation in Section 2.1.1. First is the *max-sum* criterion. The distance selects a candidate having maximum sum of distances to all executed test cases. The second one is an orthogonal approach called the *aging* criterion. This ap-

proach restricts the number of distance comparisons by considering only the n most recently executed test cases. Then all existing approaches are combined together. In other words, we have four selection criteria for apllying FSCS-ART for programs with complex input types: *max-min without aging* (that is original FSCS- ART), *max-min with aging*, *max-sum without aging*, and *max-sum with aging*. Hereafter, the criteria are respectively known as *NoAgeMaxMin*, *AgeMaxMin*, *NoAgeMaxSum*, and *AgeMaxSum*.

3.2 Experimental Work

3.2.1 Testing Object: `grep`

We use the Unix command-line utility program `grep` distributed by the GNU Project[47] as the testing object. `grep`'s manual page describes it as follows:

> The `grep` command searches one or more input files for lines containing a match to a specified pattern. By default, `grep` prints the matching lines.

These are several reasons why we chose `grep` as the testing object of the FSCS-ART application:

- we can freely access released versions, current and historical of `grep` including source codes and some of its change history.
- `grep` has manageable size and complexity
- `grep` has non-trivial input formats which are complex enough to be interesting but are still feasible for the automated input generation purpose.

`grep`'s inputs can be categorized into three groups of parameters: *options*, which consists of a list of commands to modify the searching process, *pattern* which is the regular expression to be searched for, and *files* refers to the input files to be searched. However, as testing all of `grep`'s functionalities would have been impractical, in this thesis we restrict our experimentation to testing the regular expression analyser of `grep` which is associated to the second parameter.

For the experiment purpose, we require a test pool of `grep`'s inputs and its faulty versions. The UNL Software-artifact Infrastructure Repository (SIR) [30] provides a test pool and faulty versions that can be freely used for the research purpose. However, the bugs in the SIR are related to other aspects of `grep`, so we could not use `grep`'s artifacts provided in SIR.

For this experiment, we generate our test pool designed specifically to our study objective and scope. Details can be found in Section 3.2.3. Related to the faulty versions, we are able to find one reported real bug which is suitable to our experimental platform. The real bug is documented in the change log file for `grep` version 2.4. It indicates that certain regular expression ranges were compared using raw byte codes to range boundaries, rather than using the relevant collation sequence determined by the particular Unix locale [48]. Obviously, this bug is exposed only when the locale setting is other than the default, so we used the locale "en_US.UTF-8" while performing testing to detect this bug.

However, we could not find more versions of `grep` containing "real bugs" relevant to our experiments. The examination of the change log showed that the vast majority of the bugs found and fixed during `grep` development were either platform-specific, or manifest themselves. Hence, they would not be detected quickly enough to make experimental comparisons practical. Obviously one real bug is not enough for conducting a comprehensive study. Therefore, we also use program mutation to generate more "faulty" versions of `grep` for our experiments. Program mutation is a process to intentionally seed faults into software [49, 50]. It has been widely used for several different applications within software testing and reliability, including measuring the adequacy of test suites. Details of our mutation process is described in Section 3.2.5.

3.2.2 Category-choices Scheme for `grep`

As mentioned, for the application of FSCS-ART on programs with complex input types, instead of the Euclidean distance, the category-choice based distance measure is used. Accordingly, in the initial stage, we need to define a category-choices scheme for `grep`– object testing in this chaper. As mentioned, we restrict our experimentation to testing the regular expression analyser of `grep`. As a consequence, our categories-choices scheme considers only on this portion of `grep`. To define the scheme, the user documentation was first examined, however it did not provide significant details to construct appropriate categories and classes. Therefore, we also made use the existing testing information about `grep` recorded in the SIR. The exact details of categories and choices for our experiment are presented as following:

1. Category NormalChar, classes: NormalAlNum and NormalPunct
   This category represents any literal characters. NormalAlnum is a class particularly for alphabetic or numerical literal whereas NormalPunct is for punctuation character.
   For example:

- Linux $ `grep` “in”<File>.
  It will match any lines in <File> containing “in ”.
- Linux $ `grep` “\? ”<File>.
  It will match any lines in <File> containing “? ”.

2. Category WordSymbol, classes: YesWord and NoWord
   This category represents “word” or “non-word” *metacharacters*. YesWord is a class for the presence of the word metacharacter(\w) whereas NoWord is for non-words (\W). Here, a word refers to a group of characters containing alphabetic or numeric or both whilst a non-word refers to a group of other characters.
   For example:

   - Linux $ `grep` “\w ”<File>.
     It will match any lines in <File> containing “abc123 ”.
   - Linux $ `grep` “\W ”<File>.
     It will match any lines in <File> containing “@*$ ”.

3. Category DigitSymbol, classes: YesDigit and NoDigit
   This category represents digit or non-digit metacharacters. YesDigit is a class for the presence of the digit metacharacter (\d) whereas NoDigit for the non-digit (\D)
   For example:

   - Linux $ `grep` “\d ”<File>.
     It will match any lines in <File> containing “1234 ”.
   - Linux $ `grep` “\D ”<File>.
     It will match any lines in <File> containing “abc?!@ ”.

4. Category SpaceSymbol, classes YesSpace and NoSpace
   This category represents space or non-space metacharacters. YesSpace is a class for the presence of the space metacharacter(\s) whereas NoSpace for non-space (\S).
   For example:

   - Linux $ `grep` “\s ”<File>.
     It will match any lines in <File> containing “”.
   - Linux $ `grep` “\S ”<File>.
     It will match any lines in <File> with NO space character.

5. Category NamedSymbol, classes: ALPHA, UPPER, LOWER, DIGIT, XDIGIT, SPACE, PUNCT, ALNUM, PRINT, GRAPH, CNTRL, and BLANK

This category represents any of these self-explanatory metacharacters: [:ALPHA:], [:UPPER:], [:LOWER:], [:DIGIT:], [:XDIGIT:], [:SPACE:], [:PUNCT:], [:ALNUM:], [:PRINT:], [:GRAPH:], [:CNTRL:], and [:BLANK:]. Their names starts with [[: and end with :]].
For example:

- Linux $ `grep` "[[:UPPER:]] "<File>.
  It will match any lines in <File> containing any upper case character such as "ABC ".

6. Category AnyChar, class: Dot
   This category only consists of one class, Dot to depict the presence of ".", a metacharacter representing any single character.
   For example:

   - Linux $ `grep` ". "<File>.
     It will match any lines in <File> containing any character

7. Category Range, classes: NumRange, UpcaseRange, and LowcaseRange
   This category represents any of "range" metacharacters. In details, NumRange is a class for any metacharacters representing the range of numerics whereas UpcaseRange for upper cases and Lowcase for lower cases.
   For example:

   - Linux $ `grep` "[1-7] "<File>.
     It will match any lines in <File> containing "3 ".
   - Linux $ `grep` "[M-Z] "<File>.
     It will match any lines in <File> containing "Q ".
   - Linux $ `grep` "[a-d] "<File>.
     It will match any lines in <File> containing "c ".

8. Category Bracket, classes: NormalBracket and CaretBracket
   This category represents any patterns encompassed by [] or [^], except the "range" metacharacters where each is represented by class NormalBarcket and CaretBracket respectively. [] represents the existence of any single token within the bracket whereas [^anychar] presents the any single characters instead of characters within the bracket but the absence of the first character.
   For example:

   - Linux $ `grep` "[abc] "<File>.
     It will match any lines in <File> containing "a "or "b "or "c "

- Linux $ `grep` “[˜abc] ”<File>.
  It will match any lines in <File> containing “input ”which has “b ”or “c ”but not “a ”

9. Category Iteration, classes: Qmark, Star, Plus, and Repmin
   This category represents any iteration metacharacters. Qmark is a class for the presence of the metacharacter “?” which shows the presence of the preceding character, precisely zero or one times. Star is a class for representing “*” that shows the presence of the preceding character with 0 or more occurences, where Plus is for the metacharacter “+” representing the presence of the preceding character one or more times. Repmin is a class for $\{min,max\}$ representing the presence of the preceding character with at least *min* and at most *max* occurences.
   For example:

   - Linux $ `grep` “a* ”<File>.
     It will match any lines in [File] containing no “a ”or any numbers of “a ”
   - Linux $ `grep` “a+ ”[File].
     It will match any lines in [File] containing one “a ”or >1 numbers of “a ”
   - Linux $ `grep` “a? ”[File].
     It will match any lines in [File] containing no “a ”or one “a ”
   - Linux $ `grep` “a{2,3} ”[File].
     It will match any lines in [File] containing “aa ”or “aaa ”

10. Category Parentheses, classes: NormalParentheses, Backref
    This category represents patterns encompassed by parentheses “()”. NormalParentheses is a class for ordinary “()” which shows that the whole characters inside “()” should be treated equivalently by any metacharacters following the bracket. Backref is a class for special “()” which is must be accompanied by the presence of \‘[n] ’. This metacharacter represents characters inside the n^{th} “()” in the pattern with the assumption that we have more than one “()” in the pattern.
    For example:

    - Linux $ `grep` “(abc)+ ”[File].
      It will match any lines in [File] containing at least an occurence of “abc ”such as “abc ”or “abcabcabc ”
    - Linux $ `grep` “(the) input (file) of \1 grep command ”[File].
      It will match any lines in [File] containing “the input file of the grep

command ".

11. Category Line, classes: BegLine, EndLine, and BegEndLine
    This category represents metacharacters related with the line boundary. BegLine is a class for a metacharacter "^". This metacharacter represents the pattern after ^which must be located in the beginning of the line. EndLine is a class for "$" is for the presence of the pattern before $ which must be located at the end of the line where BegEndLine is a class for the presence of two other class at the same time.
    For example:

    - Linux $ `grep` "^abc "[File].
      It will match any lines in [File] containing "abc "in the beginning of the line
    - Linux $ `grep` "abc$ "[File].
      It will match any lines in [File] containing "abc "at the end of the line
    - Linux $ `grep` "^abc$ "[File].
      It will match any lines in [File] containing "abc "that exists at the begin and also the end of line (it means that the line only contains "abc ")

12. Category Word, classes: BegWord, EndWord and BegEndWord
    This category represents metacharacters related with the word boundary. BegWord is class for a metacharacter "\<". This metacharacter indicates that for a valid match, the pattern after \< which must be located in the beginning of the word (or in other words, it must be preceded by a space. EndWord is a class for "\>" representing the presence of the pattern before \> which must be located at the end of the word where BegEndWord is a class for the presence of two other class at the same time or in other words it is for the presence of exactly the same word as in the pattern.
    For example:

    - Linux $ `grep` "\<abc "[File].
      It will match any lines in [File] containing "abc "that exists in the beginning of word. So it will match line containing "abcde "but not "inabc ".
    - Linux $ `grep` "abc>\"[File].
      It will match any lines in [File] containing "abc "that exists in the end of word. So it will match line containing "inabc "but not "abcde ".
    - Linux $ `grep` "\<abc>\"[File].
      It will match any lines in [File] containing the word "abc ".

13. Category Edge, classes: YesEdgeBeg, YesEdgeEnd, YesEdgeBegEnd, NoEdgeBeg, NoEdgeEnd and NoEdgeBegEnd
    This category represents metacharacters related with edge boundaries. YesEdgeBeg is a class for the presence of the metacharacter "\b" which has similar description with the class BegWord of the category Edge whereas YesEdgeEnd is for the similar metacharacter, however its description is equivalent to EndWord. YesEdgeBegEnd is a class for the presence of the metacharacter at the start and also at the end of the pattern in the same time which is equivalent to the description of class BegEndWord. NoEdgeBeg, NoEdgeEnd and NoEdgeBegEnd have similar descriptions with YesEdgeBeg, YesEdgeEnd, and YesEdgeBegEnd respectively but they represent the metacharacter "\B" instead of "\b". "\B" represents the occurence of the pattern after or before \B which is located not in the edge of a word.
    For example:

    - Linux $ `grep` "\babc "[File].
      It will match any lines in [File] containing "abc "that exists in the beginning of word, similar with "\< "but can be put before or after the string such as grep "abc\b "[File]
    - Linux $ `grep` "\B abc "[File].
      It will match any lines in [File] containing "abc "that exists NOT in the beginning of word so it will match "inabc "but not "abcde "and can be put after the word as well such as grep "abc \B "[File]

14. Category Combine, classes: Concatenation and Alternative
    This category represents combination of classes from other categories. Concatenation is class for any combination of other classes whereas alternative is a class for the combination using the metacharacter "|". Whilst concatenation represent the presence of all combined classes respecting their orders, alternative just represents any single class separated by |.
    For example:

    - Linux $ `grep` "[a-h][1-7](Input)* "[File].
      It will match any lines in [File] containing "a6In putInputInput "
    - Linux $ `grep` "[a-h][1-7]|(Input)* "[File].
      It will match any lines in [File] containing "a6 "or containing "InputInputInput "

Independent Category	Dependent Category
NormalChar	Bracket
WordSymbol	Iteration
DigitSymbol	Parentheses
SpaceSymbol	Line
NamedSymbol	Word
AnyChar	Edge
Range	Combine

Table 3.1: Table of Independent and Dependent Categories for `grep`

3.2.3 Generation of the Test Pool

As mentioned, SIR [30] also provides a test pool of `grep`. However, as previously discussed, we devise a method to generate a more suitable test pool. The method must construct a large number of inputs to more closely represent incremental random sampling from the entire program domain. The inputs are then placed as test case candidates in the selection process.

To perform the generation, we first divided the categories into two groups – the independent and the dependent categories. The independent group includes all categories that contain elements which can form a valid test case on their own. Dependent categories need the presence of elements of other categories to form valid test cases. Categories 1 through 7 described in Section 3.2.2 are classified as independent categories whilst the rest are classified as dependent. The dependent and independent categories are listed in Table 3.1.

Then we systematically generate candidates based on the uniform distribution where each category contributes equally. For each round of the generation, a class of each category and choice is generated in turn. For independent categories, this is straightforward. For example, the "NormalChar" category has a choice "NormalAlNum". To generate a test case which has this choice, a single character from the set containing all letters and digits is generated randomly. For dependent categories, the elements must be combined with an element from an independent category (based on the constraints written in the specification of `grep`) to make a valid test case. For example, the dependent category "Iteration" could be combined with a "NormalAlNum" character to form a regular expression. The algorithm to generate the test pool for `grep` is detailed in Figure 3.1.

The total rounds (n) that we run for this experimental work is 50000. Examples for each independent category are shown in Table 3.2, and example combinations of categories including dependent categories in Table 3.3.

Category 14, "Combine", is a special case: it involves either concatenation or selection between alternatives. When this category is selected, a choice (concatenation or selection) is then determined. The procedure described in the paragraphs above

Category	Possible Choice	Test Case
NormalChar	NormalAlNum	A
WordSymbol	NoWord	\W
DigitSymbol	YesDigit	\d
SpaceSymbol	NoSpace	\S
NamedSymbol	ALPHA	[:ALPHA:]
AnyChar	Dot	.
Range	NumRange	[1-9]

Table 3.2: Examples of test cases involving only independent categories for `grep`

Combination of Categories	Possible combination of choices	Example of Test Cases
Bracket;NormalChar	*NormalBracket*;NormalAlNum	[B]
Iteration;Range	*Star*;UpcaseRange	[A-Z]*
Parentheses; NormalChar;DigitSymbol	*NormParen*; NormalAlNum;YesDigit	(A\d)
Line;WordSymbol	*BegLine*;YesWord	^\w
Word;DigitSymbol	*EndWord*;NoDigit	\D\>
Edge;Range	*YesEdgeBegEnd*;NumRange	\b[1-9]\b
Combine;*Iteration*; *Parentheses*; NormalChar; Range	*Concatenation*;*Plus*; *NormParen*;NormalAlNum; LowcaseRange	(a[0-9])+

Table 3.3: Examples of test cases involving dependent categories for `grep`. *Note: the dependent categories and their associated choices are italicized*

is then used to generate the two subsidiary elements which are finally combined in the test case.

Obviously, the test pool generated according to the method above is not a random sampling of the entire input domain of `grep`. Instead, it is only a small subset of the input space associated with the regular expression analyzer as defined as our experimental scope. At the end of the generation process, the test pool are filtered to remove duplicate test cases. Finally, the test pool contains 171,634 inputs.

3.2.4 Distance Measurement and Test Case Selection Example

Following is an example of how FSCS-ART is applied on selecting test cases in our test pool to test `grep`. To simplify the example, we assume that two test cases, t_1 and t_2 have been selected and executed where t_1 is "([^\d])^\S$\<.\>\B\w|\({2,} [b-i][[:lower:]]" and t_2 is "[x-y]+". For this example, we set the size of the candidate set to two. To select the next test case for execution, t_3, we select randomly two test cases from the test pool. Suppose we have c_1 "(^[2-6]$)[[:blank:]]+ (\w)\D\>[^\s] (\b. \b)\1" and c_2 "[1-6]*". Based on our category-choices scheme, t_1, t_2, c_1, and c_2 belongs to different test frames as described in Table 3.4.

Based on our category-choices based distance measure, at first we calculate the distance between each candidate to the executed test cases. As mentioned in Section 2.1.1, the distance between two test cases is the number of different choices they have. Let us denote the distance between test case tc_1 and test case tc_2 by $D(tc_1, tc_2)$. From Table 3.4, we identify that the number of different choices between c_1 and t_1 is 9 while the number of different choices between c_1 and t_2 is 13. Hence, $D(c_1, t_1) = 9$ and $D(c_1, t_2) = 13$. Following the same approach, $D(c_2, t_1) = 14$ and $D(c_2, t_2) = 2$.

After calculating the distance between each candidate to the executed test cases, we apply the selection criteria as follows:

- *NoAgeMaxMin.* Based on this criterion, first we find the smallest or minimum distance ($minD$) of c_1 and c_2 to t_1 and t_2. $minD(c_1) = 9$ which is its distance to t_1 and $minD(c_2) = 2$ which is its distance to t_2. The next test case is the candidate having the maximum $minD$, which in this case is c_1.

- *AgeMaxMin.* This method is similar to the previous one but only considers the n most recently executed test cases. For this example, we choose $n = 1$ as the example only has two executed test cases (note: in the real experiments, of course $n = 1$ cannot reflect the even spreadness idea of this criterion. In our experiments, for example, we set $n = 10$). For $n = 1$, we only consider one

Category	Choices of t_1	Choices of t_2	Choices of c_1	Choices of c_2
NormalChar	NormalPunct	-	NormalPunct	-
WordSymbol	YesWord	-	YesWord	-
DigitSymbol	YesDigit	-	NoDigit	-
SpaceSymbol	NoSpace	-	YesSpace	-
NamedSymbol	LOWER	-	CNTRL	-
AnyChar	DOT	-	DOT	-
Range	LowcaseRange	LowcaseRange	NumRange	NumRange
Bracket	CaretBracket	-	CaretBracket	-
Iteration	Repminmax	Plus	Plus	Star
Parentheses	NormParen	-	BackRef	-
Line	BegEndLine	-	BegEndLine	-
Word	BegEndWord	-	EndWord	-
Edge	NoEdgeBeg	-	YesEdgeBegEnd	-
Combine	Alternative	-	Concatenation	-

Table 3.4: Test frames of t_1, t_2, c_1, and, c_2

most recently executed test case or in this case, t_2. Therefore we only consider $D(c_1, t_2)$ and $D(c_2, t_2)$. Then, we choose c_1 as it has the maximum distance.

- *NoAgeMaxSum*. For this method, first we find the sum (total) distance ($sumD$) of c_1 and c_2 to t_1 and t_2. $sumD(c_1) = 22$ and $sumD(c_2) = 16$. The next test case is the candidate having the maximum $sumD$, which in this case is c_1.

- *AgeMaxSum*. Similarly to *AgeMaxMin*, this method is like *sum without aging* but only considers the n most recently executed test cases. For this example, similarly, we choose $n = 1$. Hence, we consider only the last executed test case, which is t_2. Then, we only consider $D(c_1, t_2)$ and $D(c_2, t_2)$. As the next test case, we choose c_1 as it has the maximum distance.

3.2.5 Mutation Process

As mentioned, the faults in SIR are not related to the regular expression analyzer. Therefore, they cannot be used in this experimental work. From our examination of the `grep` change log, we were able to find one reported `grep` bug which is suitable for our experiment. Details of the real bug are described in Section 3.2.1.

However, one real bug is not enough for conducting a comprehensive study. Therefore, we also used program mutation to generate more faulty versions of `grep` for our experiments. As mentioned, program mutation is a process to insert faults deliberately into software [49, 50]. In specific, we adapt the related work of Agrawal *et al.* [51]. They described a number of *mutant operators* for the C programming language. Each mutant operator attempts to modify a class of program statements

in a systematic way. However, it must keep the modified program compilable with the expectation of different output behaviours.

In their study, Agrawal *et al.* [51] classified four group of mutations: *statement mutations*, *operator mutations*, *variable mutations*, and *constant mutations*. Statement mutations involve the changing of entire statements whereas operator mutations are simply the one-to-one replacement of arithmetic or logical binary or unary operators; such as replacing the code fragment `a+b` with `a-b`. Variable mutations involve changing which variables are used in a particular statement; constant replacement involves replacing constants with another value. In this study, we generate mutated versions for our experiments based on the work of Agrawal *et al.* [51].

Manual selection and insertion of the mutant operators to the program might choose mutated versions which are biased to ease the failure detection of the testing method applied. Therefore, we develop an automated tool to generate mutated versions of `grep` for experimentation. In developing the automated tool, it is impractical to apply all mutation operators in [51]. Therefore, we only use a subset of those mutation operators in our mutation tool. In specific, we make use of some of the *statement mutations* and *operator mutations*. We implemented two forms of statement mutation. The first form involves replacing a `continue` statement with a `break` and *vice versa*. In the second, the label in a `goto` statement is replaced with another valid label. Our mutation operators involved substituting arithmetic and logical operators. To generate a mutated version, the tool randomly chooses a line in the source code files of interest, and searches systematically for an opportunity to apply a mutation operator. Once the suitable line is found, the particular form that the mutant will take is then randomly chosen (for instance, in operator replacement, a possible replacement operator is chosen). Each mutated program only has a single mutation operation applied to it.

As our test case generation was aimed at generating a variety of regular expression patterns, the mutation process only considers the file `dfa.c` to be transformed. The file is the part of the `grep` source code relating to this functionality.

At first, we generated 164 mutated versions, or hereafter are also referred to as *mutants*. However, some mutants have very high failure rate and some have very small not. Therefore, not every mutant generated is being used for the further experiments. If the failure rate is too high, any reasonable test strategy will detect such faults easily. Ones with very low failure rates make experimentation infeasible with the resources available to us. Therefore, only nineteen versions have been suitable to be used in our study. These mutants have failure rate between 0.001 to 0.1, corresponding to F-measures of between 10 and 1000 for RT. These failure rates are relatively high. However, our previous studies indicated that ART outperforms RT

more when failure rates are low. Thus, our choice of mutated versions, if anything, were unfavourable to ART.

3.2.6 Experimental Design and Metric Used

For this experiment, we use the original version of `grep` that is used as the base for mutation process to be the oracle testing. We set the size of candidate set to be 10. Previous work [17] has shown that the failure-detection effectiveness improves as the size is increased up to about 10, and then does not improve much further. Regarding the aging approach, we consider the 10 most recently executed test cases. This number is chosen as it has been used and proved to be sufficiently effective in the previous study of ART [18].

For each run, an input is selected from the pool (using RT or ART techniques) and executed on both the oracle and the mutant under testing. Then both outputs are compared. The different outputs indicate the detection of a failure. The number of test cases needed to detect the failure (known as the *F-count*) is then recorded. As studied by Chen *et al.* [52], the population distribution of the F-measure is geometric for RT and near-geometric for most ART variants. This makes the standard deviation very high and getting acceptably narrow confidence intervals requires large samples. Therefore, to satisfy the 95% confidence interval, we run the experiements 1000 times for RT, and for each of the four ART selection criteria towards each mutant. For each experimental condition, the F-measure was calculated by averaging F-counts across all the appropriate experimental runs.

3.2.7 Data and Analysis

The results of our experiment are presented in Table 3.5. This table reports the F-measure for RT, and the ratio of the F-measure of ART compared to RT, denoted as the *F-ratio*, for four ART strategies and all mutants of `grep`. Full results, including 95% confidence intervals, are presented in Appendix B. In summary, the 95% confidence intervals are all within ± 5%-7% of their respective F-measures – for instance, for mutant-1, the F-measure for RT was 14.06, with a 95% confidence interval of (13.23, 14.89) (Table B.1).

We then perform analysis of variance (ANOVA) on data presented in Table 3.5 to test the differences between the techniques performances. In this case we choose the F-ratio, instead of F-measure, as the input data. The reason is that the ranges of F-measures of mutants are too large and not normally distributed. Therefore, we choose F-ratio instead. The F-ratio for every case of RT is set to be 1 as RT is compared to itself. Accordingly, we investigate whether different techniques have significantly different F-ratio – which in this case reflects different failure detection

Mutant	F-measure RT	F-ratio			
		Max-Min		Max-Sum	
		Aging	No Aging	Aging	No Aging
1	14.06	44.06%	67.20%	43.77%	43.84%
2	857.59	163.80%	179.16%	39.90%	27.85%
3	14.44	45.89%	68.30%	43.67%	43.73%
4	43.58	93.85%	131.95%	53.23%	58.21%
5	37.44	40.13%	56.45%	43.10%	41.28%
6	33.46	90.82%	136.28%	43.66%	46.51%
7	33.80	42.85%	65.79%	42.98%	41.68%
8	57.46	43.08%	59.36%	44.57%	42.95%
9	50.34	41.94%	61.14%	43.71%	43.36%
10	14.05	43.81%	67.12%	28.47%	43.31%
11	203.61	44.81%	64.84%	43.67%	41.74%
12	487.92	40.06%	57.72%	45.29%	41.24%
13	657.82	41.88%	56.69%	46.03%	42.70%
14	13.64	44.83%	68.18%	44.49%	44.45%
15	271.46	43.93%	57.52%	43.11%	42.22%
16	14.29	43.18%	65.84%	42.99%	42.69%
17	470.42	42.41%	55.70%	43.21%	42.60%
18	35.15	51.55%	71.35%	44.82%	43.42%
19	21.87	72.87%	104.50%	42.89%	44.02%
20*	6.00	100.62%	129.43%	98.4%	99.90%

Table 3.5: F-measures of RT and F-ratios of ART for `grep`. (Note that mutant 20, marked with "*", denotes the case of the real bug described in Section 3.2.5).

capability. The null hypothesis is no significant differences of the F-ratios of RT and the ART strategies. When the ANOVA rejects the hypothesis, or in other words it means that the techniques are significantly different, we proceed to determine which techniques contribute the most to that difference and how the technique different from each other through a Bonferroni analysis method.

Table 3.6 presents the results of the ANOVA for the F-ratio of RT and the four ART strategies. The treatments are in the first column of the table and the sum of squares, degrees of freedom, and mean of squares for each treatment are in the following columns. F defines the ratio between the treatment and the error effect (last row). The larger F shows the greater probability to reject that the techniques' F-ratio are equal. The last column contains the p-value that represents "the probability of having a value of the test statistic that is equal to or more extreme than the one we observed" [53]. We set our level of significance to be 0.05. Thus, we accept the null hypotheses if the p-value is greater than 0.05. Otherwise, we reject the hypotheses. The last column of Table 3.6 shows that the p-value is < 0.0001 which is less than the level of significance. Therefore, we reject the null hypothesis and conclude that the techniques' F-ratios are different. Details of how different

	SS	Degrees of freedom	MS	F	p
Strategies	4	4.479	1.120	21.676	< 0.0001
Error	95	4.907	0.052	—	—
Corrected Total	99	9.386	—	—	—

Table 3.6: ANOVA of different ART strategies

Contrast	Diff.	Std. Diff.	Crit. Val.	Pr>Diff	Significant
RT vs NoAgeMaxSum	0.541	7.529	2.874	< 0.0001	Yes
RT vs AgeMaxSum	0.539	7.500	2.874	< 0.0001	Yes
RT vs AgeMaxMin	0.412	5.730	2.874	< 0.0001	Yes
RT vs NoAgeMaxMin	0.188	2.612	2.874	0.010	No
NoAgeMaxMin vs NoAgeMaxSum	0.353	4.917	2.874	< 0.0001	Yes
NoAgeMaxMin vs AgeMaxSum	0.351	4.888	2.874	< 0.0001	Yes
NoAgeMaxMin vs AgeMaxMin	0.224	3.118	2.874	0.002	Yes
AgeMaxMin vs NoAgeMaxSum	0.129	1.799	2.874	0.075	No
AgeMaxMin vs AgeMaxSum	0.127	1.770	2.874	0.080	No
AgeMaxSum vs NoAgeMaxSum	0.002	0.029	2.874	0.977	No

Table 3.7: Bonferroni analysis of the differences between the ART strategies

they are and their ranks in term of their F-ratio are calculated using Bonferroni analysis. The results are presented in Table 3.7.

Table 3.7 shows that the RT has significant differences to all ART techniques. NoAgeMaxMin and NoAgeMaxMin have significant difference to other ART techniques, however there are no significant differences among AgeMaxMin, NoAgeMaxSum, and AgeMaxSum. Then, the further analysis of Bonferroni analysis ranks all the techniques, as described in Table 3.8. By default, Bonferroni analysis assumes that a larger F-ratio has a higher ranking. However, please note that for the F-ratio, a smaller value reflects a better failure detection capability of the strategy compared to RT. Therefore, to interpret the rank correctly for our purpose, we reverse the order of the ranking. Therefore, NoAgeMaxSum has the best F-ratio, followed by AgeMaxSum, AgeMaxMin, NoAgeMaxMin, and RT. However, the first three ART strategies have no significant differences, thus they are grouped in the same level (level B, whic is in this case is the level of most effective techniques). Then, NoAgeMaxMin and RT are grouped in the lower level. Here, we can conclude that all

Category	F-ratio means	Groups
RT	1	A
NoAgeMaxMin	0.812	A
AgeMaxMin	0.588	B
AgeMaxSum	0.461	B
NoAgeMaxSum	0.459	B

Table 3.8: The rank of the ART strategies using the Bonferroni analysis

ART techniques outperform RT, however NoAgeMaxMin contributes insignificant difference to RT.

3.3 Summary and Discussion

ART was proposed to enhance the fault-detection effectiveness of RT. Previous studies of ART examined only programs with numeric input types. In this chapter, we apply ART, particularly FSCS-ART on a program with complex input types. We choose `grep`, an open source line-searching program under Linux. We used a category-choice based distance measure to replace the Euclidean distance used in original FSCS-ART. In addition, we made use more selection criteria in selecting a test case for the next execution. The different selection criteria produce different ART strategies. Accordingly, in this chapter we applied four ART strategies: *NoAgeMaxMin* (that is original FSCS- ART), *AgeMaxMin*, *NoAgeMaxSum*, and *AgeMaxSum*. We used the *F-Measure* as a metric to show the failure detection effectiveness of each strategy. The F-Measure of each strategy was then compared to the F-Measure of Random Testing (RT), referred to as F-ratio. The F-ratio smaller than 1 (100%) shows that the strategy under evaluation performs better than RT in term of failure detection effectiveness. Thus, smaller F-ratio reflects better performance of the technique. Based on the experiment results, all ART strategies outperformed RT on most mutants. However, for mutant-2, mutant-4, and mutant-20, the max-min strategies were slightly worse than RT.

We also performed an analysis of variance (ANOVA) to test the differences between the ART strategies' F-ratio. The results showed that they had significant differences to RT. To know more details about the differences, we compared each strategy using the Bonferroni analysis. The results showed that NoAgeMaxSum had been the most effective strategy in detecting failure, followed by AgeMaxSum, AgeMaxMin, and RT respectively. However, there were no significant differences among the best three ART techniques thus they were grouped in a same level of performance which is the best level. NoAgeMaxMin and RT were grouped in the other level, as they had no significant differences between each other but significantly different to other techniques in the best level. In conclusion, all ART methods outperformed RT in term of the fault detection capability. However, NoAgeMaxMin is grouped in the lower level, which shows that this strategy's effectiveness is the lowest by a significant margin, of the four methods compared. This is expected, as the number of categories and choices of `grep` is small, thus leading to a coarse distance measure, which we believe is particularly unfavourable for NoAgeMaxMin. The results support our intuition to develop more selection criteria since the original max-min is not good enough for a coarse distance measure. We believe that the performance of

the max-min criteria can be improved by defining a category-choice scheme having a larger number of categories and choices.

This chapter has reported the application of four variations of ART to test `grep`. The results showed that for most cases ART outperformed RT, except for NoAgeMaxMin. As previously discussed, NoAgeMaxMin may not perform well for programs with small number of categories and choices.

In this chapter, we only tested one program which is definitely not sufficient to evaluate the performance of ART methods. Therefore, in Chapter 4 we extend the application of ART methods to additional programs with complex input types.

```
ListCand:={};\\Beginning of the Main Program
for (i:=1 to n) do \\Loop to generate each category n times
{
    \\Generate Independent Category:  Category 1 to Category 7
    for (j:=1 to 7) do
    {
        VarCandidate=GenIndependentCat(j);
        AddToList(VarCandidate, ListCand);
    }
    \\Generate Dependant Category:  Category 8 to Category 14
    for (j:=8 to 14) do
    {
        VarCandidate=GenDependantCat(j);
        AddToList(VarCandidate, ListCand);
    }
} \\end loop
\\end of Main Program

\\Procedure to generate test case candidates from independent category
GenIndependentCat(CatNum as integer)
{
                ClassName=SelectClassRandomly(CatNum);
                ClassVal=SelectValRandomly(ClassName);
                return ClassVal;
} \\end procedure

\\Procedure to generate test case candidates independent category
GenDependentCat(CatNum as integer)
{
              ListClassVal := {};
              for (i:=1 to 7) do
              {
                  ClassVal=GenerateIndependentCategory(i);
                  AddToList(ClassVal, ListClassVal);
              }
              ClassName=SelectClassRandomly(CatNum);
              ClassVal=CombinePattern(ListClassVal,ClassName);
              return ClassVal;
}\\end procedure
```

Figure 3.1: Algorithm to generate the test pool for `grep`

Chapter 4

Applying FSCS-ART on SIEMENS Programs

We consider that testing one program is not enough to demonsrate the effectiveness of ART for testing programs with complex input types. Thus, in this chapter we conduct an additional case study using our methods to test three of SIEMENS programs. The same distance measure and selection criteria of FSCS-ART as introduced in Chapter 3 are applied in this chapter. The experimental setup for these artifacts are quite different with the one for `grep`, the program tested in Chapter 3. For `grep`, we create our own mutants and test pool. Whilst in this chapter, we make use of the existing test pools and faults that have been readily avaialable for the wide use of research purpose. We define a category-choice scheme for each of SIEMENS programs we use in this study. The other aspects of experimental setup remain the same as for `grep`.

4.1 Why SIEMENS Programs ?

As mentioned, the SIEMENS programs have been widely used as the experimental subjects in the software testing areas ([23], [54], [29]). They are simple text processing utilities which were originally assembled by researchers at SIEMENS Corporate Research for control-flow and data-flow test adequacy criteria [29]. The SIEMENS programs are chosen in this study for several reasons:

- They have manageable size and complexity.
- Their input formats are non-trivial, but manageable.
- Their mutants and corresponding test pools, had already been created thus avoid the issue of the experimenters selecting these to be favourable to the method and thus this avoids a potential threat to validity.

- The source codes, mutated versions, and test pools are freely available from the SIR [30].

In this chapter, we only use three of them: `printtokens`, `printtokens2`, and `replace`. `printtokens` and `printtokens2` are lexical analysers which do exactly the same things since they are based on the same specification. However, they are implemented differently and independently. Firstly, the analyser reads a file which is specified in its input parameter. Then, based on a certain scheme, the input file is split into its lexical tokens, and the output is the separated tokens. `replace` is a command-line utility which takes three inputs: a *search string*, a *replacement string*, and an *input file*. It searches for occurrences of the search string in the input file, and produces an output file where each occurrence of the search string is replaced with the replacement string. The search string is a special format of regular expression and the replacement string is a text that can include some metacharacters.

4.2 Category-choices Schemes

As mentioned, we create our own category and choice schemes for the three of SIEMENS programs that we use in this study. As other experimental materials related to this subjects are readily available to be used for the purpose of this study, only this part which is done by us regarding the experimental material used.

The category-choice schemes for these three SIEMENS programs are defined based on our understanding of what the programs do. There are no complete specifications available in SIR website [30] for the three programs. However, the source codes contain some limited information about the program specification. In addition, we observe their behaviours based on their inputs and the corresponding outputs. All these information are compiled to partition the input values into some categories and choices. The exact details of categories and choices for the three SIEMENS programs are presented in Appendix A.

4.3 Experimental Work

4.3.1 Test Pools

As mentioned, for this study, we simply use the existing pools of test cases for the three SIEMENS programs that are available in SIR. These test pools were generated by the contributors of the software artifacts. The test pool for `replace` consists of 5,542 test cases, for `printtokens` 4,130 and `printtokens2` 4,115 test cases. The fact that these pools were not created by us brings an advantage for this study which is removing a threat to external validity of this study. However, the sizes of the test

pools are relatively small which means that a significant fraction of the pool would be selected on many runs.

4.3.2 Faults

For this study, we also use the related mutated versions (hereafter are also referred to as *mutants*) present in SIR. They are seeded faults rather than real ones but were not created by us. Thus, it provides a reasonably fair testbed to evaluate our techniques. There are 7 mutated versions of `printtokens`, and 10 of `printtokens2`. There were 32 mutated versions of `replace`. However, one of `replace`'s mutants was not revealed by any of the tests in the existing test pool, so we only used 31 mutants in this experiment.

4.3.3 Experimental Design and Metric Used

Similar to the experiments using `grep`, for each SIEMENS program, we use the original version as the testing oracle. For each run, an input is selected from the pool (using RT or ART techniques) and executed on both the oracle and the mutant under testing. Then both outputs are compared. Divergent outputs indicate the detection of a failure. The number of test cases needed to detect the failure (known as the *F-count*) is then recorded. As the population distribution of the F-measure is geometric for RT and near-geometric for most ART variants [52], the standard deviation is very high. To get acceptably narrow confidence intervals, large samples are required. Therefore, to ensure the 95% confidence interval is acceptably narrow, we run the experiements 1000 times for RT, and for each of the four ART selection criteria towards each mutant. For each experimental condition, the F-measure was calculated by averaging F-counts across all the appropriate experimental runs.

4.3.4 Data and Analysis

For each SIEMENS program, the results are presented in tables containing the F-measure of RT and the F-ratio of the four ART techniques. As explained, F-ratio is the ratio of the F-measure of the four ART techniques compared to RT. Full results, including 95% confidence intervals, are presented in Appendix B.

We then perform an analysis of variance (ANOVA) on data presented in tables of first group for each SIEMENS program to test the differences between the strategies. Similar to experiment with `grep`, here, we also use F-ratio as the raw data to do ANOVA, instead of the F-measure. As discussed in Chapter 3, the reason is that the ranges of F-measure of mutants are too large and not well distributed. The F-ratio for every case of RT is set to be 1 as RT is compared to itself. We consider whether

Mutant	F-measure RT	F-ratio			
		Max-Min		Max-Sum	
		Aging	No Aging	Aging	No Aging
1	604.31	35.45%	32.73%	26.87%	24.49%
2	76.94	45.19%	46.01%	33.50%	32.74%
3	99.16	63.37%	65.56%	41.11%	40.98%
4	132.87	54.61%	45.75%	53.68%	49.51%
5	24.05	30.04%	30.11%	27.55%	27.36%
6	20.49	31.67%	31.56%	29.39%	29.10%
7	136.27	46.05%	42.43%	40.68%	39.54%

Table 4.1: F-measures of RT and F-ratios of ART for `printtokens`

different techniques result in significantly different F-ratios – which in this case reflects different failure detection capability. The null hypothesis is no differences in the F-ratios means among the four different techniques of ART. When the ANOVA rejects the hypothesis, it means that the techniques are significantly different, thus we proceed to determine which techniques contribute the most to that difference and how the technique differ from each other through a Bonferroni analysis method.

printtokens

Table 4.1 shows that all four ART methods outperform RT on all seven mutants of `printtokens`. The magnitude of improvement is quite substantial. From 24 out of the 28 cases, the F-measure of ART methods on average is about 50% of that of RT and the smallest improvement still shows a substantial difference which is about 34%.

Table 4.2 presents ANOVA results on the F-ratio of RT and the four ART strategies. We set our level of significance to be 0.05. Thus, we accept the null hypotheses if the p-value is greater than 0.05. The last column of Table 4.2 shows that the p-value is smaller than 0.0001 which is less than the level of significance. Therefore, we reject the null hypothesis and conclude that the techniques' F-ratios are significantly different. Details of how different they are and their ranks in term of their F-ratio are calculated using Bonferroni analysis, as presented in Table 4.3. Data in this table show that the RT has significant differences to all ART techniques. However, there are no significant differences among the ART techniques. The further analysis of Bonferroni analysis ranking all the techniques is described in Table 4.4. By default, Bonferroni analysis assumes that a larger F-ratio has a higher ranking. However, please note that for the F-ratio, a smaller value reflects a better failure detection capability of the strategy compared to RT. Therefore, to interpret the rank correctly for our purpose, we reverse the order of the ranking. Therefore, NoAgeMaxSum has the best F-ratio, followed by AgeMaxSum, NoAgeMaxMin, AgeMaxMin, and RT.

	SS	Degrees of freedom	MS	F	p
Strategies	4	2.112	0.528	55.081	< 0.0001
Error	30	0.288	0.010	—	—
Corrected Total	34	2.399	—	—	—

Table 4.2: ANOVA of different ART strategies for `printtokens`

Contrast	Diff.	Std. Diff.	Crit. Val.	Pr>Diff	Significant
RT vs NoAgeMaxSum	0.652	12.456	3.030	< 0.0001	Yes
RT vs AgeMaxSum	0.639	12.209	3.030	< 0.0001	Yes
RT vs NoAgeMaxMin	0.580	11.079	3.030	< 0.0001	Yes
RT vs AgeMaxMin	0.562	10.745	3.030	< 0.0001	Yes
AgeMaxMin vs NoAgeMaxSum	0.090	1.711	3.030	0.097	No
AgeMaxMin vs AgeMaxSum	0.077	1.464	3.030	0.154	No
AgeMaxMin vs NoAgeMaxMin	0.017	0.334	3.030	0.741	No
NoAgeMaxMin vs NoAgeMaxSum	0.072	1.377	3.030	0.179	No
NoAgeMaxMin vs AgeMaxSum	0.059	1.129	3.030	0.268	No
AgeMaxSum vs NoAgeMaxSum	0.013	0.247	3.030	0.806	No

Table 4.3: Bonferroni analysis of the differences between the ART strategies for `printtokens`

All ART strategies have no significant differences, thus they are grouped in the same level and RT is grouped in the lower level.

printtokens2

Like `printtokens`, for `printtokens2`, all four ART methods substantially outperform RT on all mutants, as shown in Table 4.5. The table shows that three-quarters of the comparisons shows a factor of two improvement or greater. Even the case showing the smallest improvement still has an F-measure for ART approximately 58% of that for RT.

Table 4.6 presents ANOVA results on the F-ratio of RT and the four ART strategies. We set our level of significance to be 0.05. The last column of the table shows that the p-value is < 0.0001 which is less than the level of significance. Therefore, we reject the null hypothesis and conclude that F-ratios of the techniques are dif-

Category	Average of F-ratios	Groups
RT	1	A
NoAgeMaxMin	0.438	B
AgeMaxMin	0.420	B
AgeMaxSum	0.361	B
NoAgeMaxSum	0.348	B

Table 4.4: The rank of the ART strategies using the Bonferroni analysis for `printtokens`

Mutant	F-measure RT	F-ratio			
		Max-Min		Max-Sum	
		Aging	No Aging	Aging	No Aging
1	15.7	30.90%	30.77%	29.84%	29.86%
2	15.21	30.96%	30.70%	30.23%	30.27%
3	124.74	50.90%	45.71%	54.99%	53.06%
4	11.34	35.94%	36.07%	35.04%	34.91%
5	22.35	56.80%	58.24%	46.01%	46.12%
6	7.56	50.95%	51.16%	51.13%	51.09%
7	18.13	35.53%	35.48%	34.38%	34.74%
8	15.64	37.56%	37.81%	37.56%	37.33%
9	64.62	25.72%	24.45%	22.04%	21.43%
10	22.39	56.56%	57.67%	45.83%	45.99%

Table 4.5: F-measures of RT and F-ratios of ART for `printtokens2`

	SS	Degrees of freedom	MS	F	p
Strategies	4	2.906	0.726	8.555	< 0.0001
Error	45	0.437	0.010	—	—
Corrected Total	49	3.343	—	—	—

Table 4.6: ANOVA of different ART strategies for `printtokens2`

ferent. Details of how different they are and their ranks in term of their F-ratio are calculated using Bonferroni analysis, as presented in Table 4.7. Data in the table show that the RT has significant differences to all ART techniques. However, there are no significant differences among the ART techniques. The further analysis of Bonferroni analysis ranking all the techniques is described in Table 4.8. As mentioned, to interpret the rank correctly for our purpose, we reverse the order of the ranking. In this case, NoAgeMaxSum has the best F-ratio, followed by AgeMaxSum, NoAgeMaxMin, AgeMaxMin, and RT. All ART strategies are grouped in the same level and RT is grouped in the lower level.

replace

Tables 4.9 reports that none of the ART methods were significantly more effective than RT overall on this case study. In this case, the RT and ART performances were comparable. The table shows that the ART methods outperformed RT for about half of the mutants. It may appear from the relative performance that the worst cases for ART were quite bad, but this is largely due to quoting the F-ratio; the worst cases of RT relative to ART were quite similar to the worst cases of ART relative to RT. If we were to consider the "inverse F-ratio", that is the ratio of the F-measure for RT to the F-measure for ART, for NoAgeMaxSum for mutant 25, the inverse F-ratio would be 468%.

Contrast	Diff.	Std. Diff.	Crit. Val.	Pr>Diff	Significant
RT vs NoAgeMaxSum	0.615	13.954	2.952	< 0.0001	Yes
RT vs AgeMaxSum	0.613	13.902	2.952	< 0.0001	Yes
RT vs NoAgeMaxMin	0.592	13.426	2.952	< 0.0001	Yes
RT vs AgeMaxMin	0.588	13.341	2.952	< 0.0001	Yes
AgeMaxMin vs NoAgeMaxSum	0.027	0.613	2.952	0.543	No
AgeMaxMin vs AgeMaxSum	0.025	0.562	2.952	0.577	No
AgeMaxMin vs NoAgeMaxMin	0.004	0.085	2.952	0.932	No
NoAgeMaxMin vs NoAgeMaxSum	0.023	0.527	2.952	0.600	No
NoAgeMaxMin vs AgeMaxSum	0.021	0.476	2.952	0.636	No
AgeMaxSum vs NoAgeMaxSum	0.002	0.051	2.952	0.959	No

Table 4.7: Bonferroni analysis of the differences between the ART strategies for `printtokens2`

Category	Average of F-ratios	Groups
RT	1	A
NoAgeMaxMin	0.412	B
AgeMaxMin	0.408	B
AgeMaxSum	0.387	B
NoAgeMaxSum	0.385	B

Table 4.8: The rank of the ART strategies using the Bonferroni analysis for `printtokens2`

Table 4.10 presents ANOVA results on the F-ratio of RT and the four ART strategies. We set our level of significance to be 0.05. The last column of the table shows that the p-value is < 0.0001. Thus, we reject the null hypothesis. Details of how different they are and their ranks in term of their F-ratio are calculated using Bonferroni analysis as presented in Table 4.11 and Table 4.12. Table 4.11 shows that NoAgeMaxSum has significant differences to RT and other ART techniques except AgeMaxSum. It also shows that AgeMaxSum has only significant differences to NoAgeMaxMin whilst AgeMaxMin has no significant differences to both NoAgeMaxMin and RT. The table also reports that RT has no significant difference to NoAgeMaxMin. As discussed, the ranking order in Table 4.12 should be reversed. Thus, the results show that NoAgeMaxMin has the best F-ratio, followed by RT, AgeMaxMin, AgeMaxSum, and NoAgeMaxSum. The techniques are classified into three different levels where some techniques are included in more than one level. NoAgeMaxMin, RT, and AgeMaxMin are in the highest level (level C). The lower level (level B) includes RT, AgeMaxMin, and AgeMaxSum. AgeMaxSum and NoAgeMaxSum are in the lowest level

Mutant	F-measure RT	F-ratio			
		Max-Min		Max-Sum	
		Aging	No Aging	Aging	No Aging
1	84.32	46.60%	44.78%	44.33%	41.18%
2	149.78	53.40%	49.91%	44.24%	40.54%
3	42.78	78.88%	68.47%	105.39%	103.06%
4	37.52	82.60%	73.21%	98.93%	101.60%
5	19.69	92.85%	84.76%	119.98%	124.33%
6	58.29	132.29%	136.42%	249.48%	278.98%
7	66.38	113.06%	113.01%	149.85%	167.70%
8	103.48	119.93%	82.48%	238.50%	272.08%
9	196.06	186.96%	179.87%	503.72%	529.58%
10	239.36	150.51%	145.75%	271.44%	250.39%
11	197.24	185.51%	180.37%	500.61%	566.94%
12	18.75	70.89%	70.37%	76.51%	73.98%
13	33.04	83.54%	74.86%	100.66%	104.61%
14	38.22	114.74%	109.13%	189.51%	218.44%
15	99.48	215.42%	109.91%	158.40%	178.90%
16	66.61	112.47%	113.22%	149.31%	205.78%
17	229.46	113.84%	102.69%	134.15%	141.99%
18	24.84	162.28%	159.82%	410.33%	564.91%
19	1713.70	41.94%	27.04%	42.17%	34.22%
20	254.44	107.85%	100.19%	124.57%	130.22%
21	1706.95	119.95%	48.06%	115.17%	115.68%
22	282.10	167.14%	155.67%	271.88%	278.00%
23	255.39	107.47%	97.64%	124.58%	128.74%
24	33.20	75.36%	69.45%	79.94%	81.87%
25	1757.27	31.72%	22.30%	26.13%	21.34%
26	44.99	83.77%	84.45%	125.13%	120.96%
27	21.04	65.51%	64.61%	70.92%	71.88%
28	37.01	53.27%	48.81%	61.84%	59.43%
29	88.71	53.77%	51.95%	65.49%	64.09%
30	19.86	61.58%	59.14%	70.89%	68.90%
31	24.63	163.71%	168.24%	414.77%	566.65%

Table 4.9: F-measures of RT and F-ratios of ART for `replace`

	SS	Degrees of freedom	MS	F	p
Strategies	4	22.115	5.529	5.671	0.000
Error	150	146.242	0.975	—	—
Corrected Total	154	168.357	—	—	—

Table 4.10: ANOVA of different ART strategies for `replace`

Contrast	Diff.	Std. Diff.	Crit. Val.	Pr>Diff	Significant
NoAgeMaxSum vs NoAgeMaxMin	0.912	3.636	2.849	0.000	Yes
NoAgeMaxSum vs RT	0.846	3.374	2.849	0.001	Yes
NoAgeMaxSum vs AgeMaxMin	0.798	3.183	2.849	0.002	Yes
NoAgeMaxSum vs AgeMaxSum	0.189	0.752	2.849	0.453	No
AgeMaxSum vs NoAgeMaxMin	0.723	2.884	2.849	0.005	Yes
AgeMaxSum vs RT	0.658	2.622	2.849	0.010	No
AgeMaxSum vs AgeMaxMin	0.610	2.431	2.849	0.016	No
AgeMaxMin vs NoAgeMaxMin	0.114	0.453	2.849	0.651	No
AgeMaxMin vs RT	0.048	0.191	2.849	0.848	No
RT vs NoAgeMaxMin	0.066	0.262	2.849	0.794	No

Table 4.11: Bonferroni analysis of the differences between the ART strategies for `replace`

Category	Average of F-ratio	Groups
NoAgeMaxSum	1.846	A
AgeMaxSum	1.658	A & B
AgeMaxMin	1.048	B & C
RT	1.000	B & C
NoAgeMaxMin	0.934	C

Table 4.12: The rank of the ART strategies using the Bonferroni analysis for `replace`

4.4 Summary and Discussion

In this study, we extended the use of FSCS-ART for programs with complex input types on three of SIEMENS programs: `printtokens`, `printtokens2`, and `replace`. The distance measure and selection criteria used in the relevant study on `grep` are also used in this study. However, unlike the `grep` study, here we did not create the test pools and the mutants for the three programs under testing. We simply used those that have been generated by the contributors of the software artifacts that are available in SIR [30]. However, we created the category-choice schemes for the three programs based on the limited documentation available as well as on the observation of the programs' inputs and outputs. Other experimental settings remained the same as the study on `grep`.

The experiment data in Table 4.1 and Table 4.5 reported that F-measures of all ART techniques were smaller than of RT `printtokens` and `printtokens2`. In other words, in testing `printtokens` and `printtokens2`, all ART techniques required less number of test cases than RT in revealing the first failure. However, this was not the case for `replace`. Table 4.9 showed that only 53 out of 124 cases where F-measures of ART techniques were smaller than of RT for `replace`. In this case, the ART methods outperformed RT for about half of the mutants.

We also performed an analysis of variance (ANOVA) to test the differences between the ART strategies' F-ratio. The null hypothesis was no differences in average of F-ratios of RT and ART strategies. We set our level of significance to be 0.05. Thus, we accepted the null hypotheses if the p-value was greater than 0.05. The last column of each table of ANOVA result (Table 4.2, Table 4.6, and Table 4.10) showed that p-value for each program was < 0.05. Therefore, we rejected the null hypothesis and concluded that the techniques' F-ratios were different for the three programs.

Next, we proceeded to determine which techniques contributed the most to that difference and how the techniques were different from each other through a Bonferroni analysis method. The results for `printtokens` and `printtokens2` (Table 4.3 and Table 4.7) showed that the RT has significant differences to all ART techniques but there were no significant differences among the ART techniques. However, for `replace` the results (Table 4.11) showed that NoAgeMaxSum had significant differences to RT and other ART techniques except AgeMaxSum. AgeMaxSum had only significant differences to NoAgeMaxMin. AgeMaxMin had no significant differences to both NoAgeMaxMin and RT. RT had no significant difference to NoAgeMaxMin.

Then, the further analysis of Bonferroni analysis ranked all the techniques, as described in Table 4.4, Table 4.8, and Table 4.12. For both `printtokens` and `printtokens2`, NoAgeMaxSum had the best F-ratio, followed by AgeMaxSum, NoAgeMaxMin, AgeMaxMin, and RT. For these two programs, all ART strategies had no significant differences, thus they were grouped in the same, highest level whereas RT was grouped in the lower level. For `replace`, NoAgeMaxMin had the largest F-ratio, followed by RT, AgeMaxMin, AgeMaxSum, and NoAgeMaxSum. The highest level included NoAgeMaxMin, RT, and AgeMaxMin. The lower level included RT, AgeMaxMin, and AgeMaxSum whereas AgeMaxSum and NoAgeMaxSum were in the lowest level

In summary, we conclude that ART techniques outperformed RT significantly in most cases of this study, `printtokens` and `printtokens2`. For `replace`, NoAgeMaxMin has the highest rank. AgeMaxMin, AgeMaxSum, and RT are comparable which are in the lower rank. NoAgeMaxSum is in the lowest rank. Overall, we conclude that ART techniques and RT are comparable. We believe that our definition of the category-choice scheme for `replace` affects this outcomes. Since the behaviour of `replace` is much more complex than `printtokens` and `printtokens2`, we suspect that we have not defined the scheme 100% relevant and accurate to how `replace` really works. This might cause inputs with similar behaviours are classified into different subsets (combinations of categories and choices attributed to an input) of the input domain or vice versa. Thus, the test case selections of the ART techniques might be biased hence could affect their performances.

Chapter 5

Restricted Random Testing on Programs with Complex Input Types

As mentioned in Section 1.3, existing ART strategies can be classified at least into the following groups: ART by distance, ART by restriction, and ART by partitioning. In Chapter 3 and Chapter 4, we have demonstrated the application of ART by distance (particularly FSCS-ART) on programs with complex input types. This chapter, in turn, presents the application of an ART by exclusion strategy, known as Restricted Random Testing (RRT), on programs with such input types. Here, we similarly adopt the category-choice based distance measure as introduced in the study of FSCS-ART on complex input types. We develop several techniques to adjust the original RRT algorithm to be used with the distance measure for testing programs with complex input types. We conduct some experiments to test our new RRT techniques using `grep` that has been used in Chapter 3.

5.1 Original RRT

ART is a random testing (RT) based testing method designed upon the intuition that program failures can be identified more efficiently by using more evenly distributed test cases [17]. Accordingly, the technique tries to select the next test case far away from the executed test cases. To address this, the distance-based ART such as FSCS-ART, for each iteration, attempts to select an element in the candidate set which is located furthest away from the executed test cases [17]. Another approach was developed by Chan *et al.* [19]. They used an exclusion approach to achieve an even spread of test cases while maintaining low overheads. The approach – referred to as Restricted Random Testing (RRT) – attempts to restrict some sub-areas of the input

domain from which random test cases may be generated. In this approach, at first a circular exclusion region is defined for each test case that has been executed but revealing no failures. All executed test cases have similar area of exclusion regions (say A) where the total size of them is referred to as the target exclusion (say R) which is kept to be constant for each round of selection. Then, the next test case is generated from outside of all exclusion regions. If the next test case does not reveal any failures, the next selection will have one additional exclusion region – which is for the most recently executed test case. To maintain the constant R, the area of each exclusion region (A) must be decreased. In other words, A is getting smaller when the number of executed test cases is larger. To identify the most effective value of R to be used in RRT, Chan *et al.* conducted experiments using R varying from the small one up to the maximum one [19]. From the results, they found that the most efficient error detection was obtained when R was at maximum. However, the maximum R is not always equal to the size of the input domain. This is due to the existence of the overlapping exclusion regions. Consequently, the actual target exclusion could be smaller than R or in other words the maximum R can be larger than the size of the input domain. However, in most cases, it is difficult to estimate the maximum R in advance. In addition to the findings of the best R, they also found that the fixed shape of the exclusion region might result bias on programs with non homogeneous input domain.

To alleviate problems risen in the original RRT, the further study of RRT was conducted by Chan *et al.* [55]. They introduced an enhancement of RRT, called Normalized restricted random testing (NRRT). They attempted to enhance RRT to be independent to the shape of the input domain. Using NRRT, instead of a uniform exclusion region around each executed test case, the exclusion region is scaled or mapped to the shape of the input domain. This aims to avoid bias of the fixed shape of the exclusion region on programs with non homogeneous input domains. Using this approach, NRRT can also allow an accurate estimation of the best value of R before testing begins. Thus, this delivers faster and more accurate testing process. In summary, NRRT improved the effectiveness of the RRT algorithmis by homogenizing the input domain and exclusion regions.

5.2 Proposed Algorithms of RRT for Complex Input Types

In the previous chapters, we have introduced the use of FSCS-ART on programs with complex input types, or hereafter referred to as non-numeric inputs. Previous studies on numeric inputs showed that ART by exclusion, just like ART by distance,

has been proven to ensure the good distribution of test cases while maintaining low overheads. Therefore, here we like to extend the use of ART by exclusion to programs with non-numeric inputs.

As discussed in Chapter 3, we cannot apply the Euclidean distance to measure the distance between two non-numeric inputs. As discussed, to solve this problem, we make use of the category-choices based distance measure which is adopted from the category-choices method [38]. This specification-based method defines a number of categories which are major aspects of input parameters or operational environment. All possible values of a category are divided into some disjoint groups named choices. Choices are assigned to form a basis to define the similarity or difference of non-numeric input values. Suppose, we are given two program inputs x and y. First, we need to determine categories and choices of these two inputs. Next, we count the number of categories in which x and y have different choices, say D. Then, D is defined as the distance between x and y. A greater distance represents the situation where the two inputs are more dissimilar or more evenly spread in the numeric input context. The maximum distance of two inputs is the number of categories defined for the program under test. This is obtained when two inputs do not have any similar choices at all or in other words they are totally dissimilar.

The original RRT also uses the Euclidean distance to verify whether an input is located outside the exclusion region. In the circle exclusion regions, for example, the next test case must have distance – measured using the Euclidean distance – larger than the pre-defined radius from each executed test case. Accordingly, we also need a distance measure to apply RRT on non-numeric inputs. As the category-choice based distance measure has been proved to be effective in applying FSCS-ART on non-numeric inputs, here we also adopt the same distance measure.

After having a distance measure, the next step is to define R, the target exclusion region, so that accordingly we are able to calculate the area of each exclusion region (A). Unlike in numeric input domain, in non-numeric we only have a one-dimensional "region", which is the distance. In other words, the "area" of the region is the "radius" itself. In this case, the maximum R will be the maximum distance which is the number of categories defined for the program under test. Suppose we have n executed test cases, by following the original RRT concept then the "area" an exclusion region is R/n. Thus, after n is equal to R, the size of the exclusion region would be smaller than 1, which means any test case can be selected. In other word, the next test case is selected purely randomly, similar to RT. Most of programs with non-numeric inputs do not have a very large number of categories. This causes RRT to behave like RT after a small number of test cases executed.

To avoid this situation, here we propose an adjustment of the original RRT algorithm. The basic idea is for every test case selection, instead of R to be constant,

the size of exclusion regions (A) is held constant. As explained, this is due to the small ranges of possible R. In details, firstly, we try to find a test case having a distance larger than a pre-defined distance (A) to all executed test cases (hereafter A is referred by *LimD*). At the beginning of the selection process (referred to as *InitStep*), LimD is initialised with *InitLimD* which is a constant. However, as instead of "generating" test cases during the selection process, we pick them randomly from a pool of pre-generated test cases, it is possible that we do not find the test case satisfying the criteria in the first pick. If this happens, we select another test cases randomly until a test case satisfying the criteria is found or after N trials have been attempted. We limit the number of trials to control the computational time since there is no guarantee that a test case satisfying the criteria s exist in the test pool. If no test case satisfying the criteria s are found after N trials, then LimD is reduced by one. The process is repeated unless a test case satisfying the criteria is found or until LimD is equal to -1. Once LimD reaches -1, the selection process is identical to RT which means any random test case will be accepted as the test case satisfying the criteria . If the test case satisfying the criteria is found then it is selected to be the next test case and then executed. If a failure is revealed, the process stops. Otherwise, the *InitStep* is repeated where LimD is set back to InitLimD. The new RRT algorithm is detailed in Figure 5.1

Followings are the key variables to assist in understanding more details of the proposed algorithm:

- **InitStep**. This is the first step of selecting the next test case. After a test case satisfying the criteria is found, then it is selected and executed as the next test case but revealing no failures, the process should be back to this step.

- **LimD** (Limit of Distance). As explained, this has similar concept with the size of the exclusion region (A) in the numeric context. The next test case must have distance larger than LimD to all executed test case.

- **MaxLimD** (Maximum LimD). This is the maximum value of LimD, which is the number of categories defined for the program under test minus one.

- **InitLimD** (Initial LimD). This is the initial value of LimD which is set constant in every InitStep.

- **Trial**. This is an attempt to randomly select a test case from the input domain. If the test case has distance larger than LimD to all executed test case, it will be chosen as the next test case to be executed. Otherwise, we do another trial.

- **MaxTrial** (Maximum Trial). This is the maximum number of trials attempted to find a test case having distance larger than LimD before LimD is reduced

```
1Begin
2   Set E empty ** E is a set to store the executed test cases
3   Set MT = MaxTrial ** a variable to set the maximum trial
4   Select randomly a test case t from the input domain
5   Execute t
6   If t reveals a failure
7       Return the size of E as the F-measure
8       Terminate the program
9   Else
10      Add t in E
11  End If
12  Set LimD = InitLimD ** the initial value of LimD
13  Set Trial = 0
14  Set Found = false ** a variable inidicating a trial successful or not
15  While(Trial <= MT)or(Found = false) **until Max MT or t satisfying the criteria
16      Select t randomly from the input domain
17      If the distance of t to any of elements of E is <= LimD
18          Trial = Trial + 1
19      Else ** the distance of t to all elements in E > LimD
20          Found = true
21          Terminate the Loop and GoTo Line 5
22      Endif
23  End While
24  If Trial > MT
25      LimD = LimD - 1
26      GoTo Line 13
27  End If
28End
```

Figure 5.1: RRT algorithm for testing programs with complex input types

by one.

From the algorithm above, we observe that values of certain key parameters may significantly affect the performance of RRT on non-numeric input types. Specifically, we need to answer the following questions to apply the method effectively:

- What is the best value of MaxTrial ?
- What is the best value of InitLimD ?
- We also note that the aging technique (see Chapter 3) is applicable. Does it also improve the effectiveness of RRT ?

Following are some discussion related to the three questions above:

- **The best value of MaxTrial**. Intuitively, a larger number of MaxTrial will give a higher chance to find the test case satisfying the criteria but might be

more expensive. However, setting MaxTrial to be too large might be useless. This is due to the fixed and limited size of the test pool size, particularly when the size of E is large. At this stage, the executed test cases have been well spread over the input domain. Thus, it might be very few or no test cases located far from the executed test cases.

- **The best value of InitLimD**. Setting InitLimD to MaxLimD means attempting to search for a furthest away test case from E. We believe this choice will maximize the evenspreadness of test cases.

- **The impact of aging technique**. Considering aging method means only the n most recently executed test cases are included in the selection process. As mentioned, the possible values of distances on non-numeric inputs depends on the number of the categories defined. However, most of programs with non-numeric inputs do not have a very large number of categories. Hence, the possible distance values are only integers ranging from 0 to the number of the categories defined. Therefore when the number of the executed test cases is very large (which is when the number of failure causing inputs are very few), the LimD will be set to -1 to find the test case satisfying the criteria . By applying aging technique, the number of executed test cases that is included in the selection process is restricted. Thus, the even spreadness concept is still achievable. The lower overhead is the other benefit of applying aging technique.

5.3 Experimental Work

5.3.1 Experimental Artifacts

We use `grep` as the experimental subject. Details of experimental setup are basically the same as those in Section 3.2. The test pool and the twenty mutants of `grep` are also reused in this chapter. As we also like to compare the perfomance of FSCS-ART and RRT on non-numeric inputs, using exactly the same experimental artifacts will achieve consistent, accurate, and fair comparisons. The reason of using `grep` but not the three SIEMENS programs (`printtokens`, `printtokens2`, and `replace`) is that the inputs of the test pool were generated based on the uniform distribution where each category contributed equally (details are presented in Section 3.2.3). This ensures that all elements in the test pool are well spread. However, this is not guaranteed for the test pools of the SIEMENS as they were not created by us. Since RRT is an exclusion based method, the uniform distribution of the inputs across the test pool is crutial to follow the intuition behind the RRT concept.

5.3.2 Experimental Design

As discussed, there are three aspects that might affect the performance of RRT on non-numeric inputs: the value of MaxTrial, InitLimD, and the aging technique. To evaluate our hypotheses according these aspects, we setup three different groups of experiments, each investigates one aspect. To save time and resources, we conduct them consecutively. The result of one aspect, can be used in the next aspect. As we do not have any initial indication of the best MaxTrial values, we choose this as the first aspect to be investigated. Next, from the findings, we continue to the InitLimD aspect. Finally, we conduct experiments of the aging aspect. According to this sequence, we define three groups of experiments: Group-MaxTrial(GMT), Group-InitLimD (GIL), and Group-Aging (GA).

We conduct experiments in Group-MaxTrial(GMT) to determine the most effective and efficient value of the MaxTrial. To give consistent evaluation, we set InitLimD constant and use the no-aging method while we use different values of MaxTrial. We choose InitLimD = MaxInitLimD as this is the most straightforward approach and no-aging technique which follows the original concept of ART. In this group, we investigate the different values of Maxtrial: 10, 25, 100, and 200.

The experiments in Group-InitLimD (GIL) aim to identify the best value of InitLimD to get the best performance of RRT in term of its effectiveness and efficiency. Similar to GMT experiment, here we set the other aspects constant. As mentioned, to save time and resource, we do the experiments after completing GMT experiments. We set MaxTrial with the best value found from GMT. With the same reason as in GMT, we also use no-aging technique here. In this group, we investigate the following different values of InitLimD: MaxLimD, 0.5*MaxLimD, and ProportionalInitLimD. For ProportionalInitLimD, the InitLimD is reduced proportionally to the increase of the number of executed test cases.

From the experiments in Group-Aging (GA), we like to identify the influence of the aging in term of the effectiveness of RRT to reveal any failures. Therefore, we use the aging approach here by only considering the 10 most recently executed test cases. This number is chosen as it has been used and proved to be sufficiently effective in the previous study of ART [18]. In these experiments, we set MaxTrial with the best value found from GMT and InitLimD with the best value found from GIL.

5.3.3 Metric

Here, we use the same metric used for evaluating the FSCS-ART on non-numeric input types, referred to as F-measure. As mentioned, the original version of `grep` and its twenty mutants presented in Chapter 3 are reused here. For each experi-

ment, we use the original version of `grep` as the test oracle and each of the mutant as the program under test. For each run, an input is selected from the pool using RRT techniques and executed on both the oracle and the mutant under testing. Then both outputs are compared. The different outputs indicate the detection of a failure. The number of test cases needed to detect the failure (known as the *F-count*) is then recorded. As studied by Chen *et al.* [52], the population distribution of the F-measure is near-geometric for most ART variants. Thus, the standard deviation is very high and thus large samples are required to get acceptably narrow confidence intervals. Therefore, to satisfy the 95% confidence interval, we run the experiments 1000 times for each of RRT technique. For each experimental condition, the F-measure was calculated by averaging F-counts across all the appropriate experimental runs.

5.3.4 Experiment Results

According to our experimental setup, there are some dependencies among the three group experiments. GIL makes use of findings in GMT and GA makes use of findings in GMT and GIL. Therefore, we conduct the GMT experiments first, and then followed by GIL and GA, consecutively. To evaluate RRT, we compare the F-measure of RRT to the F-measure of RT of a certain mutant, which is referred to as F-ratio. From experiments in Chapter 3, we know that the maximum F-measure of RT for mutants of `grep` is about 857.59. To control the unexpected large F-measure of RRT, in the implementation we restrict the number of executed test cases to 5000 (which is about six times the maximum F-measure). After the testing reaches 5000 test cases but reveals no failure, it is stopped. In this case, we just conclude that the F-measure is extremely large which is reported as >>> in the tables of experiment results.

For each group of experiments, we present tables containing the F-ratio of RRT techniques to RT. As explained in the previous chapters, F-ratio is the ratio of the F-measure of RRT compared to RT. Full results of F-measure of each group are presented in Appendix C.

GMT Experiment Results

Table 5.1 shows the F-ratio of RRT in GMT-experiments. The table reports the F-ratio of different MaxTrial values for each mutant. We conduct GMT with five different values of MaxTrial: 10, 50, 75, 100, and 200. As explained, the experiments in this group do not use aging technique, and InitLimD is set to MaxLimD. From the results, we identify that the F-ratio of most mutants with different MaxTrial are not significantly different. Using ANOVA (Table 5.4) and Bonferroni analysis(Table 5.5),

M-ID	MT=10	MT=25	MT=50	MT	MT=100	MT=200
Mutant1	29.59%	26.88%	27.24%	28.38%	28.09%	27.52%
Mutant2	>>>	>>>	>>>	>>>	>>>	>>>
Mutant3	31.16%	31.30%	31.51%	32.55%	31.23%	30.33%
Mutant4	118.84%	115.12%	110.62%	112.48%	115.58%	110.07%
Mutant5	27.70%	26.68%	27.62%	28.87%	28.85%	27.22%
Mutant6	166.83%	159.00%	160.52%	155.86%	160.70%	154.51%
Mutant7	32.63%	29.67%	30.50%	31.01%	30.86%	30.44%
Mutant8	31.59%	31.31%	30.79%	31.57%	28.94%	30.35%
Mutant9	30.29%	29.46%	28.55%	28.84%	28.05%	28.33%
Mutant10	29.04%	27.40%	27.62%	26.62%	27.62%	28.11%
Mutant11	36.01%	34.48%	34.11%	35.25%	33.24%	35.27%
Mutant12	31.98%	30.36%	29.10%	29.48%	30.02%	29.71%
Mutant13	31.56%	27.87%	26.32%	26.51%	26.93%	27.44%
Mutant14	29.77%	28.15%	29.25%	27.13%	27.86%	27.79%
Mutant15	33.59%	32.42%	29.72%	30.24%	31.16%	29.84%
Mutant16	29.32%	28.62%	28.90%	29.18%	27.71%	27.29%
Mutant17	32.83%	30.26%	32.41%	29.56%	29.14%	29.99%
Mutant18	41.68%	37.35%	36.33%	37.95%	38.21%	36.27%
Mutant19	103.34%	101.19%	97.94%	98.03%	94.24%	89.35%
Mutant20	37.00%	37.00%	39.50%	38.67%	38.67%	39.17%

Table 5.1: GMT experiment results:Average F-ratio of RT and RRT with various number of MaxTrial (MT) values for `grep`. (Note that "MT=n" denotes that the MaxTrial used for the related experiment is n)

we found that RT has significant difference to RRT, however different MaxTrial values do not affect the F-ratio of RRT significantly. As a larger MaxTrial requires more overhead, we choose MaxTrial=10 as the MaxTrial setting in experiments in GIL and GA.

GIL Experiment Results

Table 5.2 shows the f-ratio of RRT in GIL-experiments. The table reports the F-ratio of different InitLimD settings for each mutant. We conduct experiments with different InitLimD settings: InitLimD=MaxLimD, InitLimD=0.5*MaxLimD, and InitLimD=Proportional. As explained, the experiments in this group do not use aging technique and MaxTrial is set to 10. Using ANOVA (Table 5.6) and Bonferroni analysis(Table 5.7), we found that RT has significant difference to RRT, however different values of InitLimD do not affect the F-ratio of RRT significantly. The table reports that RRT with InitLimD=MaxLimD has the highest rank (as explained in previous chapters, the ranking order is reversed for the ANOVA of F-ratios). Thus, for the experiment in GA, we set InitLimD to be equal to MaxLimD.

M-ID	ILD=MaxLimD	ILD=0.5*MaxLimD	ILD=Proportional
Mutant1	26.88%	36.74%	35.00%
Mutant2	>>>	>>>	>>>
Mutant3	31.30%	39.42%	38.21%
Mutant4	115.12%	120.44%	118.36%
Mutant5	26.68%	31.29%	31.67%
Mutant6	159.00%	177.98%	164.78%
Mutant7	29.67%	32.74%	34.09%
Mutant8	31.31%	32.21%	30.56%
Mutant9	29.46%	29.19%	30.99%
Mutant10	27.40%	35.62%	35.90%
Mutant11	34.48%	35.31%	32.58%
Mutant12	30.36%	26.58%	29.16%
Mutant13	27.87%	29.67%	28.71%
Mutant14	28.15%	36.60%	36.00%
Mutant15	32.42%	31.89%	31.73%
Mutant16	28.62%	35.96%	34.97%
Mutant17	30.26%	28.99%	25.73%
Mutant18	37.35%	41.71%	39.27%
Mutant19	101.19%	107.43%	101.12%
Mutant20	37.00%	54.33%	54.18%

Table 5.2: GIL experiment results:Average F-ratio of RT and RRT with different values of InitLimD (ILD) for `grep`. (Note that "ILD=n" denotes that the InitLimD used for the related experiment is n)

M-ID	NoAging	Aging
Mutant1	26.88%	58.68%
Mutant2	>>>	51.06%
Mutant3	31.30%	58.10%
Mutant4	115.12%	46.65%
Mutant5	26.68%	54.81%
Mutant6	159.00%	44.83%
Mutant7	29.67%	59.41%
Mutant8	31.31%	55.97%
Mutant9	29.46%	58.94%
Mutant10	27.40%	59.22%
Mutant11	34.48%	53.57%
Mutant12	30.36%	50.85%
Mutant13	27.87%	54.25%
Mutant14	28.15%	60.78%
Mutant15	32.42%	52.22%
Mutant16	28.62%	58.85%
Mutant17	30.26%	56.36%
Mutant18	37.35%	50.67%
Mutant19	101.19%	40.51%
Mutant20	37.00%	22.83%

Table 5.3: GA experiment results: Average F-ratio of RT and RRT with NoAging and Aging approaches for `grep`

	SS	Degrees of freedom	MS	F	p
Model	6	4.858	0.810	7.074	< 0.0001
Error	126	14.420	0.114	—	—
Corrected Total	132	19.278	—	—	—

Table 5.4: ANOVA of GMT experiments

GA Experiment Results

Table 5.3 shows the F-ratio of RRT in GA-experiments. The table reports the F-ratio of RRT without aging and RRT with aging. As explained, the experiments in this group do not use aging technique and MaxTrial is set to 10. From the table, we identify that for all mutants, RRT without aging outperforms RRT with aging, except for mutant-2, mutant-4, mutant-6, and mutant-19. For the latter four mutants, RRT without aging performs worse than RT. However, RRT with aging outperforms RT for all mutants. Using ANOVA (Table 5.8) and Bonferroni analysis(Table 5.9), we found that RT has significant difference to RRT, however RRT without aging and RRT with aging do not have significant differences.

Category	Average of F-ratios	Groups
RT	1.000	A
MT10	0.476	B
MT25	0.455	B
MT50	0.452	B
MT75	0.452	B
MT100	0.451	B
MT200	0.442	B

Table 5.5: The rank of RRT with different settings in GMT

	SS	Degrees of freedom	MS	F	p
Model	3	3.962	1.321	12.281	< 0.0001
Error	73	7.849	0.108	—	—
Corrected Total	76	11.811	—	—	—

Table 5.6: ANOVA of GIL experiments

Category	Average of F-ratios	Groups
RT	1.000	A
HalfMaxLD	0.507	B
Proportional	0.491	B
MaxLimD	0.455	B

Table 5.7: The rank of RRT with different settings in GIL

	SS	Degrees of freedom	MS	F	p
Model	2	3.478	1.739	37.410	< 0.0001
Error	56	2.603	0.046	—	—
Corrected Total	58	6.081	—	—	—

Table 5.8: ANOVA of GA experiments

Category	Average of F-ratios	Groups
RT	1.000	A
Aging	0.524	B
NoAging	0.455	B

Table 5.9: The rank of RRT with different settings in GA

5.4 Data Analysis

From the Table 5.1, the results of GMT reported that RRT outperformed RT significantly by about 30% except for mutant-2, mutant-4, mutant-6, and mutant-19. For mutant-2, the F-measure of RRT is >>> for all settings of MaxTrial. This means that the F-measure is larger than 5000. As explained, the testing would be stopped after the number of executed test cases reaches 5000, due to the control of the computational cost. For this case, we just conclude that the F-measure is extremely large. For mutant-4 and mutant-19, the perfomance of RRT techniques in GMT is comparable to RT. For mutant-6, RRT is worse than RT for about 160%. As reported by the Bonferonny Analysis, different settings of MaxTrial did not give significant impact to the effectiveness of RRT.

Data of GIL experiments, as shown in Table 5.2, are similar to of GMT. RRT also outperformed RT significantly by about 30% except for mutant-2, mutant-4, mutant-6, and mutant-19. The effectiveness of RRT for the latter four mutants is similar with in GMT. As reported by the Bonferonny Analysis, different settings of InitLimD did not give significant impact to the effectiveness of RRT.

The results of GA, as shown in Table 5.3, are quite different with of GMT and GIL experiments. RRT with aging outperformed RT by about 60% for all mutants, including mutant-2, mutant-4, mutant-6, and mutant-19. However, the results reported that RRT without aging outperformed RRT with aging by about 50% except for mutant-2, mutant-4, mutant-6, and mutant-19. Please note that RRT without aging in this group is similar to RRT in GMT with MaxTral=10 and similar to RRT in GA with InitLimD=MaxLimD. The outcome that RRT with aging could outperform RT for mutant-2, mutant-4, mutant-6, and mutant-19 is surprising. The results of RRT in other groups showed that for these mutants the performance of RRT was always similar or worse than RT. This finding is very interesting to be further investigated.

5.5 Summary and Discussion

RRT is a new approach for ART on programs with complex input types. As ART by exclusion has been proven to be effective in testing programs with numeric inputs while maintaining low overheads, thus here we like to extend the use of ART by exclusion to programs with non-numeric inputs.

Accordingly, in this chapter we present a new algorithm of RRT on program with non-numeric input types. The algorithm also uses the category-choice-based distance measure as used in FSCS-ART on the same type of programs. The maximum target exclusion region (R) in the orginal RRT is relevant to the number

of categories defined for a program with non-numeric inputs. As this number is relatively small, we cannot follow exactly the approach used in RRT for testing programs with numeric inputs. Here we propose an adjustment of the RRT algorithm by maintaining the exclusion region (A) to be constant. The exclusion region in this context is the distance itself, referred to as LimD. For each round, a test case is randomly selected and checked whether it is outside the exclusion regions which is true if its distance is larger than LimD to all executed test cases. However, if there are no test case satisfying the criteria s after a certain number of trials, LimD is reduced gradually.

As discussed in Section 5.2, there are several key parameters which could affect the effectiveness of RRT. Followings are some discussions of the results related to those key parameters:

1. The best value of MaxTrial. The Bonferroni analysis of GMT experiments (Table 5.5) reported that RRT with larger MaxTrial had higher rank. This is consistent with our intuition that a larger number of MaxTrial will give a higher chance to find the test case satisfying the criteria. However, the results showed that differences of their perfomances are not significant.

2. The best value of InitLimD. The Bonferroni analysis of GIL experiments (Table 5.7) reported that RRT with larger InitLimD had higher rank. This confirms our intuition that a larger InitLimD will give more well spread of test cases. However, the results showed that differences are not significant.

3. The impact of aging technique. As explained, we argue that aging approach is useful in the situation where the F-measure is very large (the number of failure causing inputs are very small). This is confirmed by the experiment results in the Table 5.3, particularly for the case of mutant-2. For this mutant, aging outperfomed no aging by about 50%. However, for mutant-4, mutant-6, and mutant-19, even these mutants have relatively small F-measure, aging also could help to improve the performance of RRT. We suspect that this is due to the special characteristics of the failure causing inputs of these mutants.

In summary, we conclude that for any settings of Maxtrial and InitLimD, RRT without aging outperformed RT for about 30% except for mutant-2, mutant-4, mutant-6, and mutant-19. However, aging technique could improve the performance of RRT for these mutants up to 40% of RT. This shows that RRT in this study gives significant contribution in improving RT. In Chapter 3, Table 3.8 showed that NoAgeMaxSum has the best performance of all other FSCS-ART techniques. Table B.3 reported that the average F-ratio of NoAgeMaxSum is about 40% except for mutant-20 which is about 99%. This showed RRT without aging outperformed NoAgeMaxSum for most mutants, except mutant-2, mutant-4, mutant-6,

and mutant-19. However, for those mutants, F-ratio RRT with aging is about similar to NoAgeMaxSum. So in general, we conclude that RRT outperformed FSCS-ART for testing `grep`.

Surprisingly, the performance of NoAgeMaxMin in Chapter 3 for mutant-2, mutant-4, mutant-6, and mutant-19 were also worse than RT, similar to RRT without aging. We believe that this is not a coincidence. We suspect that this is due to special characteristics of the failure causing inputs in those mutants. Hence, further investigation of this matter is conducted and the results will be reported in Chapter 6.

In this chapter, we have demonstrated that the new approaches of RRT have been effective in testing `grep`. As this is the preliminary study of such application of RRT, there are still many aspects for further improvement of the methods.

Chapter 6

Failure Patterns of Complex Input Types

The next contribution of this thesis is the identification of failure causing input (FCI) patterns in programs with complex input types. Similar to programs with numeric inputs, we like to investigate the regularity of FCIs in this type of programs. Here, we make use of the experimental data and results of the previous chapters (study of FSCS-ART and RRT on `grep`). In particular, we examine the FCIs of all mutated versions and the performance of the ART techniques in those chapters. In those studies, we found that the results of a few mutants are different, which are inconsistent with other mutants. We suspect that this is due to the special characteristics of FCI in those few mutants. We like to identify these characteristics and how they affect the behaviour of our ART techniques.

6.1 Introduction

Failure causing inputs (FCIs) of a program under test are inputs which cause the program to fail. In the numeric context, Chan *et al.* [28] observed some common types of errors that would give rise to some regularities. Accordingly, they identified three groups of "failure patterns": the block pattern, the strip pattern, and the point pattern. The block pattern represents a single compact and contiguous region of FCIs in the program under test's input domain. The strip pattern is for a thin and long contiguous region, and the point is for many number of small regions where each region contains one or a very few number of FCIs. In the numeric context, ART techniques perform well when block and strip patterns are present as it supports the basic intuition of ART that two test cases that are close to each other are more likely to have the same failure behaviour than two test cases that are widely spaced.

In this thesis, we examine the regularity of FCI types in programs with complex

input types. Then, we investigate how our ART techniques behave towards each type of the identified FCI patterns. This study is motivated by the analysis of results in Chapter 3 and Chapter 5. In these chapters, we found that the results of a few mutants of `grep` are "different", which are inconsistent with most mutants. The other surprising finding is that for those few mutants, NoAgeMaxMin behaved similarly to RRT with no aging. We suspect that this is due to the special characteristics of FCI in those few mutants.

In this study, we investigate the FCI distribution of all mutants of `grep` in the input domain determined by the test pool as detailed in Section 3.2.3. In particular, we are interested to find out the distribution of the *failure subdomains* (FSD) – partitions of the input domain which contain FCI – of `grep`'s mutants. We believe that our ART techniques behave differently to different distributions of FSD. We expect that we could use this finding to improve the effectiveness of our existing ART methods in detecting failures on programs with non-numeric inputs.

6.2 Design of the Study

As mentioned above, in this study we make use of the experimental data gathered from experiments in Chapter 3 and Chapter 5 as the basic resources to conduct this study. Specifically, we make use of the data of FCI of the twenty mutants of `grep` used in those chapters. We also consider the F-measure data of RT and our ART techniques for those mutants to be correlated with the FCI data.

Here, we introduce some new notations: *test frame*, *subdomain*, and *failure subdomain*. We describe a test frame as a combination of choices specified for an input of a program under test. A test frame of an input is produced from the category-partition scheme defined for the program under test. Based on the scheme, we identify which choices of the categories the input fits in. The combination of the identified choices is the test frame of the input. It is possible that more than one test case lies in a test frame. Then, we state that the test cases fitting into the same test frame are located in the same subdomain of the input domain. In other words, a subdomain is a fraction of the input domain where all inputs sharing the same test frame are located. Thus, if there are n distinct test frames identified for all inputs in an input domain then there are also n subdomains in the input domain. A subdomain is then referred to as a failure subdomain (FSD) if it contains at least one FCI.

Firstly, this study considers the distribution of the FSD in the input domain of all mutants of `grep`. Unlike in the numeric context, it is difficult to describe the distribution of non-numeric inputs. Therefore, here we attempt to show the distribution of such inputs by examining the following components of an FSD: the

distance, the size and the failure rate. In general, the distance between two subdomains is actually the distance between any two test cases located in the subdomains. In other words, the distance is the number of categories in which the test frames of both subdomains have different choices. The failure rate of a subdomain is the ratio of the failure causing inputs to the total number of inputs in the subdomain. The non-failure subdomains have zero failure rate. If all inputs in a (failure) subdomain are failure causing inputs, then its failure rate is 1.

6.3 Data and Analysis

As mentioned, to identify the regularities of failure causing inputs in `grep`, we observe the failure subdomains distribution for all mutants of `grep`. As mentioned, there are three components that we use as the basic attributes of the distribution of an FSD: the distance, the size, and the failure rate. The distance of an FSD refers to the average distances between the FSD and all other FSD in the input domain. The size of an FSD is the number of inputs in the FSD whereas the failure rate is the ratio of the number of FCI to the size of the FSD.

We classify an FSD based on the three components mentioned, as follow:

1. Distance. We split the category into three choices: compact, medium-spread, well-spread. As the maximum distance of two subdomains are 14 (which is the number of category in `grep`'s category-choice scheme), each choice is in interval of 14/3 which is about 4.67

    - Compact. An FSD is grouped in this class if its average distance is between 0 and 4.67
    - Medium-spread. An FSD is grouped in this class if its average distance is between 4.67 and 9.33.
    - Well-spread. An FSD is grouped in this class if its average distance is greater than 9.33.

2. Size. We split the category into four choices: dot, small-size, medium-size, and large-size. As the sizes of the FSD have a huge range, we define each choice as follows:

    - Dot. An FSD is grouped in this class if its size is only 1.
    - Small-size. An FSD is grouped in this class if its size is betwen 2 and 100
    - Medium-size. An FSD is grouped in this class if its size is between 100 and 1000

- Large-size. An FSD is grouped in this class if its size is only greater than 1000

3. Failure rate. We split the category into three choices: low-frate, medium-frate, and high-frate. As the range of the failure rate is between 0 and 1, each choice is in interval of 1/3 which is about 0.33

   - Low-frate. An FSD is grouped in this class if its failure rate is smaller than 0.33
   - Medium-frate. An FSD is grouped in this class if its failure rate is between 0.33 and 0.67
   - High-frate. An FSD is grouped in this class if its failure rate is larger than 0.67

The classification above is specific to our category-partition scheme for `grep`, that is presented in Chapter 3. To be applied in other program, the classification somehow needs to be adjusted.

Table 6.1 reports the average distance of all FSD, Table 6.2 shows the average size of all FSD, and Table 6.3 presents the average failure rate of all FSD for each mutant of `grep`. From Table 6.1, we identify that all mutants have average distances about 8 to 9 (medium-spread), except for mutant-2 which is 2.91 (compact). Table 6.2 shows the average size of FSD for each mutant of `grep`. It is surprising that the average FSD size for all mutants is small, and for most of them, the size is 1. It means that for most FSD of all mutants, each only contains one input which is FCI. However, for mutant-2, mutant-4, mutant-6, mutant-19, and mutant-20, their average FSD sizes are relatively larger than others. Table 6.3 shows that all mutants have average failure rates about 0.9 (high-frate), except for mutant-2, which is 0.09 (low-frate). For mutant-4, the average is 0.71 which is high-frate but relatively smaller than the failure rates of other mutants.

To further investigate the distribution of FSD towards the combination of the classifications above, we present Table 6.4 containing detailed types of all FSD for all mutants of `grep`. We do not include the distance component here, as from Table 6.1, we identify that in average all mutants have well-spread FSD except mutant-2 which has compact FSD. Accordingly, to simplify the presentation of data in Table 6.4, we assume that the FSD in all mutants are well-spread except in mutant-2 which is compact. In the table, column DOT refers to the combination of two classes: dot and high-frate. The reason is that all dot-type FSD must be high-frate as their failure rates are 1. The table reports that all mutants have a large number of dot-type FSD except mutant-2 which has no dot FSD at all. Mutant-2 only has 11 FSD with small-size and low-frate and 1 FSD with large-size and low-frate . Mutant-1, mutant-5,

M-ID	Average FSD Distance
Mutant1	8.91
Mutant2	2.91
Mutant3	8.91
Mutant4	9.25
Mutant5	8.91
Mutant6	8.38
Mutant7	8.76
Mutant8	8.86
Mutant9	8.50
Mutant10	8.91
Mutant11	8.62
Mutant12	8.63
Mutant13	8.74
Mutant14	8.95
Mutant15	8.99
Mutant16	8.90
Mutant17	8.96
Mutant18	8.91
Mutant19	8.26
Mutant20	9.28

Table 6.1: Average Distance of FSD of each mutant for `grep`

mutant-8, mutant-9, mutant-10, mutant-12, mutant-13, mutant-14, mutant-16, and mutant-17 have no medium-size and large-size FSD at all. They only have dot and small-size FSD. Other mutants have at least one large-size FSD.

Table 6.1 and Table 6.3 report that mutant-2 has distinct average distance and average failure rate of FSD to the other mutants. Table 6.2 shows that mutant-2, mutant-4, mutant-6, mutant-19, and mutant-20 have average size of FSD > 2 where others are only about 1. Moreover, mutant-2 has the largest average size which is 14. Table 6.4 reports that mutant-2, mutant-3, mutant-4, mutant-6, mutant-7, mutant-11, mutant-15, mutant-18, mutant-19, and mutant-20 have large-size and medium-size FSD. Others have small-size FSD only. Results in Chapter 3 and Chapter 5 show that the problematic mutants towards our ART techniques (particularly, NoAgeMaxMin and RRT without aging) are mutant-2, mutant-4, mutant-6, mutant-19, and mutant-20 (the latter one is only for NoAgeMaxMin). Particularly, for RRT without aging, mutant-2 performed much worse than RT. This is consistent to our finding which shows that mutant-2 is the most distinct mutant with respect to its FSD characteristics. Other mutants performing bad (mutant-4, mutant-6, mutant-19, and mutant-20) also have distinct characteristics of FSD compared to other mutants.

M-ID	Recap. Sizes	Number of FSD	Average Size
Mutant1	12071	11913	1.013
Mutant2	176	13	13.54
Mutant3	11819	10384	1.14
Mutant4	3668	1385	2.65
Mutant5	4759	4733	1.01
Mutant6	5041	943	5.35
Mutant7	5111	4772	1.07
Mutant8	2916	2899	1.01
Mutant9	3509	3494	1.01
Mutant10	12080	11922	1.01
Mutant11	836	736	1.14
Mutant12	367	367	1
Mutant13	266	265	1.01
Mutant14	12503	12283	1.02
Mutant15	619	611	1.013
Mutant16	12054	11896	1.01
Mutant17	364	364	1
Mutant18	4905	3755	1.31
Mutant19	7890	2243	3.52
Mutant20	75477	13349	5.65

Table 6.2: Average FSD size of each mutant for `grep`

6.4 Discussions

As mentioned, mutant-2 has different characteristics of FSD among all mutant. Mutant-2 has the smallest average distance showing that its FSD are very compact. This shows that they are located close to each other. Regarding the FSD size and failure rate, mutant-2 has the largest average of FSD size (large-size) but the smallest average failure rate (low-frate). In other words, for mutant-2, all FCIs are located in FSDs which are located very close to each other. However, those FSDs have very small failure rates which means the ratio of the FCIs to the non-FCIs in the FSD is very small.

Our ART techniques are based on the intuition to select the next test case far away from the executed test cases. As the average of FSD sizes of mutant-2 is large, any input in such FSD has high chance to be selected randomly as a candidate in FSCS-ART. As the failure rate is small, the non-FCI inputs have higher chances to be selected than the FCIs in such FSD. Once a non-FCI of such FSD is selected and executed, any inputs from that FSD or also from all other FSDs (since all of them are very close to each other) are much less likely to be selected by our original FSCS-ART techique, NoAgeMaxMin. Therefore, NoAgeMaxMin required much more test cases than RT to reveal the first failure in mutant-2.

M-ID	Average FSD Failure Rate
Mutant1	0.99
Mutant2	0.09
Mutant3	0.98
Mutant4	0.71
Mutant5	0.98
Mutant6	0.96
Mutant7	0.98
Mutant8	0.98
Mutant9	0.98
Mutant10	0.99
Mutant11	0.95
Mutant12	0.97
Mutant13	0.98
Mutant14	0.97
Mutant15	0.96
Mutant16	0.99
Mutant17	0.98
Mutant18	0.98
Mutant19	0.98
Mutant20	0.99

Table 6.3: Average Failure Rate of FSD of each mutant for `grep`

However, aging technique can solve such problem. Aging only includes the n most recently executed test cases in the selection process. Therefore, even though a non-FCI of such FSD is selected and executed, after the execution of the next n test cases, it will be disregarded as the executed test cases. Thus, the chance of the FCIs to be selected as the next test cases will be equal to the non-FCIs of such FSD.

The similar explanation is also applied for RRT without aging. Once a non-FCI of such FSD is selected and executed, all FCIs of mutant-2, which are very close to that non-FCI would be excluded. This causes the performance of RRT without aging is much worse than RT in revealing the first failure. However, aging technique will solve this problem as this approach will ignore an executed test case after the execution of the next n test cases.

Other mutants on which NoAgeMaxMin and RRT without aging performed worse than RT (mutant-4, mutant-6, mutant-19, and mutant-20) also have large-size FSDs with low or medium failure rates. The similar explanation as for mutant-2, is also applied for these mutants. Once a non-FCI of an FSD is selected, all FCIs in the FSD will be less likely to be chosen as the next test cases. This causes the performance of the ART techniques is worse RT. However, the performance for these mutants is not as bad as for mutant-2. The reason is that for these mutants, there

Mut	DOT	Small LowFr	Small MedFr	Small HighFr	MedSize LowFr	MedSize MedFr	MedSize HighFr	Large LowFr	Large MedFr	Large HighFr
Mut1	11430	6318	157	0	0	0	0	0	0	0
Mut2	0	11	0	0	0	0	0	1	0	0
Mut3	9943	5	306	127	0	0	0	0	1	0
Mut4	940	359	54	2	22	1	0	3	0	0
Mut5	4514	0	191	26	0	0	0	0	0	0
Mut6	869	1	64	3	0	2	0	0	3	0
Mut7	4572	1	158	39	0	0	0	1	0	0
Mut8	2750	0	131	17	0	0	0	0	0	0
Mut9	3305	0	173	15	0	0	0	0	0	0
Mut10	11435	8	318	157	0	0	0	0	0	0
Mut11	678	16	36	4	0	0	0	1	0	0
Mut12	347	0	19	0	0	0	0	0	0	0
Mut13	254	0	9	1	0	0	0	0	0	0
Mut14	11605	197	316	161	0	0	0	0	0	0
Mut15	573	6	30	0	0	0	0	1	0	0
Mut16	11419	0	318	157	0	0	0	0	0	0
Mut17	345	0	17	0	0	0	0	0	0	0
Mut18	3550	14	163	23	1	0	0	3	0	0
Mut19	2116	1	92	28	0	1	1	0	1	2
Mut20	11745	11	324	1203	0	1	27	0	1	2

Table 6.4: Type of FSD of each mutant for `grep`. (Note that LowFr, MedFr, HighFr respectively refer to low failure rate, medium failure rate, and high failure rate)

are some FSDs located far away from the selected FSD. Thus, the FCIs of such FSDs still have chance to be selected as the next test cases. However, as explained, aging technique can be used to solve this problem.

For other mutants, all of our ART techniques performed better than RT since they have many dot and small-size FSDs with medium or high failure rates. This characteristics increase the chance of our ART techniques to select the FCIs. Thus, the performance of our ART techniques are better than RT for these mutants.

6.5 Summary and Conclusion

This chapter examines the failure pattern in programs with complex input types. The study is motivated by the findings of previous chapters, showing that for some mutants our ART techniques performed "differently" with for most mutants. We suspected that this corresponded to the failure causing inputs (FCI) distribution of the mutants. We made use the data of FCIs for the twenty mutated versions of `grep` reported in Chapter 3.

In this study, we introduced the concept of *test frame*, *subdomain*, and *failure subdomain*. A test frame is a combination of choices specified for an input. A subdomain is a fraction of the input domain where all inputs sharing a same test frame are located. A subdomain is then referred to as failure subdomain (FSD) if it contains at least an FCI.

We investigated three components of an FSD: its average distance to all other FSDs, its size, and its failure rate. We believe these components affected the behaviour of our ART techniques. From the distance prespective, we classified the FSDs into: compact, medium-spread, and well-spread. From the size prespective, we grouped them into: DOT, small-size, medium-size, and large-size. From the failure rate prespective, we grouped them into: low-frate, medium-frate, and high-frate.

The data showed that for mutant-2 on which our ART techniques performed much worse than RT, all FSDs of this mutant are compact, that is they are very close to each other. Furthermore, their sizes are large and their failure rates are low. For other mutants on which our ART techniques performed worse than RT (however not as worse as mutant-2), they also have some similar FSDs which are in large-size and low failure rate. As discussed, such characteristics of FSD will be unfavourable for our ART techniques, particularly NoAgeMaxMin and RRT without aging. However, aging technique could help to improve the poor performance of those ART techniques.

Another finding of this study is that our ART techniques are favourable in the case where there are a lot of FSDs having dot or small-sizes with medium or high failure rates in the program under test. This has been demonstrated by the good

performance of our ART techniques for testing mutants with such characteristics of FSD.

As the future work, it might be useful to combine the use of non-aging and aging of ART. ART may start without aging approach first. However, after certain number of test cases executed but revealing no failure, the approach is switched to aging.

This chapter presented the first investigation of the FCI pattern in programs with complex input types. However, this study only involves one case study to be investigated. We are interested to continue the study further using more case studies to verify the consistency of the regularities of patterns in programs with complex input types.

Chapter 7

Metamorphic Testing as a Test Oracle for ART on Complex Input Types

This chapter presents Metamorphic Testing (MT) as an alternative test oracle for applying ART. The more complex the program is, the more difficult it can be to find a test oracle. Therefore, an option to use other alternative test oracle would be very beneficial for applying ART on programs with complex input types. In this chapter, we study the potential of using MT as a test oracle for applying ART, particularly for programs with complex input types. In this study, we investigate how effective MT can be used as the test oracle compared to the "common-oracle".

7.1 Introduction

In previous studies of ART, the existence of a test oracle has been assumed, and usually implemented by defining an unmodified program as the test oracle and a defined mutated version as the program under test. If the output of the program using an input from the input domain is different to the output of the test oracle, then the input is identified as a failure causing input (FCI). We describe this kind of test oracle as "common-oracle". However, such an approach is only feasible for evaluation of a testing method. In the reality, we need the use of a "real" test oracle. However, the more complex the program is, the more difficult it can be to find such test oracle. Therefore, the existence of an alternative to such test oracle would be very beneficial for applying ART on programs with complex input types.

Metamorphic Testing (MT) is a common testing method used to alleviate the test oracle problem [9]. MT makes use of some properties of the program under test to define metamorpic relations (MRs). An MR consists of two relations which are

used for two purpose: (1) to generate more test cases, referred to as the *follow-up test cases* from existing test cases, referred to as the *source test cases* and (2) to verify the correctness of the program using the relationship of those test cases and their outputs. More details of MT are described in Section 2.3.

In this chapter, we study the potential of using MT as an alternative test oracle for applying ART particularly on programs with complex input types. We compare the effectiveness of MT as a test oracle compared to "common-oracle". We re-use the category-choice scheme, the test pool, and the twenty mutated versions of `grep` as well as their FCIs as reported in Chapter 3. For this purpose, we test the mutated versions using MT to identify their "FCIs" in the context of MT. Then, we compare these "FCIs" with FCIs identified by "common-oracle" reported in Chapter 3. The comparison results are used to analyse the effectiveness of MT as the test oracle of ART for testing `grep`.

7.2 Metamorphic Relations (MR) for `grep`

Before testing the twenty mutants of `grep` using MT, first of all, we need to develop metamorphic relations (MR) for grep. The development of the MRs are restricted to the test pool of `grep` that is described in detail in Section 3.2.3. Specifically, these MRs are only associated with the regular expression parameter of `grep`'s input. Accordingly, here we present twelve MRs particularly for testing the regular expression parameter of `grep`.

We define some notations that are used in the description of the MRs, as follows:

- RE1: the regular expression parameter of the source test case
- SF1: the input file parameter of the source test case
- O1: the output of the source test case
- RE2: the regular expression parameter of the follow-up test case
- SF2: the input file parameter of the follow-up test case
- O2: the output of the follow-up test case

Details of each MR are as follows:

1. MR1: RE2 is generated from RE1 by changing a range character set in RE1 to its equivalent enumeration character set where its elements are re-arranged randomly. For this MR, RE1 must contain a range character set and SF1 must be equal to SF2. For example: if RE1 is "[1-3]" then one possible RE2 is "[213]". This MR is violated if O2 is not equal to O1.

2. MR2: RE2 is generated from RE1 by enumerating elements included in a range character set of RE1 in a random order and inserting "|" in between. For this MR, RE1 must contain a range character set and SF1 must be equal to SF2. For example: if RE1 is "[1-3]" then one possible RE2 is "2|3|1". This MR is violated if O2 is not equal to O1.

3. MR3: RE2 is generated from RE1 by enclosing each element in RE1 with "[]" and inserting "|" in between. For this MR, RE1 must be a collection of simple characters encompassed by "[]" and SF1 must be equal to SF2. For example: if RE1 is "[abc]" then one possible RE2 is "[a]|[b]|[c]". This MR is violated if O2 is not equal to O1.

4. MR4: RE2 is generated from RE1 by splitting a range character set in RE1 into two smaller interval of ranges and inserting "|" in between. For this MR, RE1 must contain a range character set and SF1 must be equal to SF2. For example: if RE1 is "[1-4]" then one possible RE2 is "[1-2]|[3-4]". This MR is violated if O2 is not equal to O1.

5. MR5: RE2 is generated from RE1 by re-arranging the order of its elements randomly and enclosing RE1 with "[]". For this MR, RE1 must be a collection of simple characters which does not have any meta-character symbols and SF1 must be equal to SF2. For example: if RE1 is "abc" then one possible RE2 is "[bca]". This MR is violated if all lines of O1 are not present in O2

6. MR6: RE2 is generated from RE1 by re-arranging the order of its elements randomly and inserting | in between its elements. For this MR, RE1 must be a collection of simple characters which does not have any meta-character symbols and SF1 must be equal to SF2. For example: if RE1 is "abc" then one possible RE2 is "b|a|c". This MR is violated if all lines of O1 are not present in O2.

7. MR7: RE2 is generated from RE1 by replacing a range character set of RE1 with a smaller interval of the range. For this MR, RE1 must contain a range character set and SF1 must be equal to SF2. For example: if RE1 is "[1-4]" then one possible RE2 is "[1-3]". This MR is violated if all lines of O1 are not present in O2.

8. MR8: RE2 is generated from RE1 by replacing a range character set of RE1 with a larger interval of the range. For this MR, RE1 must contain a range character set and SF1 must be equal to SF2. For example: if RE1 is "[1-4]" then one possible RE2 is "[1-9]". This MR is violated if all lines of O2 are not present in O1. Note: this MR is the inverse of MR7. We do not include the

inverse of other MRs, except MR7, in this chapter due to the restricted source test cases available in the test pool.

9. MR9: RE2 is generated from RE1 by adding | at the end of RE1 and followed by one of these meta-character symbols: [[:digit:]], [^[:digit:]], \w, and \W. For this MR, SF2 is generated from SF1 by adding a new line that is matched by the added meta-character. For example: if RE1 is "abc" then one possible RE2 is "abc|\w". \w is a meta-character matching any word characters in a line. To form SF2, "Hello world" is one possible example of line that can be added after the last line of SF1. This MR is violated if either all lines of O1 or the new line added are/is not present in O2.

10. MR10: RE2 is generated from RE1 by adding a repetition meta-character, either + or {1}, at the end of RE1. For example: if RE1 is "abc" then one possible RE2 is "abc{1}". For this MR, SF2 must be equal to SF1. This MR is violated if O2 is not equal to O1.

11. MR11: RE2 is generated from RE1 by replacing a certain meta-character symbol of RE1 with its negation. For this MR, RE1 must contain one element of these pairs of meta-characters: \w and its negation \W, \s and its negation \S, [[:digit:]] and its negation [^[:digit:]], or [[:alphanum:]] and its negation [^[:alphanum:]]. In this case, SF1 and SF2 must be equal and consist of two lines. The lines vary depending on the meta-character present in RE1. One of the lines must be matched by the meta-character and the other one by its negation. For example: if RE1 is "^\w$" then RE2 must be "^\W$". In this case, both SF1 and SF2 must consist of a line matched by "^\w" and a line matched by "^\W". This MR is violated if either O1 is equal to O2 or if all lines of O1 combined with all lines of O2 are not equal to SF1 (or SF2).

12. MR12: RE2 is generated from RE1 by replacing one or more elements of RE1 with ".". For example: if RE1 is "abcd" then one possible RE2 is "a..d". For this MR, SF2 must be equal to SF1. This MR is violated if all lines of O1 are not present in O2.

7.3 Experimental Work

7.3.1 Experimental Design

As mentioned, in this study we choose `grep` as the experimental subject as it has been used in the previous chapters studying about the application of ART on programs with complex input types. In those studies, the "common-oracle" has been

Metamorphic Relation	Number of Eligible Test Cases
MR1	33607
MR2	13774
MR3	2084
MR4	17327
MR5	1452
MR6	1452
MR7	33607
MR8	33607
MR9	159853
MR10	60017
MR11	29026
MR12	6821

Table 7.1: Number of Eligible Test cases for Each MR in this study

used as the test oracle in evaluating ART. We make use of the data of FCI of the twenty mutated versions of `grep` here to be compared with the experiment results of this study. Details of the twenty mutated versions of `grep` are described in Section 3.2.5.

For this purpose, first of all, we test the twenty versions of `grep` using the defined MRs presented in Section 7.2. The source test cases for all MRs are selected randomly from the test pool of `grep`. Details of the test pool is described in Section 3.2.3. The test pool consists of 171,634 test cases. However, due to different characteristic of each MR, not all test cases in the test pool are eligible to be the source test cases of the MRs. Table 7.1 shows the number of test cases in the test pool that are eligible for each MR.

For each MR, we use all eligible test cases as the source test cases to test the twenty mutated versions. Next, we identify pairs of test cases that violate the MRs. The source test cases of the pairs are then identified as FCI of the MR under testing, or referred to as FCI-MT. Next, for each mutant, we compare the FCI-MT with FCI that have been identified by the "common-oracle" of ART. The latter group of FCI is referred to as FCI-ART. Using the comparison results, for each mutant and each MR, we present two questions that help us to evaluate the effectiveness of MT compared to the "common-oracle" of ART to be used as as the test oracle: (1) how many common elements do FCI-MT and FCI-ART have ? and (2) how many elements that only belong to FCI-MT ?. The answer of the first question describes the capability of MT to replace "common-oracle" of ART whereas the second one reflects the capability of MT to supplement "common-oracle" of ART.

M-ID	MR1	MR2	MR3	MR4	MR5	MR6	MR7	MR8	MR9	MR10	MR11	MR12
1	0	1163	0	412	0	0	0	38	2059	0	0	0
2	1	0	152	0	0	0	0	0	2	93	0	0
3	0	0	1151	0	0	0	0	31	77492	0	0	0
4	9	20	79	0	79	79	0	29	4750	222	0	0
5	1	1163	0	412	0	860	0	24	145462	93	0	0
6	0	53	0	0	0	0	0	10	0	21	0	0
7	88	14	6	0	6	6	0	19	16495	1987	0	0
8	0	1	1173	315	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	7	16480	729	0	0
10	0	1163	0	412	0	0	0	24	380	0	0	0
11	2	0	120	0	0	0	0	0	4628	2	0	0
12	7	0	0	0	0	0	0	0	1802	4	0	0
13	1	0	0	0	0	0	0	0	0	7	0	0
14	0	905	0	425	0	860	0	27	158828	2	41	0
15	3	0	15	0	0	0	0	107	135	7	0	2
16	16	136	0	412	0	0	27	12	278	137	0	0
17	1	0	0	0	0	0	0	0	1	0	0	0
18	2	1	0	0	0	0	0	6	526	0	0	0
19	0	0	0	0	0	0	0	4	35	1	0	0
20	1	0	0	0	0	0	0	3	0	0	0	0

Table 7.2: Number of tests revealing failures by MRs

7.3.2 Experiment Results

Table 7.2 reports the number of source test cases of all mutants that violate the twelve MRs. As the number of eligible test cases for each MR is different, Table 7.3 reports the percentage of the eligible test cases that violate the MRs. These tables show that at least one MR could reveal the failures in all mutated versions. Among all MRs, MR9 reveals failures in most mutated versions whereas MR5, MR7, MR11, and MR12 only reveal failures in one of the mutated versions. Table 7.4 to Table 7.15 report the comparison of FCI-ART and FCI-MT for all mutants from MR1 to MR12 respectively. In this tables, the second column contains the number of FCI-ART, the third column contains the number of FCI-MT, the fourth column contains the number of common elements of FCI-ART and FCI-MT, and the last column contains the number of elements only belong to FCI-MT for all mutants. From these tables, we identify that for most mutated versions, the number of common elements of FCI-ART and FCI-MT is significantly large for MR3, MR8, MR9, and MR11. The number of FCI that only belongs to FCI-MT is significantly larger for MR2, MR3, MR4, MR6, MR8, MR9, and MR10.

M-ID	MR1	MR2	MR3	MR4	MR5	MR6	MR7	MR8	MR9	MR10	MR11	MR12
1	0.00%	8.44%	0.00%	2.38%	0.00%	0.00%	0.00%	0.11%	1.29%	0.00%	0.00%	0.00%
2	0.00%	0.00%	7.29%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.15%	0.00%	0.00%
3	0.00%	0.00%	55.23%	0.00%	0.00%	0.00%	0.00%	0.09%	48.48%	0.00%	0.00%	0.00%
4	0.03%	0.15%	3.79%	0.00%	5.44%	5.44%	0.00%	0.09%	2.97%	0.37%	0.00%	0.00%
5	0.00%	8.44%	0.00%	2.38%	0.00%	59.23%	0.00%	0.07%	91.00%	0.15%	0.00%	0.00%
6	0.00%	0.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.03%	0.00%	0.03%	0.00%	0.00%
7	0.26%	0.10%	0.29%	0.00%	0.00%	0.41%	0.00%	0.06%	10.32%	3.31%	0.00%	0.00%
8	0.00%	0.01%	56.29%	1.82%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
9	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%	10.31%	1.21%	0.00%	0.00%
10	0.00%	8.44%	0.00%	2.38%	0.00%	0.00%	0.00%	0.07%	0.24%	0.00%	0.00%	0.00%
11	0.01%	0.00%	5.76%	0.00%	0.00%	0.00%	0.00%	0.00%	2.90%	0.00%	0.00%	0.00%
12	0.02%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	1.13%	0.01%	0.00%	0.00%
13	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%
14	0.00%	6.57%	0.00%	2.45%	0.00%	59.23%	0.00%	0.08%	99.36%	0.00%	0.14%	0.00%
15	0.01%	0.00%	0.72%	0.00%	0.00%	0.00%	0.00%	0.32%	0.08%	0.01%	0.00%	0.03%
16	0.05%	0.99%	0.00%	2.38%	0.00%	0.00%	0.08%	0.04%	0.17%	0.23%	0.00%	0.00%
17	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
18	0.01%	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%	0.33%	0.00%	0.00%	0.00%
19	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.02%	0.00%	0.00%	0.00%
20	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%	0.00%	0.00%

Table 7.3: Percentage of tests revealing failures by MRs

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	3648	0	-	3648	-
2	0	1	-	-	1
3	3109	0	-	3109	-
4	604	9	9	595	0
5	1433	1	1	1432	0
6	635	0	-	635	-
7	1483	88	49	1434	39
8	919	0	-	919	-
9	1058	0	-	1058	-
10	3652	0	-	3652	-
11	218	2	0	218	2
12	111	7	1	110	6
13	79	1	0	79	1
14	3792	0	-	3792	-
15	166	3	1	165	2
16	3650	16	3	3647	13
17	112	1	0	112	1
18	1246	2	0	1246	2
19	1226	0	-	1226	-
20	4840	1	1	4839	0

Table 7.4: Comparison of FCI-ART and FCI-MT for MR1

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	1163	-	-	1163
2	0	0	-	-	-
3	0	0	-	-	-
4	286	20	20	266	0
5	0	1163	-	-	1163
6	355	53	30	325	23
7	0	14	-	-	14
8	1	1	1	0	0
9	0	0	-	-	-
10	0	1163	-	-	1163
11	2	0	-	2	-
12	0	0	-	-	-
13	0	0	-	-	-
14	61	905	12	49	893
15	0	0	-	-	-
16	1	136	0	1	136
17	0	0	-	-	-
18	87	1	0	87	1
19	531	0	-	531	-
20	1163	0	-	1163	-

Table 7.5: Comparison of FCI-ART and FCI-MT for MR2

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	0	-	-	-
2	152	152	152	0	0
3	1167	1151	1151	16	0
4	153	79	77	76	2
5	0	0	-	-	-
6	544	0	-	544	-
7	301	6	6	295	0
8	0	1173	-	-	1173
9	0	0	-	-	-
10	0	0	-	-	-
11	91	120	90	1	30
12	0	0	-	-	-
13	0	0	-	-	-
14	0	0	-	-	-
15	15	15	15	0	0
16	0	0	-	-	-
17	0	0	-	-	-
18	491	0	-	491	-
19	552	0	-	552	-
20	1205	0	-	1205	-

Table 7.6: Comparison of FCI-ART and FCI-MT for MR3

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	412	-	-	412
2	0	0	-	-	-
3	0	0	-	-	-
4	78	0	-	78	-
5	0	412	-	-	412
6	0	0	-	-	-
7	0	0	-	-	-
8	0	315	-	-	315
9	0	0	-	-	-
10	0	412	-	-	412
11	0	0	-	-	-
12	0	0	-	-	-
13	0	0	-	-	-
14	83	425	13	70	412
15	0	0	-	-	-
16	0	412	-	-	412
17	0	0	-	-	-
18	0	0	-	-	-
19	0	0	-	-	-
20	497	0	-	497	-

Table 7.7: Comparison of FCI-ART and FCI-MT for MR4

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	0	-	-	-
2	130	0	-	130	-
3	822	0	-	822	-
4	139	79	77	62	2
5	0	0	-	-	-
6	544	0	-	544	-
7	301	6	6	295	0
8	0	0	-	-	-
9	0	0	-	-	-
10	0	0	-	-	-
11	61	0	-	61	-
12	0	0	-	-	-
13	0	0	-	-	-
14	0	0	-	-	-
15	9	0	-	9	-
16	0	0	-	-	-
17	0	0	-	-	-
18	491	0	-	491	-
19	552	0	-	552	-
20	860	0	-	860	-

Table 7.8: Comparison of FCI-ART and FCI-MT for MR5

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	0	-	-	-
2	130	0	-	130	-
3	822	0	-	822	-
4	139	79	77	62	2
5	0	860	-	-	860
6	544	0	-	544	-
7	301	6	6	295	0
8	0	0	-	-	-
9	0	0	-	-	-
10	0	0	-	-	-
11	61	0	-	61	-
12	0	0	-	-	-
13	0	0	-	-	-
14	0	860	-	-	860
15	9	0	-	9	-
16	0	0	-	-	-
17	0	0	-	-	-
18	491	0	-	491	-
19	552	0	-	552	-
20	860	0	-	860	-

Table 7.9: Comparison of FCI-ART and FCI-MT for MR6

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	3648	0	-	3648	-
2	0	0	-	-	-
3	3109	0	-	3109	-
4	604	0	-	604	-
5	1433	0	-	1433	-
6	635	0	-	635	-
7	1483	0	-	1483	-
8	919	0	-	919	-
9	1058	0	-	1058	-
10	3652	0	-	3652	-
11	218	0	-	218	-
12	111	0	-	111	-
13	79	0	-	79	-
14	3792	0	-	3792	-
15	166	0	-	166	-
16	3650	27	3	3647	24
17	112	0	-	112	-
18	1246	0	-	1246	-
19	1226	0	-	1226	-
20	4840	0	-	4840	-

Table 7.10: Comparison of FCI-ART and FCI-MT for MR7

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	3648	38	38	3610	0
2	0	0	-	-	-
3	3109	31	28	3081	3
4	604	29	2	602	27
5	1433	24	7	1426	17
6	635	10	1	634	9
7	1483	19	15	1468	4
8	919	0	-	919	-
9	1058	7	7	1051	0
10	3652	24	24	3628	0
11	218	0	-	218	-
12	111	0	-	111	-
13	79	0	-	79	-
14	3792	27	26	3766	1
15	166	107	4	162	103
16	3650	12	1	3649	11
17	112	0	-	112	-
18	1246	6	6	1240	0
19	1226	4	4	1222	0
20	4840	3	2	4838	1

Table 7.11: Comparison of FCI-ART and FCI-MT for MR8

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	7959	2059	170	7789	1889
2	194	2	0	194	2
3	8088	77492	7050	1038	70442
4	3411	4750	2145	1266	2605
5	4333	1E+05	3888	445	141574
6	4696	0	-	4696	-
7	3289	16495	2745	544	13750
8	2609	0	-	2609	-
9	2258	16480	1605	653	14875
10	7961	380	50	7911	330
11	609	4628	380	229	4248
12	294	1802	260	34	1542
13	239	0	-	239	-
14	8235	2E+05	7924	311	150904
15	565	135	2	563	133
16	7918	278	13	7905	265
17	334	1	0	334	1
18	4676	526	0	4676	526
19	7010	35	18	6992	17
20	71844	0	-	71844	-

Table 7.12: Comparison of FCI-ART and FCI-MT for MR9

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	14	0	-	14	-
2	154	93	91	63	2
3	799	0	-	799	-
4	2234	222	38	2196	184
5	0	93	-	-	93
6	3899	21	3	3896	18
7	290	1987	6	284	1981
8	0	1	-	-	1
9	0	729	-	-	729
10	14	0	-	14	-
11	60	2	0	60	2
12	0	4	-	-	4
13	0	7	-	-	7
14	129	2	0	129	2
15	9	7	0	9	7
16	0	137	-	-	137
17	2	0	-	2	-
18	1121	0	-	1121	-
19	5365	1	0	5365	1
20	32281	0	-	32281	-

Table 7.13: Comparison of FCI-ART and FCI-MT for MR10

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	1	0	-	1	-
2	0	0	-	-	-
3	0	0	-	-	-
4	112	0	-	112	-
5	0	0	-	-	-
6	0	0	-	-	-
7	1	0	-	1	-
8	2	0	-	2	-
9	0	0	-	-	-
10	1	0	-	1	-
11	4	0	-	4	-
12	0	0	-	-	-
13	0	0	-	-	-
14	83	41	41	42	0
15	0	0	-	-	-
16	1	0	-	1	-
17	0	0	-	-	-
18	1	0	-	1	-
19	0	0	-	-	-
20	24	0	-	24	-

Table 7.14: Comparison of FCI-ART and FCI-MT for MR11

M-ID	FCI-ART	FCI-MT	comm	FCI-ARTonly	FCI-MTonly
1	0	0	-	-	-
2	39	0	-	39	-
3	443	0	-	443	-
4	40	0	-	40	-
5	0	0	-	-	-
6	16	0	-	16	-
7	9	0	-	9	-
8	0	0	-	-	-
9	0	0	-	-	-
10	0	0	-	-	-
11	54	0	-	54	-
12	0	0	-	-	-
13	0	0	-	-	-
14	0	0	-	-	-
15	6	2	0	6	2
16	0	0	-	-	-
17	0	0	-	-	-
18	14	0	-	14	-
19	16	0	-	16	-
20	443	0	-	443	-

Table 7.15: Comparison of FCI-ART and FCI-MT for MR12

7.4 Summary and Discussions

In this study, we investigate the potential of using MT as an alternative test oracle for applying ART on programs with complex input types. We re-used `grep` as experimental study and details of the experimental setup were basically the same as those reported in Chapter 3. We examined the effectiveness of MT by comparing FCI-MT (FCIs reported by MT) with FCI-ART (FCIs reported by "common-oracle", as reported in Chapter 3). For this purpose, we test `grep` using MT by defining twelve MRs. These properties of the MRs were only related to the regular expression parameter of `grep` as restriced in the study of `grep` in Chapter 3. We implemented the twelve MRs towards the twenty mutated versions of `grep`. For each MR, we tested all mutated versions using all eligible test cases in the test pool as the source test cases. Next, we identified the FCI-MT of the MRs for all mutated versions. Then, we compared them with FCI-ART.

As reported in Table 7.2 and Table 7.3, the results showed that at least one MR could reveal the failures in all mutated versions. Among all MRs, MR9 revealed failures in most mutated versions whereas MR5, MR7, MR11, and MR12 only revealed failures in one of the mutated versions. Table 7.4 to Table 7.15 reported the comparison of FCI-ART and FCI-MT for all mutants from MR1 to MR12 respectively. These tables showed that for most mutated versions, the number of common

elements of FCI-ART and FCI-MT is significantly large for MR3, MR8, MR9, and MR11. This shows that MT is effective to be an alternative test oracle for evaluating ART particularly in testing programs with complex input types, like `grep`. Even though all FCIs of FCI-ART are not "covered" by FCI-MT, if there are no other test oracles available, this alternative would be very beneficial. The tables also reported that the number of FCI that only belongs to FCI-MT is significantly large for MR2, MR3, MR4, MR6, MR8, MR9, and MR10. This shows that MT is not only useful to be used an alternative test oracle, but also to be the supplement to the existing test oracle of the program under test. By generating the next test cases from the source test cases, MT doubles the number of the executed test cases hence more FCI can be identified. In other words, more failures can be revealed. From this study, we also identified that each MR has different number of FCI. This shows that each MR has different "quality". The larger number FCIs revealed by an MR reflects the better "quality" of the MR. In other words, the effectiveness of MT as the test oracle also depends on the quality of the set of MRs defined.

From the prespective of a study focussing on metamorphic testing, this chapter can be extended to investigate the effectiveness of a set of MRs defined for a program under test. The cumulative elements of FCI-ART covered by the cumulative elements of FCI-MT can be used to measure the comprehensiveness of the defined MRs. The larger number of common elements reflects the more comprehensive set of MRs. Not many of previous studies have investigated the comprehensiveness of a set of MRs. Thus, this study is worth to be conducted in the future study of MT.

Chapter 8

The Application of ART on Metamorphic Testing

In this chapter, we demonstrate that adaptive random testing (ART) strategies can be used in combination with Metamorphic Testing (MT). We apply ART in the source test case selection of the implementation of the metamorphic relations (MRs) defined for the program under test. In previous studies of MT, the test cases were merely selected randomly from the input domain. In this study, we are interested to identify the improvement of MT after ART strategies applied in the source test case selection.

8.1 Introduction

ART methods have been a subject of considerable research interest recently. Liu and Zhu [21] conducted some experimental work to evaluate of the reliability of the existing ART methods, Tappenden and Miller [22] introduced an ART-based search algorithm, called evolutionary Adaptive Random Testing (eART), Tse *et al.* [23] extended the use of ART in the regression testing area. Motivated by these studies, in this thesis we also like to investigate the use of ART techniques in other area of testing, in particular in the application of Metamorphic Testing (MT).

MT is a property-based testing method because it makes use of some properties of the program under test to define metamorpic relations (MRs). MRs are somehow used to generate more test cases, referred to as *follow-up test cases* from the available test cases, which are referred to as *source test cases*. The relationship between these test cases are then used to verify the correctness of the program under test. A detailed description of MT is presented in Section 2.3.

In previous studies of MT, the source test cases were selected randomly from the input domain. In this chapter, we apply ART strategies in the source test

case selection of the implementation of the MRs defined for the program under test. In the original study of MT, the number of source test cases to be selected is normally predefined. However, here the testing stops whenever the implemented MR is violated. This follows the basic concept of ART which stops the testing after the first failure revealed. ART attempts to select test cases evenly spread until the first failure revealed. However, the previous studies of ART have not investigated the effects of continuing the testing after the first failure revealed. Thus, in this study we follow the similar procedure. The number of source test case selected would be different for each MR, and depends on when each MRs is violated by the pair of source and follow-up test case selected and executed. Then, we investigate the improvement of MT when ART is used, instead of RT. The effectiveness is measured by comparing the number of pairs of source and follow-up test cases required by ART and that is required by RT to violate the implemented MR at the first time. This measurement is similar to the *F-measure* which has been commonly used as the effectiveness metric in ART studies.

8.2 Experimental Work

8.2.1 Experimental Design

In this study, we again use `grep` which has been used in the previous chapters. The same experimental settings are re-used, in particular, we make use of the same category-choice scheme, the mutated versions, and the test pool which have been described in details in Chapter 3. We also make use of the subset of metamorphic relations (MRs) as described in Chapter 7. However, we do not use all MRs, as some MRs have a very small number of eligible source test cases to be used, as shown in Table 7.1. We therefore only use MR1, MR2, MR7, MR8, MR9, and MR10. We believe that it is meaningless to compare the difference of the impact of ART and RT when the size of the input space is very small.

As mentioned, this study aims to investigate the improvement of MT by applying ART in the source test case selection of the defined MRs. We measure the improvement of MT by comparing the numbers of pairs of source and follow-up test cases required by ART and RT to violate the implemented MR for the first time. This measurement is intuitively similar to the *F-measure* which has been commonly used as the metric in ART studies.

In this experiment, we only apply some of ART techniques that have been introduced in this thesis. We choose AgeMaxMin and AgeMaxSum among all other FSCS-ART techniques introduced in Chapter 3 to be used in this study. We also use RRT with aging that has been presented in Chapter 5. In this study, we set

the InitLim to 7 and MaxTrial to 10. We choose these techniques as they are most efficient in their groups but their performance are still significantly better than RT.

8.2.2 Results

Table 8.1 shows the number of pairs of source test cases and follow-up test cases required to reveal the violation of the MRs for the first time using the original MT approach where the source test cases are selected randomly. Table 8.2, Table 8.3, and Table 8.4 show the data using AgeMaxMin, AgeMaxSum, and RRT with aging respectively in the source test case selection of the implementation of the used MRs. In these tables, cases with data "0.00" refers to cases when there are no pairs of test cases violating the relevant MR after conducting exhaustive testing across all eligible test cases in the test pool of `grep`. The F-ratio of those data is presented in Table 8.5, Table 8.6, and Table 8.7 respectively. In those tables, cases with NA refers to cases "0.00" in the previous set of tables, which show that no pairs of test cases violating the relevant MR after conducting exhaustive testing across all eligible test cases.

From Table 8.5, we identify that the F-ratio is smaller than 1 in most cases. It shows that AgeMaxMin outperfoms RT for most cases. Data in Table 8.6 report that AgeMageSum performs similarly to RT for most cases, and RT outperforms AgeMaxSum for several cases. Table 8.7 shows that RRT outperforms RT for most cases.

8.2.3 Data Analysis

To summarize the analysis of data presented in Table 8.5, Table 8.6, and Table 8.7, we conduct the analysis of variance (ANOVA) of those data. Table 8.8 presents the results of the ANOVA for the F-ratio of RT and the four ART strategies. The treatments are in the first column of the table and the sum of squares, degrees of freedom, and mean of squares for each treatment are in the following columns. The F value defines the ratio between the treatment and the error effect (last row). The larger F shows the greater probability to reject that the techniques' F-ratio are equal. The last column contains the p-value that represents "the probability of having a value of the test statistic that is equal to or more extreme than the one we observed" [53]. We set our level of significance to be 0.05. Thus, we accept the null hypotheses if the p-value is greater than 0.05. Otherwise, we reject the hypotheses. The last column of Table 8.8 shows that the p-value is < 0.0001 which is less than the level of significance. Therefore, we reject the null hypothesis and conclude that the techniques' F-ratios are different. Details of how different they are and their ranks in term of their F-ratio are calculated using Bonferroni analysis. The results

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	0.00	10.70	0.00	590.05	21.40	0.00
Mutant2	16510.48	0.00	0.00	0.00	40203.84	353.94
Mutant3	0.00	0.00	0.00	738.58	1.06	0.00
Mutant4	3944.15	655.49	0.00	729.31	33.54	133.09
Mutant5	14269.42	10.67	0.00	1020.58	0.09	297.98
Mutant6	0.00	254.01	0.00	2740.58	0.00	2248.12
Mutant7	387.25	1231.15	0.00	1172.28	8.67	17.51
Mutant8	0.00	6959.66	0.00	0.00	0.00	28590.97
Mutant9	0.00	0.00	0.00	3149.56	8.51	81.20
Mutant10	0.00	11.21	0.00	1030.35	422.39	0.00
Mutant11	11194.54	0.00	0.00	16465.10	33.83	19072.26
Mutant12	4424.22	0.00	0.00	16388.78	84.32	12449.02
Mutant13	15883.65	0.00	0.00	0.00	0.00	7722.82
Mutant14	0.00	14.34	0.00	1198.99	0.01	19723.89
Mutant15	9307.25	0.00	0.00	304.08	1753.90	11262.75
Mutant16	2005.17	100.16	1026.75	2861.31	591.49	439.65
Mutant17	17070.88	0.00	0.00	0.00	50356.23	0.00
Mutant18	10919.06	7281.94	0.00	4534.58	315.55	0.00
Mutant19	0.00	0.00	0.00	6990.27	4616.14	28235.37
Mutant20	18077.62	0.00	0.00	8498.71	0.00	0.00

Table 8.1: Number of pairs of source and follow-up test cases required to violate the MRs for the first time using RT to select the source test case

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	0.00	4.18	0.00	505.33	57.94	0.00
Mutant2	9832.65	0.00	0.00	0.00	23003.56	698.60
Mutant3	0.00	0.00	0.00	590.61	0.78	0.00
Mutant4	1926.53	802.03	0.00	1020.90	18.14	312.08
Mutant5	10030.57	4.21	0.00	747.77	0.08	721.03
Mutant6	0.00	163.22	0.00	1830.05	0.00	2954.45
Mutant7	212.34	1168.37	0.00	962.80	3.23	18.10
Mutant8	0.00	3133.42	0.00	0.00	0.00	26801.40
Mutant9	0.00	0.00	0.00	2415.17	3.43	45.06
Mutant10	0.00	4.17	0.00	732.08	216.42	0.00
Mutant11	6983.71	0.00	0.00	9965.82	25.72	19940.27
Mutant12	2608.21	0.00	0.00	9498.72	31.42	11548.28
Mutant13	10217.24	0.00	0.00	0.00	0.00	7547.51
Mutant14	0.00	7.92	0.00	706.24	0.00	15361.05
Mutant15	5220.65	0.00	0.00	180.27	439.49	8719.76
Mutant16	2226.52	105.36	1313.01	1677.54	502.31	401.74
Mutant17	10238.23	0.00	0.00	0.00	31494.78	0.00
Mutant18	7243.97	3065.24	0.00	2878.14	119.29	0.00
Mutant19	0.00	0.00	0.00	3617.89	2340.18	25747.17
Mutant20	10347.01	0.00	0.00	5069.50	0.00	0.00

Table 8.2: Number of pairs of source and follow-up test cases required to violate the MRs for the first time using AgeMaxMin

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	0.00	11.54	0.00	883.77	75.21	0.00
Mutant2	17937.78	0.00	0.00	0.00	46481.99	697.59
Mutant3	0.00	0.00	0.00	1142.41	1.07	0.00
Mutant4	4265.74	1021.74	0.00	1141.53	32.71	290.17
Mutant5	15787.62	11.43	0.00	1369.56	0.08	681.34
Mutant6	0.00	325.34	0.00	3260.72	0.00	2994.53
Mutant7	379.23	1124.30	0.00	1521.84	8.47	29.48
Mutant8	0.00	6378.70	0.00	0.00	0.00	28620.32
Mutant9	0.00	0.00	0.00	4295.13	8.52	82.24
Mutant10	0.00	11.56	0.00	1355.35	401.43	0.00
Mutant11	12615.04	0.00	0.00	16649.99	33.11	18965.35
Mutant12	4088.16	0.00	0.00	16762.06	87.15	11013.53
Mutant13	18244.92	0.00	0.00	0.00	0.00	12948.58
Mutant14	0.00	15.79	0.00	1459.72	0.00	16741.35
Mutant15	8716.32	0.00	0.00	313.94	1259.26	7279.18
Mutant16	2073.77	141.55	1309.41	2518.70	596.02	435.22
Mutant17	16526.63	0.00	0.00	0.00	45752.90	0.00
Mutant18	11924.34	7099.20	0.00	5248.37	293.88	0.00
Mutant19	0.00	0.00	0.00	7608.97	4024.92	26676.18
Mutant20	13908.29	0.00	0.00	7327.25	0.00	0.00

Table 8.3: Number of pairs of source and follow-up test cases required to violate the MRs for the first time using AgeMaxSum

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	0	11.59	0	602.44	55.62	0
Mutant2	9001.23	0	0	0	24034.45	655.29
Mutant3	0	0	0	816.97	0.75	0
Mutant4	1567.94	751.35	0	772.13	18.43	309.35
Mutant5	12000.78	11.83	0	842.83	0.08	627.00
Mutant6	0	155.24	0	1348.20	0	2800.44
Mutant7	210.33	1170.33	0	1170.33	2.93	18.95
Mutant8	0	3202.33	0	0	0	2070.00
Mutant9	0	0	0	2010.50	3.33	45.90
Mutant10	0	11.84	0	670.77	201.85	0
Mutant11	7100.98	0	0	9800.39	24.57	19200.23
Mutant12	1568.33	0	0	1002.45	31.91	10500.35
Mutant13	11001.99	0	0	0	0	7812.33
Mutant14	0	15.89	0	741.09	0.01	15800.11
Mutant15	4501.66	0	0	172.62	422.94	8900.12
Mutant16	2100.44	116.45	1151.43	1700.22	410.45	586.15
Mutant17	10004.31	0	0	0	34000.20	0
Mutant18	7300.23	2800.88	0	2399.12	117.45	0
Mutant19	0	0	0	3500.12	2587.12	2801.23
Mutant20	9001.89	0	0	6000.23	0	0

Table 8.4: Number of pairs of source and follow-up test cases required to violate the MRs for the first time using RRT with aging

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	NA	39.08%	NA	85.64%	270.74%	NA .00
Mutant2	59.55%	NA	NA	NA	57.22%	197.38%
Mutant3	NA	NA	NA	79.97%	73.14%	NA
Mutant4	48.85%	122.36%	NA	139.98%	54.08%	234.48%
Mutant5	70.29%	39.44%	NA	73.27%	84.78%	241.97%
Mutant6	NA	64.26%	NA	66.78%	NA	131.42%
Mutant7	54.83%	94.90%	NA	82.13%	37.22%	103.35%
Mutant8	NA	45.02%	NA	NA	NA	93.74%
Mutant9	NA	NA	NA	76.68%	40.24%	55.50%
Mutant10	NA	37.18%	NA	71.05%	51.24%	NA
Mutant11	62.38%	NA	NA	60.53%	76.03%	104.55%
Mutant12	58.95%	NA	NA	57.96%	37.26%	92.76%
Mutant13	64.33%	NA	NA	NA	NA	97.73%
Mutant14	NA	55.26%	NA	58.90%	28.57%	77.88%
Mutant15	56.09%	NA	NA	59.28%	25.06%	77.42%
Mutant16	111.04%	105.19%	127.88%	58.63%	84.92%	91.38%
Mutant17	59.97%	NA	NA	NA	62.54%	NA
Mutant18	66.34%	42.09%	NA	63.47%	37.80%	NA
Mutant19	NA	NA	NA	51.76%	50.70%	91.19%
Mutant20	57.24%	NA	NA	59.65%	NA	NA

Table 8.5: F-ratio of AgeMaxMin to violate the MRs for the first time

	MR1	MR2	MR7	MR8	MR9	MR10
Mutant1	NA	107.93%	NA	149.78%	351.45%	NA
Mutant2	108.64%	NA	NA	NA	115.62%	197.09%
Mutant3	NA	NA	NA	154.68%	101.04%	NA
Mutant4	108.15%	155.88%	NA	156.52%	97.53%	218.02%
Mutant5	110.64%	107.19%	NA	134.19%	91.30%	228.65%
Mutant6	NA	128.08%	NA	118.98%	NA	133.20%
Mutant7	97.93%	91.32%	NA	129.82%	97.68%	168.34%
Mutant8	NA	91.65%	NA	NA	NA	100.10%
Mutant9	NA	NA	NA	136.37%	100.09%	101.29%
Mutant10	NA	103.12%	NA	131.54%	95.04%	NA
Mutant11	112.69%	NA	NA	101.12%	97.86%	99.44%
Mutant12	92.40%	NA	NA	102.28%	103.36%	88.47%
Mutant13	114.87%	NA	NA	NA	NA	167.67%
Mutant14	NA	110.16%	NA	121.75%	28.57%	84.88%
Mutant15	93.65%	NA	NA	103.24%	71.80%	64.63%
Mutant16	103.42%	141.32%	127.53%	88.03%	100.77%	98.99%
Mutant17	96.81%	NA	NA	NA	90.86%	NA
Mutant18	109.21%	97.49%	NA	115.74%	93.13%	NA
Mutant19	NA	NA	NA	108.85%	87.19%	94.48%
Mutant20	76.94%	NA	NA	86.22%	NA .00	NA

Table 8.6: F-ratio of AgeMaxSum to violate the MRs for the first time

	MR1	MR2	MR7	MR8	MR9	MR10
mutant1	NA	108.40%	NA	102.10%	259.90%	NA
mutant2	54.52%	NA	NA	NA	59.78%	185.14%
mutant3	NA	NA	NA	110.61%	70.41%	NA
mutant4	39.75%	114.63%	NA	105.87%	54.96%	232.43%
mutant5	84.10%	110.91%	NA	82.58%	81.52%	210.42%
mutant6	NA	61.11%	NA	49.19%	NA	124.57%
mutant7	54.31%	95.06%	NA	99.83%	33.83%	108.20%
mutant8	NA	46.01%	NA	NA	NA	7.24%
mutant9	NA	NA	NA	63.83%	39.10%	56.54%
mutant10	NA	105.58%	NA	65.10%	47.79%	NA
mutant11	63.43%	NA	NA	59.52%	72.63%	100.67%
mutant12	35.45%	NA	NA	6.12%	37.84%	84.35%
mutant13	69.27%	NA	NA	NA	NA	101.16%
mutant14	NA	110.82%	NA	61.81%	35.71%	80.11%
mutant15	48.37%	NA	NA	56.77%	24.11%	79.02%
mutant16	104.75%	116.27%	112.14%	59.42%	69.39%	133.32%
mutant17	58.60%	NA	NA	NA	67.52%	NA
mutant18	66.86%	38.46%	NA	52.91%	37.22%	NA
mutant19	NA	NA	NA	50.07%	56.05%	9.92%
mutant20	49.80%	NA	NA	70.60%	NA	NA

Table 8.7: F-ratio of RRT with aging to violate the MRs for the first time

	SS	Degrees of freedom	MS	F	p
Strategies	3	7.036	2.345	14.957	<0.0001
Error	274	42.963	0.157		
Corrected Total	277	49.999			

Table 8.8: ANOVA of RT, AgeMaxMin, AgeMaxSum, and RRT with aging

are presented in Table 8.9. By default, Bonferroni analysis assumes that a larger F-ratio has a higher ranking. As discussed in previous chapters, for the F-ratio, a smaller value reflects a better failure detection capability of the strategy compared to RT. Therefore, to interpret the rank correctly for our purpose, we reverse the order of the ranking. Data in Table 8.9 show that AgemaxMin has the highest ranking, followed by RRT. Both of these strategies are grouped in level B which is the level of most effective techniques. Then, RT and AgeMaxSum are grouped in the lowest level, level A.

8.2.4 Summary and Discussion

In previous studies of MT, the source test cases were merely selected randomly from the input domain. This study investigates the improvement of applying Adaptive Random Testing (ART) methods as the source test case selection strategies in applying Metamorphic Testing (MT). We aim to identify whether the effectiveness of

Category	Average of F-ratios	Groups
AgeMaxSum	1.159	A
RT	1.000	A
RRT with aging	0.783	B
AgeMaxMin	0.779	B

Table 8.9: The rank of RTRT, AgeMaxMin, AgeMaxSum, and RRT with aging using the Bonferroni analysis

MT can be improved after applying ART in the source test case selection. In the original study of MT, the number of source test cases to be selected is normally predefined. However, in this study, it is not. The testing stops whenever the implemented MR is violated, to follow the whole idea of ART. In this scenario, the number of source test case selected would be different for each MR. In this case, the number of test cases executed before the first violation of an MR is used to evaluate the improvement of MT after applying ART for the source test case selection of the defined metamorphic relations (MRs). This measurement is intuitively similar to the *F-measure* which has been commonly used as the metric in ART studies.

In this study, we follow similar experimental settings of `grep`, as reported in Chapter 3 and Chapter 7. For applying MT, we use a subset of the MRs that have been introduced in Chapter 7: MR1, MR2, MR7, MR8, MR9, and MR10 as others have very small sizes of input spaces. We choose AgeMaxMin and AgeMaxSum of FSCS-ART introduced in Chapter 3, and RRT with aging, InitLim is set to 7, and MaxTrial = 10 as introduced in Chapter 5. We choose these techniques as they are most efficient but their performance are still significantly better than RT.

The experiment results show that AgeMaxMin and RRT are more effective than RT and AgeMaxSum. This shows that most of ART strategies used in this study have been effective for the source test case selection of MT. As the ART techniques used here are not the best ones as found in Chapter 3 and Chapter 5, we believe when we apply the best ART techniques, the performance of MT would be further improved. We believe that the bad perfomance of AgeMaxSum in this study is related to the special failure causing input (FCI) patterns of the mutated versions of `grep` for the implemented MRs, as discussed in Chapter 6. The FCI patterns of this study could be a potential case study in the future investigation of the regularities of FCI in programs with complex input types. In summary, in this study we have demonstrated that some ART strategies have been useful to be applied in the metamorphic testing (MT) area.

Chapter 9

Conclusion

This thesis has contributed some enhancements of the application of adaptive random testing (ART) for testing programs with complex input types. In detail, the contributions are:

- Demonstrating the effectiveness of the application of FSCS-ART on real programs with complex input types. These programs are `grep`, and three of SIEMENS programs: `replace`, `printtokens`, and `printtokens2`.
- Developing new approaches to apply ART by exclusion, called Restricted Random Testing (RRT), to test programs with complex input types.
- Investigating the application of the new approaches of RRT using `grep` as the experimental subject.
- Investigating the regularities of failure causing input patterns in the twenty mutated versions of `grep`.
- Corelating the identified patterns of the failure causing inputs to the ART behaviours in testing the twenty mutated versions of `grep`.
- Investigating the potential of using Metamorphic Testing (MT) as the test oracle for the application of ART.
- Investigating the effectiveness of the application of ART as the source test case selection strategy in MT. In this contribution, `grep` is also used as the experimental subject.

This thesis involved a lot of experiments to demonstrate the effectiveness of ART for testing programs with complex input types. Most experiments used `grep` as the experimental subject. We did this intentionally as most chapters were related to each other, hence this made the comparison of the results easier. In some of the

contributions, we analysed the experimental results to identify the new contributions to the study of adaptive random testing for testing programs with such types.

One of the most significant contributions is the development of new approaches to apply ART by exclusion to test programs with complex input types. This has not been investigated in the previous studies of ART. In this thesis, we also demonstrated that the new approaches of RRT have been effective in testing `grep`. As this is the preliminary study of such application of RRT, there is a lot of work that can be done in the future to improve the present approaches as well as to demonstrate its effectiveness using a larger number of case studies.

The other significant contribution of this thesis is the investigation of the failure causing inputs patterns in programs with complex input types. The previous studies of ART have identified the failure causing input patterns only for programs with numeric input type. Thus, we believe that this study is a significant contribution for this area of research. We are interested to study this further using more case studies to verify the consistency of the regularities of failure causing input (FCI) patterns in programs with complex input types.

In this thesis, we also investigated the potential of using Metamorphic Testing (MT) as the test oracle when applying ART on programs with complex input types. The results showed that MT was a useful technique to be used in the application of ART when other test oracles were not available.

The other contribution of this thesis is the application of ART in other area of testing, in particular in the application of MT. We applied ART in the source test case selection of the MRs defined for the program under test. The experimental data showed that most of ART strategies applied could improve the effectiveness of MT in detecting MR violations.

This thesis also presented the application of FSCS-ART on real programs with complex input types, which were `grep` and three of SIEMENS programs: `printtokens`, `printtokens2`, and `replace`. The experiment results showed that all FSCS-ART strategies outperformed RT significantly in all case studies, except `replace`.

We argue that such bad performance of ART on `replace` is due to the category-choice scheme defined for the program. As there is no complete specification of the SIEMENS programs available and `replace` is much more complex than `printtokens` and `printtokens2`, our category-choice scheme for `replace` might not be accurately relevant to what `replace` actually does. As the category-choice based distance measure is used when applying ART on programs with complex input types, the "quality" of the scheme has a very improtant role to determine the effectiveness of ART strategies. As a future work, it is worthy to conduct an in-depth study of defining a good category-choice scheme for a program under test. This study can be conducted by examining the requirements of creating a good category-choice scheme.

One potential way is by creating different schemes for a program under test independently based on certain criteria. Then, ART is applied to test the program using distance measure based on the different schemes. The "quality" of the schemes can be measured by comparing the effectiveness of ART using those different schemes.

In summary, this thesis has made significant contributions to the study of ART on programs with complex input types by demonstrating the effectiveness of FSCS-ART, developing new approaches of applying RRT, identifying the failure causing input patterns, demonstrating the effectiness of MT as the test oracle of ART, and applying ART on other area of testing.

Bibliography

[1] W. E. Perry, *Effective Methods for Software Testing, the third edition.* Wiley Publishing, Inc, 2006.

[2] N. Parrington and M. Roper, *Understanding software testing.* Ellis Horwood Limited, 1989.

[3] G. J. Myers, *The Art of Software Testing.* John Wiley and Sons, Inc, 1979.

[4] W. E. Howden, "Reliability of the path analysis testing strategy," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 208–215, 1976.

[5] J. B. Goodenough and S. L. Gerhart, "Toward a theory of test data selection," *IEEE Transactions on Software Engineering*, vol. 1, no. 2, pp. 156–173, 1975.

[6] H. Zhu and L. Jin, "Software quality assurance and testing," *Academic Press, Beijing*, 1997.

[7] A. C. Barus, T. Y. Chen, D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: A case study of greedy algorithm," in *In Proceedings of the 3rd IFIP CEE Conference on Software Engineering Techniques (CEE-SET 2008).*

[8] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[9] T. Chen, S. Cheung, and S. Yiu, "Metamorphic testing : a new approach for generating next test cases," Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[10] J. C. Miller and C. J. Maloney, "Systematic mistake analysis of computer programs," *Communications of the ACM*, vol. 6, no. 2, pp. 58–63, 1963.

[11] W. Hetzel, *The complete guide to software testing.* Collins, 1984.

[12] J. W. Duran and J. J. Wiorkowski, "Quantifying software reliability by sampling," *IEEE Transactions on Reliability*, vol. 29, pp. 141–144, 1980.

[13] J. W. Duran and S. C. Ntafos, "A report om random testing," in *Procceedings of the 5th International Conference on Software Engineering*, 1981, pp. 179–183.

[14] ——, "An evaluation of random testing," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 438–444, 1984.

[15] T. Yoshikawa, K. Shimura, and T. Ozawa, "Random program generator for java jit compiler test system," in *Procceedings of the Third International Conference on Quality Software (QSIC 03)*. IEEE Computer Society, 2003, pp. 20–23.

[16] D. Slutz, "Massive stochastic testing of sql," in *PProceedings of the 24th International Conference on Very Large Data Bases (VLDB98)*, 1998.

[17] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference*, ser. Lecture Notes in Computer Science, vol. 3321, 2004, pp. 320–329.

[18] F.-C. Kuo, "On adaptive random testing," Ph.D. dissertation, Swinburne University of Technology, 2006.

[19] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: Adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, 2006.

[20] T. Y., G. Eddy, R. G. Merkel, and P. K. Wong, "Adaptive random testing through dynamic partitioning," in *Proceedings of the 4th International Conference on Quality Software (QSIC 04)*. IEEE Computer Society Press, 2004, pp. 79–86.

[21] Y. Liu and H. Zhu, "An experimental evaluation of the reliability of adaptive random testing methods," in *Proceedings of the 2nd International Conference on Secure System Integration and Relibiality Improvement*. IEEE Computer Society Press, 2008.

[22] A. F. Tappenden and J. Miller, "A novel evolutionary approach for adaptive random testing," *to be appeared on the IEEE transactions on the Reliability*.

[23] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *to appear in Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*. IEEE Computer Society Press, Los Alamitos, CA, 2009.

[24] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favourable conditions for adaptive random testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 6, pp. 805 –825, 2007.

[25] T. Y. Chen and R. Merkel, "Quasi-random testing," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 562–568, 2007.

[26] J. Mayer, "Lattice-based adaptive random testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*. New York, USA: ACM press, 2005, pp. 333–336.

[27] R. Merkel, "Analysis and enhancements of adaptive random testing," Ph.D. dissertation, Swinburne University of Technology, 2005.

[28] T. Y. Chen, T. H. Tse, and Y. T. Yu, "Proportional sampling strategy: A compendium and some insights," *Information and Software Technology*, vol. 58, no. 1, pp. 65–81, 2001.

[29] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proceedings of International Conference of Software Engineering*, 2004, pp. 191–200.

[30] G. Rothermel, E. Sebastian, H. Do, and A. Kinneer, "Software-artifact infrastructure repository." [Online]. Available: http://sir.unl.edu

[31] P. E. Ammann and J. C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[32] P. G. Bishop, "The variation of software survival times for different operational input profiles," in *FTSC-23. Digitest of Papers, the Twenty-Third International Symposium on Fault-Tolerant Computing*. IEEE Computer Society Press, 1993, pp. 98–107. [Online]. Available: http://www.adelard.co.uk/resources/papers/pdf/ftsc23.pdf

[33] T. Y. Chen and D. H. Huang, "Adaptive random testing by localization," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*. IEEE Computer Society, 2004, pp. 292–298.

[34] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and S. P. Ng, "Mirror adaptive random testing," *Information and Software Technology*, vol. 46, no. 15, pp. 1001–1010, 2004.

[35] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Artoo: Adaptive random testing for object-oriented software," in *Proceedings of the 30th International Conference on Software Engineering*, Liepzig, Germany, May 2008, pp. 71–80.

[36] H. Jaygarl and C. K. Chang, "Practical extensions of a randomized testing tool," in *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC'09)*. IEEE Computer Society Press, 2009.

[37] Y. Lin, X. Tang, Y. Chen, and J. Zhao, "A divergence-oriented approach to adaptive random testing of java programs," in *Proceedings of the 24th IEEE/ACM International Conference on International Conference on Automated Software Engineering*, 2009.

[38] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating functional tests," *Communications of the ACM*, vol. 31, no. 6, June 1988.

[39] K. P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *In Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, pp. 485–494.

[40] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *a Special Issue on Service Engineering of International Journal of Web Services Research*, vol. 4, no. 2, pp. 60–80, 2007.

[41] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: a case study," in *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC)*. IEEE Computer Society Press, Los Alamitos , California, 2002, pp. 327–333.

[42] T. Y. Chen, D. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th lbero- American Symposium on Software Engineering and Knowledge Engineering (JIISIC)*. Polytechnic University of Madrid, 2004, pp. 569–583.

[43] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, New York, 2002, pp. 191–195.

[44] ——, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 2, pp. 1–9, 2003.

[45] A. Gotlieb, "Exploiting symmetries to test programs," in *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE)*, 2003, pp. 365–374.

[46] W. K. Chan, T. Y. Chen, T. T. H. Lu, H., and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: a metamorphic approach," *International Journal of Software Engineering*, vol. 16, no. 5, pp. 677–703, 2006.

[47] The GNU Project, "Grep home page," accessed 2006-26-04. [Online]. Available: http://www.gnu.org/software/grep

[48] The GNU Project, "Gnu C library reference manual," July 2001, accessed 2005-05-08. [Online]. Available: http://www.gnu.org/software/libc/manual/html_node/index.html

[49] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th International Conference on Software Engineering*. New York, NY, USA: ACM, 2005, pp. 402–411.

[50] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An experimental determination of sfficient mutant operators," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 2, pp. 99–118, 1996.

[51] H. Agrawal, R. A. DeMillo, R. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. H. Spafford, "Design of mutant operators for the c programming language," Software Engineering Research Center, Purdue University, West Lafayette, Indiana, USA, Tech. Rep. SERC-TR-41-P, March 1989.

[52] T. Y. Chen, F.-C. Kuo, and R. Merkel, "On the statistical properties of testing effectiveness measures," *Journal of Systems and Software*, vol. 79, no. 5, pp. 591–601, 2006.

[53] R. Kirk, *Experimental Design: Procedures for the Behavioral Sciences, third ed.* Brooks/Cole, 1995.

[54] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *In Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2000)*. ACM SIGSOFT Software Engineering Notes, pp. 102–112.

[55] K. P. Chan, T. Y. Chen, and D. Towey, "Normalized restricted random testing," in *Proceedings of 8th Ada Europe Internation Conference on Reliable Software Technologies*. LNCS 2655 Springer-Verlag, June 2003, pp. 368–381.

Appendix A

Category-choices Schemes

`printtokens` and `printtokens2` have the same scheme as they are implemented based on the same specification. Following is the scheme for the two programs:

1. Category NumOfInputs, classes: Input=0 and Input=1. This category specifies the number of parameters of the input. The class "Input=0" is for an input which has no parameters (an empty string input) and the second class is for an input which has one parameter (file input name).

2. Category FileExist, classes: Yes and No This category specifies the existence of the file input. The class is "Yes" if the file is present, otherwise the class is "No".

3. Category HasEmptyString, classes: Yes and No This category specifies the presence of an empty string in the file input. The class is "Yes" if the empty string is present, otherwise the class is "No".

4. Category HasStringLength80, classes: Yes and No. This category specifies the existence of a string with length equal to 80 in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

5. Category HasStringLengthLess80, classes: Yes and No. This category specifies the existence of a string with length less than 80 in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

6. Category HasStringLengthGreater80, classes: Yes and No. This category specifies the existence of a string with length greater than 80 in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

7. Category HasStringWithoutDoubleQuotes, classes: Yes and No. This category specifies the existence of a string having no double quotes in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

8. Category HasStringWithEvenDoubleQuotes, classes: Yes and No. This category specifies the existence of a string enclosed by a pair of double quotes in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

9. Category HasStringWithOddDoubleQuotes, classes: Yes and No. This category specifies the existence of a string not enclosed by a pair of double quotes in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

10. Category BlankInsideEnclosedDoubleQuote, classes: Yes and No. This category specifies the existence of a blank string enclosed by a pair of double quotes in the file input. The class is "Yes" if such a string is present, otherwise the class is "No".

11. Category Has#, classes: Yes and No. This category specifies the existence of # in the file input. The class is "Yes" if # is present, otherwise the class is "No".

12. Category HasCharAfter#, classes: Yes and No. This category specifies the existence of any characters after # in the file input. The class is "Yes" if the characters are present, otherwise the class is "No".

13. Category HasLambda, classes: Yes and No. This category specifies the existence of keyword "lambda" in the file input. The class is "Yes" if the keyword "lambda" is present, otherwise the class is "No".

14. Category HasAnd, classes: Yes and No. This category specifies the existence of keyword "and" in the file input. The class is "Yes" if the keyword "and" is present, otherwise the class is "No".

15. Category HasIf, classes: Yes and No. This category specifies the existence of keyword "if" in the file input. The class is "Yes" if the keyword "if" is present, otherwise the class is "No".

16. Category HasOr, classes: Yes and No. This category specifies the existence of keyword "or" in the file input. The class is "Yes" if the keyword "or" is present, otherwise the class is "No".

17. Category HasXor, classes: Yes and No. This category specifies the existence of keyword "Xor" in the file input. The class is "Yes" if the keyword "or" is present, otherwise the class is "No".

18. Category HasStandAloneAlphaNum, classes: Yes and No. This category specifies the existence of alphanumeric outside double quotes and not after # in the file input. The class is "Yes" if such characters are present, otherwise the class is "No".

19. Category HasLParen, classes: Yes and No. This category specifies the existence of left parenthesis in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

20. Category HasRParen, classes: Yes and No. This category specifies the existence of right parenthesis in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

21. Category HasLBracket, classes: Yes and No. This category specifies the existence of left bracket in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

22. Category HasRBracket, classes: Yes and No. This category specifies the existence of right bracket in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

23. Category HasQuote, classes: Yes and No. This category specifies the existence of a single quote in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

24. Category HasBackQuote, classes: Yes and No. This category specifies the existence of a back quote in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

25. Category HasComma, classes: Yes and No. This category specifies the existence of a comma in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

26. Category HasGreaterEqual, classes: Yes and No. This category specifies the existence of (>=) in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

27. Category HasSpace, classes: Yes and No. This category specifies the existence of space in the file input. The class is "Yes" if the character is present, otherwise the class is "No".

28. Category HasOtherChar, classes: Yes and No. This category specifies the existence of any characters in the file input which are not included in previous

characters. The class is "Yes" if such characters are present, otherwise the class is "No".

Following is the scheme for `replace`:

1. Category NumOfInputParameters, classes: Input=0, Input=1, Input=2, and Input=3. This category specifies the number of parameters of the input. For each class, the class is "Input=N" if the input has N parameters.

2. Category RE_ESC, classes: HasESC and NoESC. This category specifies the existence of escape symbol (@) in the regular expression parameter. The class is "HasESC" if the metacharacter is present, otherwise the class is "NoESC".

3. Category RE_BOL, classes: HasBOL and NoBOL. This category specifies the existence of Beginning of Line symbol (%) as a metacharacter in the regular expression parameter. The class is "HasBOL" if the metacharacter is present, otherwise the class is "NoBOL".

4. Category RE_EOL, classes: HasEOL and NoEOL. This category specifies the existence of End of Line symbol (%) as a metacharacter in the regular expression parameter. The class is "HasEOL" ifthe metacharacter is present, otherwise the class is "NoEOL".

5. Category RE_?, classes: HasMetachar? and NoMetachar?. This category specifies the existence of metacharacter ? in the regular expression parameter. The class is "HasMetachar?" if the metacharacter is present, otherwise the class is "NoMetachar?".

6. Category RE_*, classes: HasMetachar* and NoMetachar*. This category specifies the existence of metacharacter * in the regular expression parameter. The class is "HasMetachar*" if the metacharacter is present, otherwise the class is "NoMetachar*".

7. Category RE_EnumCharSet, classes: HasEnumCharSet and NoEnumCharSet. This category specifies the existence of enumeration type of character set in the regular expression parameter. The class is "HasEnumCharSet*" if the character set is present, otherwise the class is "NoEnumCharSet".

8. Category RE_RangeCharSet, classes: HasRangeCharSet and NoRange-CharSet. This category specifies the existence of range type of character set in the regular expression parameter. The class is "HasRangeCharSet*" if the character set is present, otherwise the class is "NoRangeCharSet".

9. Category RE_MixCharSet, classes: HasMixCharSet and NoMixCharSet. This category specifies the existence of both enumeration and range type of character set in the regular expression parameter. The class is "HasMixCharSet*" if both character sets are present, otherwise the class is "NoMixCharSet".

10. Category RE_MetacharNegate, classes: HasMetacharNegate and NoMeta- charNegate. This category specifies the existence of negate symbol (^) as a metacharacter in the regular expression parameter. The class is "HasMetacharNegate" if the metacharacter is present, otherwise the class is "NoMetacharNegate".

11. Category RE_MetacharDash, classes: HasMetacharDash and NoMetachar-Dash. This category specifies the existence of dash symbol (-) as a metacharacter in the regular expression parameter. The class is "HasMetacharDash" if the metacharacter is present, otherwise the class is "NoMetacharDash".

12. Category RE_MetacharTab, classes: HasMetacharTab and NoMetachar- Tab. This category specifies the existence of tab symbol (@t) as a metacharacter in the regular expression parameter. The class is "HasMetacharTab" if the metacharacter is present, otherwise the class is "NoMetacharTab".

13. Category RE_MetacharNewLine, classes: HasMetacharNewLine and NoMetacharNewLine. This category specifies the existence of new-line symbol (@n) as a metacharacter in the regular expression parameter. The class is "HasMetacharNewLine" if the metacharacter is present, otherwise the class is "NoMetacharNewLine".

14. Category RE_Length, classes: <=MAXSTR, >MAXSTR, and = 0. This category specifies the length of the regular expression parameter. The class is "<=MAXSTR" if the length of the regular expression is less or equal to a pre-determined constant MAXSTR, ">MAXSTR" if the length greater than MAXSTR, "=0" if the length is zero (empty string).

15. Category RS_Esc, classes: HasEsc and NoEsc. This category specifies the existence of escape symbol (@) as a metacharacter in the replacing string parameter. The class is "HasEsc" if the symbol is present, otherwise the class is "NoEsc".

16. Category RS_&, classes: HasMetachar& and NoMetachar&. This category specifies the existence of symbol & as a metacharacter in the replacing string parameter. The class is "HasMetachar&" if the symbol is present, otherwise the class is "NoMetachar&".

17. Category RS_MetacharTab, classes: HasMetacharTab and NoMetacharTab. This category specifies the existence of symbol tab (@t) as a metacharacter in the replacing string parameter. The class is "HasMetacharTab" if the symbol is present, otherwise the class is "NoMetacharTab".

18. Category RS_MetacharNewLine, classes: HasMetacharNewLine and NoMetacharNewLine. This category specifies the existence of symbol new line (@n) as a metacharacter in the replacing string parameter. The class is "HasMetacharNewLine" if the symbol is present, otherwise the class is "NoMetacharNewLine".

19. Category RS_Length, classes: <=MAXSTR, >MAXSTR, and = 0. This category specifies the length of the replacing string parameter. The class is "<=MAXSTR" if the length of the replacing string is less or equal to a pre-determined constant MAXSTR, ">MAXSTR" if the length greater than MAXSTR, and "=0" if the length is zero (empty string).

20. Category F_EndStr, classes: HasEndStr and NoEndStr. This category specifies the existence of end string character in the file referred by the third parameter of an input. The class is "HasEndStr" if the character is present, otherwise the class is "NoEndStr".

21. Category F_String≤MAXSTR, classes: HasString≤MAXSTR and NoString≤MAXSTR. This category specifies of a string shorter than or equal in length to MAXSTR in the file referred by the third input parameter. The class is "HasString≤MAXSTR" if at least a string with length less or equal to MAXSTR is present in the file, otherwise the class is "NoString≤MAXSTR".

22. Category F_String>MAXSTR, classes: HasString>MAXSTR and NoString>MAXSTR. This category specifies the presence of a string greater in length than MAXSTR in the file referred by the third input parameter. The class is "HasString>MAXSTR" if at least a string with length greater than MAXSTR is present in the file, otherwise the class is "NoString> MAXSTR".

23. Category F_EmptyString, classes: HasSEmptyString and NoEmptyString. This category specifies the presence of a string shorter than or equal in length to MAXSTR in the file referred by the third input parameter. The class is "HasEmptyString" if at least an empty string is present in the file, otherwise the class is "NoEmptyString".

Appendix B

Full Experimental Results of FSCS-ART

M-ID	F-measure	sDev	95% CI	
			-	+
1	14.06	13.41	13.23	14.89
2	857.59	838.35	805.63	909.55
3	14.44	13.66	13.60	15.29
4	43.58	41.47	41.01	46.15
5	37.44	36.98	35.15	39.73
6	33.46	33.06	31.41	35.51
7	33.80	34.46	31.66	35.93
8	57.46	54.95	54.05	60.87
9	50.34	50.47	47.21	53.47
10	14.05	13.43	13.22	14.88
11	203.61	196.68	191.42	215.80
12	487.92	502.34	456.78	519.05
13	657.82	621.05	619.32	696.31
14	13.64	13.17	12.82	14.45
15	271.46	267.06	254.91	288.02
16	14.29	13.52	13.45	15.13
17	470.42	462.72	441.74	499.10
18	35.15	35.07	32.98	37.33
19	21.87	21.09	20.56	23.18
20*	6.00	6.66	5.55	6.45

Table B.1: Statistical data for experiments applying RT to `grep`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	6.19	4.69	5.90	6.48	9.45	8.21	8.94	9.96
2	1404.75	1398.33	1318.08	1491.42	1536.50	1459.01	1442.62	1630.37
3	6.63	5.27	6.30	6.95	9.87	8.10	9.36	10.37
4	40.90	43.00	38.24	43.57	57.51	59.46	53.82	61.20
5	15.03	13.73	14.17	15.88	21.14	19.67	19.92	22.36
6	30.39	30.99	28.47	32.31	45.61	51.86	42.39	48.82
7	14.48	13.37	13.65	15.31	22.24	20.74	20.95	23.52
8	24.76	24.20	23.25	26.26	34.11	33.51	32.03	36.19
9	21.11	20.11	19.87	22.36	30.78	27.89	29.05	32.51
10	6.16	4.71	5.86	6.45	9.43	8.21	8.92	9.94
11	91.23	91.10	85.59	96.88	132.01	133.01	123.77	140.25
12	195.44	183.33	184.08	206.80	281.60	293.04	263.44	299.76
13	275.49	271.45	258.66	292.31	372.93	381.14	349.30	396.55
14	6.11	4.63	5.83	6.40	9.30	8.15	8.79	9.80
15	119.25	122.02	111.68	126.81	156.14	150.27	146.83	165.45
16	6.17	4.68	5.88	6.46	9.41	8.18	8.90	9.91
17	199.51	203.68	186.88	212.13	262.00	251.89	246.39	277.62
18	18.12	16.99	17.07	19.17	25.08	24.98	23.53	26.63
19	15.94	16.73	14.90	16.97	22.86	26.48	21.21	24.50
20*	6.04	6.61	5.59	6.49	7.77	10.17	7.07	8.46

Table B.2: Statistical data for experiments applying AgeMaxMin and NoAgeMaxMin to `grep`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	6.15	4.70	5.86	6.45	6.16	4.75	5.87	6.46
2	342.19	355.72	320.14	364.23	238.87	224.23	224.97	252.77
3	6.31	5.00	6.00	6.62	6.32	4.98	6.01	6.62
4	23.20	22.31	21.82	24.58	25.37	28.00	23.63	27.11
5	16.14	15.59	15.17	17.11	15.46	14.47	14.56	16.35
6	14.61	13.45	13.78	15.44	15.56	14.83	14.64	16.48
7	14.53	13.08	13.72	15.34	14.09	12.26	13.33	14.85
8	25.61	25.21	24.05	27.17	24.68	23.11	23.25	26.11
9	22.00	19.03	20.82	23.18	21.83	19.94	20.59	23.06
10	4.00	4.70	5.84	6.42	6.09	4.67	5.80	6.38
11	88.91	90.36	83.30	94.51	84.99	83.51	79.82	90.17
12	220.98	235.96	206.36	235.61	201.23	201.13	188.77	213.70
13	302.76	304.41	283.89	321.63	280.88	271.62	264.05	297.72
14	6.07	4.61	5.78	6.35	6.06	4.65	5.77	6.35
15	117.02	118.50	109.67	124.36	114.62	112.90	107.62	121.62
16	6.14	4.71	5.85	6.43	6.10	4.65	5.81	6.39
17	203.26	205.27	190.54	215.99	200.41	206.67	187.60	213.21
18	15.76	14.54	14.85	16.66	15.26	13.68	14.41	16.11
19	9.38	8.11	8.88	9.88	9.63	8.76	9.08	10.17
20*	5.91	6.35	5.47	6.34	6.00	6.41	5.56	6.43

Table B.3: Statistical data for experiments applying AgeMaxSum and NoAgeMaxSum to `grep`

M-ID	**F-measure**	**sDev**	**95% CI**	
			-	+
1	604.31	574.87	568.68	639.94
2	76.94	75.03	72.29	81.59
3	99.16	101.70	92.86	105.46
4	132.87	131.20	124.74	141.00
5	24.05	24.34	22.54	25.56
6	20.49	19.45	19.28	21.70
7	136.27	131.30	128.14	144.41

Table B.4: Statistical data for experiments applying RT to `printtokens`

M-ID	aging				no aging			
	Mean F-measure	sDev	95% CI		Mean F-measure	sDev	95% CI	
			-	+			-	+
1	214.23	215.02	200.91	227.56	197.78	196.08	185.62	209.93
2	34.77	33.68	32.69	36.86	35.40	34.00	33.29	37.51
3	62.84	60.71	59.08	66.60	65.01	61.29	61.21	68.81
4	72.56	74.06	67.97	77.15	60.79	59.76	57.09	64.50
5	7.23	6.13	6.84	7.61	7.24	6.28	6.85	7.63
6	6.49	5.54	6.15	6.83	6.47	5.40	6.13	6.80
7	62.75	62.62	58.87	66.63	57.82	53.76	54.49	61.15
dcv								

Table B.5: Statistical data for experiments applying AgeMaxMin and NoAgeMaxMin to `printtokens`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	162.37	147.56	153.22	171.51	147.97	138.07	139.41	156.52
2	25.78	24.61	24.25	27.30	25.19	23.38	23.74	26.64
3	40.76	39.54	38.31	43.21	40.64	40.24	38.14	43.13
4	71.32	74.24	66.72	75.92	65.78	67.09	61.63	69.94
5	6.63	5.23	6.30	6.95	6.58	5.09	6.26	6.90
6	6.02	4.81	5.72	6.32	5.96	4.69	5.67	6.25
7	55.44	55.14	52.02	58.85	53.88	55.18	50.46	57.30

Table B.6: Statistical data for experiments with AgeMaxSum and NoAgeMaxSum to `printtokens`

M-ID	**F-measure**	**sDev**	**95% CI**	
			-	+
1	15.70	15.20	14.75	16.64
2	15.21	14.87	14.29	16.14
3	124.74	124.72	117.01	132.47
4	11.34	10.16	10.71	11.97
5	22.35	21.70	21.00	23.69
6	7.56	7.02	7.12	7.99
7	18.13	16.93	17.08	19.18
8	15.64	14.64	14.73	16.54
9	64.62	64.29	60.63	68.60
10	22.39	21.75	21.04	23.74

Table B.7: Statistical data for experiments applying RT to `printtokens2`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	4.85	3.79	4.61	5.08	4.83	3.70	4.60	5.06
2	4.71	3.66	4.48	4.94	4.67	3.53	4.45	4.89
3	63.49	63.48	59.55	67.42	57.01	48.98	53.98	60.05
4	4.08	2.91	3.90	4.26	4.09	2.95	3.91	4.27
5	12.69	11.78	11.96	13.42	13.01	12.20	12.26	13.77
6	3.85	2.82	3.67	4.02	3.87	2.92	3.68	4.05
7	6.44	5.37	6.11	6.78	6.43	5.34	6.10	6.76
8	5.87	4.70	5.58	6.17	5.91	4.94	5.61	6.22
9	16.62	15.64	15.65	17.59	15.80	14.53	14.90	16.70
10	12.66	11.78	11.93	13.39	12.91	12.17	12.16	13.67

Table B.8: Statistical data for experiments applying AgeMaxMin and NoAgeMaxMin to `printtokens2`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	4.68	3.24	4.48	4.88	4.69	3.23	4.49	4.89
2	4.60	3.21	4.40	4.80	4.61	3.19	4.41	4.80
3	68.59	68.49	64.34	72.83	66.19	65.38	62.14	70.24
4	3.97	2.68	3.81	4.14	3.96	2.64	3.80	4.12
5	10.28	8.80	9.74	10.83	10.31	9.00	9.75	10.87
6	3.86	2.95	3.68	4.05	3.86	2.88	3.68	4.04
7	6.24	4.89	5.93	6.54	6.30	4.94	5.99	6.61
8	5.87	4.48	5.60	6.15	5.84	4.46	5.56	6.11
9	14.24	12.86	13.44	15.04	13.85	12.31	13.08	14.61
10	10.26	8.76	9.72	10.80	10.30	8.93	9.74	10.85

Table B.9: Statistical data for experiments with AgeMaxSum and NoAgeMaxSum with `printtokens2`

M-ID	F-measure	sDev	95% CI	
			-	+
1	84.32	81.16	79.29	89.35
2	149.78	145.49	140.76	158.80
3	42.78	42.17	40.16	45.39
4	37.52	36.95	35.23	39.81
5	19.69	19.45	18.49	20.90
6	58.29	59.71	54.59	61.99
7	66.38	62.10	62.53	70.23
8	103.48	100.13	97.28	109.69
9	196.06	193.93	184.04	208.08
10	239.36	231.81	224.99	253.72
11	197.24	195.23	185.14	209.34
12	18.75	17.72	17.65	19.85
13	33.04	32.82	31.01	35.08
14	38.22	38.02	35.87	40.58
15	99.48	95.99	93.53	105.43
16	66.61	62.34	62.75	70.48
17	229.46	225.77	215.46	243.45
18	24.84	23.07	23.41	26.27
19	1713.70	1546.93	1617.82	1809.58
20	254.44	252.56	238.78	270.09
21	1706.95	1519.64	1612.76	1801.14
22	282.10	283.81	264.51	299.69
23	255.39	253.51	239.68	271.11
24	33.20	33.86	31.10	35.29
25	1757.27	1557.57	1660.73	1853.81
26	44.99	44.44	42.23	47.74
27	21.04	20.34	19.78	22.30
28	37.01	34.17	34.89	39.13
29	88.71	91.68	83.03	94.39
30	19.86	18.45	18.71	21.00
31	24.63	22.93	23.21	26.05

Table B.10: Statistical data for experiments applying RT to `replace`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	39.30	37.16	36.99	41.60	37.76	35.30	35.57	39.95
2	79.98	77.20	75.20	84.77	74.75	70.11	70.41	79.10
3	33.74	31.39	31.80	35.69	29.29	25.75	27.69	30.88
4	30.99	28.92	29.20	32.79	27.47	24.33	25.96	28.98
5	18.28	17.44	17.20	19.36	16.69	15.22	15.75	17.63
6	77.11	74.92	72.46	81.75	79.52	79.05	74.62	84.42
7	75.05	70.73	70.67	79.43	75.02	68.99	70.74	79.29
8	124.11	110.03	117.29	130.93	85.35	68.80	81.09	89.62
9	366.54	318.48	346.80	386.28	352.64	290.64	334.62	370.65
10	360.26	300.47	341.64	378.88	348.87	309.83	329.66	368.07
11	365.90	318.27	346.18	385.63	355.76	292.86	337.61	373.91
12	13.29	12.64	12.51	14.07	13.19	12.40	12.43	13.96
13	27.60	25.95	25.99	29.21	24.73	21.23	23.42	26.05
14	43.86	40.80	41.33	46.39	41.71	39.32	39.28	44.15
15	214.29	200.10	201.89	226.69	109.33	73.88	104.75	113.91
16	74.92	70.68	70.54	79.30	75.42	69.25	71.13	79.71
17	261.20	249.51	245.74	276.67	235.62	199.72	223.24	248.00
18	40.31	38.55	37.92	42.70	39.70	38.73	37.30	42.10
19	718.76	570.78	683.38	754.13	463.46	348.72	441.84	485.07
20	274.41	261.91	258.18	290.64	254.93	219.99	241.30	268.57
21	2047.45	1223.83	1971.60	2123.30	820.37	437.35	793.27	847.48
22	471.52	435.74	444.51	498.52	439.15	344.82	417.78	460.52
23	274.48	260.75	258.32	290.64	249.36	205.66	236.61	262.11
24	25.02	24.56	23.50	26.54	23.06	19.85	21.83	24.29
25	557.43	483.93	527.44	587.43	391.94	324.73	371.81	412.06
26	37.69	36.06	35.45	39.92	37.99	36.48	35.73	40.25
27	13.79	12.47	13.01	14.56	13.60	11.59	12.88	14.31
28	19.72	17.73	18.62	20.82	18.07	15.11	17.13	19.00
29	47.70	44.57	44.93	50.46	46.09	44.78	43.31	48.86
30	12.23	11.07	11.54	12.91	11.74	9.84	11.13	12.35
31	40.32	38.40	37.94	42.70	41.44	41.37	38.87	44.00

Table B.11: Statistical data for experiments applying AgeMaxMin and NoAgeMaxMin to `replace`

M-ID	aging				no aging			
	F-measure	sDev	95% CI		F-measure	sDev	95% CI	
			-	+			-	+
1	37.38	34.65	35.23	39.53	34.73	31.73	32.76	36.69
2	66.27	62.51	62.39	70.14	60.72	54.75	57.33	64.12
3	45.09	44.12	42.35	47.82	44.09	45.67	41.26	46.92
4	37.12	35.94	34.89	39.35	38.12	38.48	35.74	40.51
5	23.63	22.68	22.22	25.03	24.48	24.19	22.98	25.98
6	145.42	137.58	136.89	153.95	162.61	155.40	152.98	172.25
7	99.47	91.35	93.81	105.13	111.32	107.69	104.64	117.99
8	246.80	218.05	233.29	260.32	281.56	237.38	266.84	296.27
9	987.57	668.46	946.14	1029.00	1038.28	529.79	934.44	1142.12
10	649.70	480.60	619.91	679.49	599.32	421.83	573.18	625.47
11	987.41	668.01	946.01	1028.81	1118.24	586.00	1076.30	1160.18
12	14.34	14.56	13.44	15.25	13.87	13.49	13.04	14.71
13	33.26	32.13	31.27	35.25	34.56	35.17	32.38	36.74
14	72.44	67.89	68.23	76.64	83.49	76.86	78.73	88.26
15	157.57	144.49	148.61	166.52	177.97	161.26	167.97	187.96
16	99.46	91.71	93.77	105.14	137.07	140.27	128.38	145.77
17	307.82	259.73	291.72	323.92	325.81	263.86	309.45	342.16
18	101.93	102.57	95.57	108.29	140.33	134.52	131.99	148.67
19	722.67	562.38	687.81	757.52	586.41	445.94	558.77	614.05
20	316.95	267.88	300.35	333.55	331.32	267.09	314.77	347.88
21	1965.93	994.14	1904.32	2027.55	1974.62	781.42	1853.54	2095.70
22	766.97	553.06	732.69	801.25	784.26	496.75	753.47	815.04
23	318.17	268.34	301.53	334.80	328.80	265.67	312.33	345.26
24	26.54	25.36	24.97	28.11	27.18	25.90	25.57	28.78
25	459.11	387.14	435.12	483.11	374.96	303.76	356.14	393.79
26	56.29	54.20	52.93	59.65	54.42	51.70	51.21	57.62
27	14.92	13.38	14.09	15.75	15.13	13.98	14.26	15.99
28	22.89	21.90	21.53	24.25	22.00	20.37	20.73	23.26
29	58.10	57.24	54.55	61.65	56.86	53.12	53.56	60.15
30	14.08	13.64	13.23	14.92	13.68	12.36	12.92	14.45
31	102.15	102.50	95.80	108.50	139.55	135.26	131.17	147.94

Table B.12: Statistical data for experiments with AgeMaxSum and NoAgeMaxSum with `replace`

Appendix C

Full Experimental Results of RRT

M-ID	RT	MT10	MT25	MT50	MT	MT100	MT200
Mutant1	14.06	4.16	3.78	3.83	3.99	3.95	3.87
Mutant2	857.59	>>>	>>>	>>>	>>>	>>>	>>>
Mutant3	14.44	4.50	4.52	4.55	4.70	4.51	4.38
Mutant4	43.58	51.79	50.17	48.21	49.02	50.37	47.97
Mutant5	37.44	10.37	9.99	10.34	10.81	10.80	10.19
Mutant6	33.46	55.82	53.20	53.71	52.15	53.77	51.70
Mutant7	33.80	11.03	10.03	10.31	10.48	10.43	10.29
Mutant8	57.46	18.15	17.99	17.69	18.14	16.63	17.44
Mutant9	50.34	15.25	14.83	14.37	14.52	14.12	14.26
Mutant10	14.05	4.08	3.85	3.88	3.74	3.88	3.95
Mutant11	203.61	73.31	70.20	69.46	71.78	67.69	71.81
Mutant12	487.92	156.06	148.11	142.00	143.85	146.47	144.98
Mutant13	657.82	207.59	183.36	173.15	174.37	177.18	180.51
Mutant14	13.64	4.06	3.84	3.99	3.70	3.80	3.79
Mutant15	271.46	91.18	88.00	80.68	82.08	84.58	81.00
Mutant16	14.29	4.19	4.09	4.13	4.17	3.96	3.90
Mutant17	470.42	154.43	142.33	152.45	139.04	137.07	141.07
Mutant18	35.15	14.65	13.13	12.77	13.34	13.43	12.75
Mutant19	21.87	22.60	22.13	21.42	21.44	20.61	19.54
Mutant20	6.00	2.22	2.22	2.37	2.32	2.32	2.35

Table C.1: GMT experiment results:Average F-measure of RT and RRT with various number of MaxTrial (MT) values for `grep`

M-ID	RT	ILD=MaxLimD	ILD=0.5*MaxLimD	ILD=Proportional
Mutant1	14.06	3.78	5.17	4.92
Mutant2	857.59	>>>	>>>	>>>
Mutant3	14.44	4.52	5.69	5.52
Mutant4	43.58	50.17	52.49	51.58
Mutant5	37.44	9.99	11.71	11.86
Mutant6	33.46	53.2	59.55	55.14
Mutant7	33.8	10.03	11.07	11.52
Mutant8	57.46	17.99	18.51	17.56
Mutant9	50.34	14.83	14.70	15.60
Mutant10	14.05	3.85	5.00	5.04
Mutant11	203.61	70.2	71.90	66.33
Mutant12	487.92	148.11	129.71	142.27
Mutant13	657.82	183.36	195.15	188.88
Mutant14	13.64	3.84	4.99	4.91
Mutant15	271.46	88	86.57	86.13
Mutant16	14.29	4.09	5.14	5.00
Mutant17	470.42	142.33	136.38	121.06
Mutant18	35.15	13.13	14.66	13.81
Mutant19	21.87	22.13	23.50	22.12
Mutant20	6.00	2.22	3.26	3.25

Table C.2: GIL experiment results:Average F-measure of RT and RRT with different values of InitLimD (ILD) for `grep`

M-ID	RT	NoAging	Aging
Mutant1	14.06	3.78	8.25
Mutant2	857.59	>>>	437.91
Mutant3	14.44	4.52	8.39
Mutant4	43.58	50.17	20.33
Mutant5	37.44	9.99	20.52
Mutant6	33.46	53.20	15.00
Mutant7	33.80	10.03	20.08
Mutant8	57.46	17.99	32.16
Mutant9	50.34	14.83	29.67
Mutant10	14.05	3.85	8.32
Mutant11	203.61	70.20	109.07
Mutant12	487.92	148.11	248.13
Mutant13	657.82	183.36	356.84
Mutant14	13.64	3.84	8.29
Mutant15	271.46	88.00	141.76
Mutant16	14.29	4.09	8.41
Mutant17	470.42	142.33	265.15
Mutant18	35.15	13.13	17.81
Mutant19	21.87	22.13	8.86
Mutant20	6.00	2.22	1.37

Table C.3: GA experiment results:Average F-measure of RT and RRT with NoAging and Aging approaches for `grep`

List of Publications

A. C. Barus, T. Y. Chen, D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of Heuristic Methods: A Case Study of Greedy Algorithm", in *Proceedings of the 3rd IFIP CEE Conference on Software Engineering Techniques (CEE-SET 2008)*. Brno, Czech Republic, October 13-15, 2008.

A. C. Barus, T. Y. Chen, F.-C. Kuo, R. Merkel, and G. Rothermel, "Adaptive Random Testing of Programs with Arbitrary Input Types: A Methodology and an Empirical Study", *IEEE Transactions on Software Engineering*, invited to submit a revised version.

A. C. Barus, and T. Y. Chen, "Restricted Random Testing on Programs with Complex Input Types", in preparation.

A. C. Barus, T. Y. Chen, F.-C. Kuo, and R. Merkel, "The Investigation of Failure Causing Input Patterns on Programs with Complex Input Types", in preparation.

A. C. Barus, T. Y. Chen, F.-C. Kuo, and R. Merkel, "Metamorphic Testing: An Alternative Test Oracle for Evaluating Adaptive Random Testing", in preparation.

A. C. Barus, T. Y. Chen, F.-C. Kuo, and R. Merkel, "The Application of Adaptive Random Testing on the Source Test Case Selection of Metamorphic Testing", in preparation.