Master of Science Thesis
University of Twente
Formal Methods and Tools

Evaluating and Predicting Actual Test Coverage

Mark Timmer
June 24, 2008

Committee:
Dr. M.I.A. Stoelinga (UT/FMT)
Dr. ir. A. Rensink (UT/FMT)
Prof. Dr. J.C. van de Pol (UT/FMT)

When it is not in our power to determine what is true, we ought to follow what is most probable.

- René Descartes

Abstract

This thesis proposes a new notion of semantic coverage in formal testing: *actual coverage*. It is defined for test case and test suite executions, as well as for sequences of their executions. A fault is considered to be completely covered if an execution showed its presence, and it is considered partly covered if an execution increased the confidence in its absence.

Actual coverage can be used to evaluate a test process after it has taken place, but we also describe how to predict actual coverage in advance. To support these estimations, a probabilistic execution model is introduced. We derive efficient formulae for both the evaluation and the prediction of actual coverage, making tool support feasible.

We show that for an infinite number of executions our measure coincides with an existing notion of semantic coverage, called potential coverage. This notion, however, does not deal with the fact that in practice only a finite number of executions will be performed. With actual coverage it is possible to predict the actual coverage of any given number of test case or test suite executions.

An extensive detailed example is provided to demonstrate the applicability of our measure.

Acknowledgements

This thesis is, besides the end of my final project, also the end of six years as a student at the University of Twente. I look back at this period with great joy. It gave me the opportunity to become fascinated by the world of theoretical computer science, and in addition provided me with several means to learn much more.

First, I had the honour of being a board member of two amazing associations: I.C.T.S.V. Inter-*Actief* and D.S.V. 4 happy feet. My efforts for them taught me basic management skills, but more importantly I made a lot of friends who I hope will stick around for a long time.

Second, the faculty of Electrical Engineering, Mathematics and Computer Science allowed me to teach numerous *practica* (lab sessions), *werkcolleges* (exercise classes), and even some *hoorcolleges* (lectures). I am very grateful for this wonderful opportunity to 'contaminate' as many students as possible with my enthusiasm for mathematics, programming and formal methods.

With regard to my thesis, I want to thank my main supervisor Mariëlle Stoelinga. During the previous nine months she guided me in the process of obtaining a coherent, consistent, and most importantly understandable theoretical framework. In spite of my stubbornness on several issues, she persistently kept trying to get me to improve my work.

I also want to thank my other supervisors, Arend Rensink and Jaco van de Pol. They provided many useful remarks on drafts of this thesis as well.

The process of creating this thesis would have been a lot less fun without my friends in the *afstudeerhok*. I would like to thank Frank, Paul, Wouter, Viet-Yen, Jorge, Jan-Willem V, Jan-Willem K, Alfons, David, Niek, and Erwin for the many laughs we had. (Let the order of these names not be interpreted as a final statement with respect to our endless not-so-serious discussions on the importance of formal methods versus software engineering, as they are just based on everyone's seating location in our room).

I would also like to thank Jaco van de Pol, Mariëlle Stoelinga and Joost-Pieter Katoen for providing me the opportunity to work as a PhD student in Enschede.

I would like to thank my closest friends and my family.

Finally, my greatest gratitude goes to Thijs, for supporting me and surviving my countless moments of unnecessary stress while producing the final version of this thesis.

Enschede, June 2008
Mark Timmer

Table of Contents

1 Introduction **1**

1.1 Motivation 2

1.1.1 The intuition behind potential coverage 2

1.1.2 The limitations of potential coverage 3

1.2 Overview and results 4

1.2.1 Main results 4

1.3 Related work 5

1.3.1 Code coverage 5

1.3.2 Specification coverage 6

1.3.3 Probabilistic approaches to testing 6

2 Preliminaries **9**

2.1 Basic notations 9

2.2 Input-output labeled transition systems 10

2.2.1 The basics 10

2.2.2 Traces and paths 12

2.3 Test cases for LTSs 13

2.4 Weighted fault models 14

2.4.1 Coverage measures based on weighted fault models 15

2.4.2 Consistency of WFMs with LTSs 16

2.5 Fault automata 17

2.6 From FA to WFM 18

2.6.1 Finite depth weighted fault models 18

2.6.2 Discounted weighted fault models 19

3 Nondeterministic fault automata **21**
3.1 Nondeterministic LTSs . . . 22
3.1.1 Determinising LTSs . . . 22
3.2 Nondeterministic FAs . . . 23
3.2.1 Determinising FAs . . . 23
3.3 Dealing with quiescence . . . 24

4 From potential to actual coverage **27**
4.1 The limitation of potential coverage . . . 27
4.2 Requirements for a new notion of coverage . . . 28
4.3 The intuition behind actual coverage . . . 30
4.4 The formal ingredients of actual coverage . . . 30
4.5 An introductory example . . . 32

5 Probabilities in test case executions **35**
5.1 Probability spaces and random variables . . . 36
5.2 Conditional probabilities . . . 37
5.3 The test case execution experiment . . . 37
5.4 Trace occurrence and branching . . . 39
5.5 Conditional branching probabilities . . . 41
5.6 The probabilistic execution model . . . 42
5.7 Probabilistic fault automata . . . 43
5.8 Deriving p^{cbr} and p^{br} . . . 44
5.8.1 Branching probabilities assuming a flawless system . . . 45
5.8.2 Presence probabilities . . . 45
5.8.3 Deriving p^{cbr} . . . 46
5.8.4 Deriving p^{br} . . . 49
5.9 Risk-based testing . . . 52

6 Evaluating actual coverage **55**
6.1 Basic notion of fault coverage . . . 55
6.2 Coverage probabilities . . . 56
6.2.1 Obtaining a certain coverage probability . . . 58
6.3 Fault coverage and actual coverage . . . 58
6.4 Another approach to coverage probabilities . . . 60
6.4.1 Obtaining a certain coverage probability . . . 63

7 Predicting actual coverage **65**
7.1 Probability distributions for actual coverage 65
7.2 Expected actual coverage 67
7.3 Variance of actual coverage 74
7.4 An approximation for coverage prediction 75

8 Actual coverage of test suites **81**
8.1 Probabilities in test suite executions 81
8.2 Evaluating actual coverage for test suites 82
8.3 Predicting actual coverage of test suites 85

9 A detailed example **93**
9.1 The specification of a simple system 93
9.2 Implementations of the system 95
9.3 Test cases for the implementations 97
9.3.1 Calculating potential coverage 97
9.3.2 Calculating expected actual coverage 98
9.4 Executing the test cases and test suite 100
9.4.1 Simulating test case executions 102

10 Conclusions and Future Work **105**
10.1 Summary of the results 105
10.2 Evaluation 106
10.3 Future work 107

A The subset construction algorithm **109**
A.1 An example 110
A.2 Complexity 111

B Lemmas for Theorem 7.7 **115**

References **117**

List of Notations, Functions and Abbreviations **121**

Chapter 1

Introduction

In the last decades, software has become more and more complex, making it more and more important to perform *testing*; the process of finding faults in an already implemented system, investigating its quality and reducing the risk of unexpected behaviour.

As indicated by several papers, such as [ZE00] and [SLK01], about half of a software project's budget is spent on testing. Still, the United States National Institute of Standards and Technology has assessed that software errors cost the U.S. economy about sixty billion dollars annually [New02]. The institute estimated that more than a third of these costs could be eliminated if testing occurred earlier in the software development process.

It should therefore come as no surprise that the theory of testing has become an intensively studied academic subject. Several problems have been (partially) addressed, such as test case generation (e.g. using TorX [BFS05]), languages for describing tests (e.g. TTCN [PM92]), and formalisms for describing systems (e.g. LOTOS [BB87]). An extensive overview can be found in the famous book of Myers [Mye79] [MSBT04].

The fact that testing is still the topic of many international scientific conferences (e.g. TESTCOM, FATES, QEST, FASE, CONCUR) clearly indicates that a lot of work yet has to be done.

Since practically every system can potentially perform an infinite number of different sequences (*traces*) of actions, testing is unfortunately inherently incomplete; no test suite will be able to find every possible fault in a non-trivial system. This insight is not very recent, as it was already captured by Dijkstra in a famous quote almost forty years ago: *"Program testing can only be used to show the presence of bugs, but never to show their absence"* [Dij70]. Although the purpose of this statement was to direct towards formal verification, we also consider it a starting point for the quest to estimate test suite quality.

The inherent incompleteness of test suites brings us to the main topic of this research project: *test coverage*. Since no test suite can be 'perfect', it is important to be able to quantitatively assess how many faults we expect it to find. Also, we want to be able to derive afterwards how useful an execution or a sequence of executions has been. For this purpose, the notion of *actual coverage* is introduced.

We apply methods from the area of model-based testing, which uses formal models such as labeled transition systems to automatically generate, execute and evaluate test suites.

Organisation of this chapter
First, we explain the motivation behind our contribution to the field of test coverage in Section 1.1. Then, Section 1.2 provides an overview of this thesis, discussing the main results. Finally, Section 1.3 puts our work in perspective by discussing related work.

1.1 Motivation

Most papers applying test coverage define it as a quantitative measure to estimate the quality of a test suite. It is often based on certain system characteristics, and the extent to which they are 'covered' by a test suite.

Almost all definitions of coverage take a *syntactic* point of view. They are for instance based on the number of statements executed by a test case, or the number of branches taken [MSBT04]. A major disadvantage of using these notions is that testing systems that behave identically, but are implemented differently, might result in different coverage values. We might even get different coverage values if we replace the specification by a semantically equivalent, but syntactically different one. This is undesirable, since it is more important to know that most of the behaviour of a system is correct, than to know that most of its syntactic constructs are correct. For instance, replacing the statement '`i = i + 2;`' by '`i++; i++;`' has an impact on the statement coverage a test case yields, even though the amount of functionality that is tested for correctness does not change.

Not many definitions of coverage from a *semantic* point of view have been provided. However, a start has been made in a paper by Brandán Briones, Brinksma and Stoelinga [BBS06], as part of the PhD thesis of Brandán Briones [Bri07]. This work has already laid down a semantic framework for test comparison, where coverage of a test case is defined as the number of *potential faults* that are *potentially detected*, weighted by the severity of each fault. Therefore, the notion of coverage introduced in this work will from now on be called *potential coverage*. This coverage measure actually deals with the observable behaviour of a system, not being concerned by the syntactic properties of the implementation or the specification.

1.1.1 The intuition behind potential coverage

The starting point in [BBS06] is the testing framework of [Tre96]. It is based on input-output labeled transition systems (IOLTSs, shortened to LTSs) and a conformance relation called `ioco`. The LTSs contain both input actions and output actions, explicitly making a distinction between actions the user provides and actions the system provides. Furthermore, a special output action δ denotes *quiescence*. Quiescence means that the system does not produce any output actions until the user provides an input action. The conformance relation `ioco` basically states that an implementation is correct if it can always handle every possible input action, and it can never produce an unexpected output action.

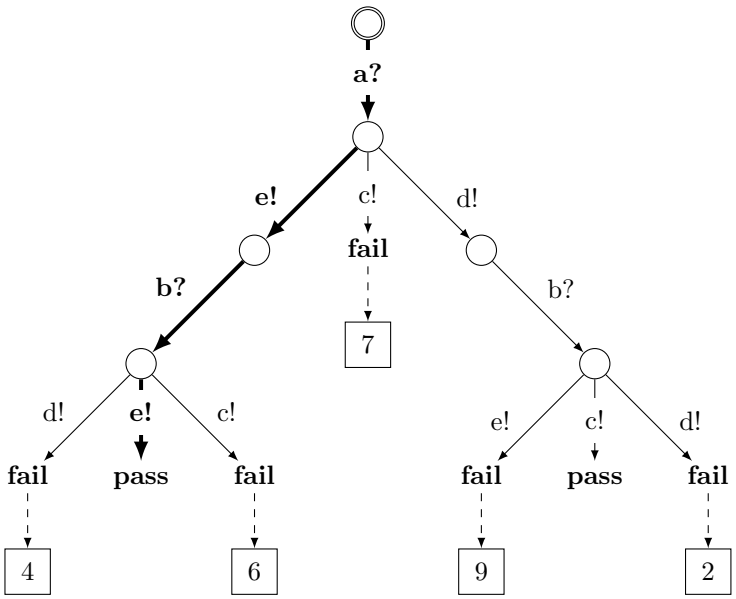


Figure 1.1: An example test case

Since δ is considered as an output action, the unexpected absence of all output actions is also considered erroneous.

[BBS06] introduces weighted fault models, which assign an error weight (in $\mathbb{R}^{\geq 0}$) to each trace. As an example, consider the test case shown in Figure 1.1. We show the error weights of all the erroneous traces that can potentially be detected by this test case. For example, if the system produces a *c!* after an *a?* has been provided by the user, that will be considered incorrect. Therefore, we assign a positive error weight (in this case 7), indicating the severity of the fault. All correct traces receive an error weight of 0.

The erroneous traces this test case can observe are *a? e! b? d!*, *a? e! b? c!*, *a? d! b? e!*, *a? d! b? d!*, and *a? c!*. The measure of potential coverage now states that the *absolute coverage* of this test case is $7 + 4 + 6 + 9 + 2 = 28$ (the accumulated weight of its erroneous traces). When this value is divided by the total accumulated weight of all erroneous traces that could occur in the system (the *total coverage*), this is called the *relative coverage* of the test case. Assuming that the total coverage is 200, the relative coverage of the test case under consideration is 14 percent.

1.1.2 The limitations of potential coverage

As explained above, potential coverage indicates which faults can *potentially* be detected by a test case. If a test case is executed once, however, not all erroneous traces can actually be shown present or absent [HT96]. Consider the test case of Figure 1.1 again. After the input action *a?* one of the outputs *e!*, *c!* or *d!* will occur. After *e!* occurs, there is no way to know if a fault might occur in traces starting with *a? d!*. Therefore, not all the traces that can potentially be covered by this test case are actually covered in an execution.

Because of this discrepancy, it is not possible to use potential coverage to

predict how many faults will be covered when executing a test case or test suite a certain number of times. After all, we would need a model of the probabilistic behaviour of the system. If for example the probability of the system choosing an *e!* is much larger than the probability of the system choosing a *d!*, we expect to need more executions before all faults are covered than if the probabilities were equal.

To solve these limitations, a new notion for coverage is introduced here: *actual coverage*.

Moreover, the framework as it is restricts itself to *deterministic* fault automata. Therefore, a system that is described by a *nondeterministic* LTS should first be transformed into an equivalent deterministic LTS, before a fault automaton can be made to describe the severity of its erroneous traces. This project aims at extending the framework by also allowing nondeterministic fault automata, such that error weights can directly be assigned to erroneous traces in the nondeterministic LTS.

1.2 Overview and results

This thesis is divided into 10 chapters. Chapter 2 first gives an overview of the framework developed in [BBS06]. Then, in Chapter 3 we explain in detail how the methods of Chapter 2 can be used for nondeterministic system descriptions. It turns out that this extension is not possible considering the current definition of quiescence. We therefore propose an extended notion of quiescence.

Chapter 4 explains the motivation behind actual coverage, and provides an intuition for its definition. This chapter also informally introduces the ingredients of our theoretical framework on actual coverage, which are formally discussed in Chapter 5 through 8.

Chapter 9 uses a detailed example to illustrate all the concepts of our framework. Several specifications and calculations are provided, giving a feeling for the process of applying our methods. Obviously, many of the calculations performed here by hand should in practice be performed by a tool.

Chapter 10 concludes this thesis by evaluating the developed methods. Also, it gives directions for future work.

1.2.1 Main results

The main results of this thesis are summarised as follows.

- We extend the existing framework for potential coverage from [BBS06], explaining in detail how to deal with *nondeterministic systems*. The notion of *quiescence* is updated to support its preservation under determinisation. (Chapter 3)
- We develop a new notion of coverage: *actual coverage*. It deals not only with test cases or test suites, but also with the *number of executions* planned and the *probabilistic behaviour* of the system. (Chapter 4)
- We present *probabilistic execution models* (PEMs), containing the probabilities necessary to calculate actual coverage. We introduce *probabilistic fault automata* (PFAs) to syntactically specify PEMs. We also provide guidelines for obtaining the probabilities. (Chapter 5)

- We formally define the actual coverage of a given execution or sequence of executions. *Coverage probabilities* are introduced to take into account how certain we are of the absence of faults. (Chapter 6)

- We provide an *efficient calculation* to predict the actual coverage of test cases. (Chapter 7)

- We generalise all the methods above to test suites. (Chapter 8)

Combining the results described above, we obtain a framework that describes the expected behaviour of a system and the expected outcome of test case and test suite executions. The framework is useful in test evaluation, but also in test selection. Since the most important properties can be calculated in polynomial time (approximately based on the number of times the test case is to be executed multiplied by the number of possible outcomes), it seems feasible to implement the theory in a tool.

1.3 Related work

In the past decades many papers on test coverage have been published. Several different definitions have been used, each with its own properties. According to [WS00], coverage is generally defined as 'the number of faults *detected*, divided by the number of potential faults'. This means that a test case execution that does not detect any errors has by definition no coverage. Since observing the absence of faults also increases our confidence in a system, this definition does not satisfy our needs.

The most important related work for this project is [BBS06], since it describes the framework this project extends. Moreover, since the framework is based on `ioco` testing, an important paper is [Tre96], defining that formalism.

In the following, we will first discuss the most important related work on code coverage. Then, we discuss related work concerning specification coverage. Finally, we provide directions towards interesting work on probabilistic test approaches.

1.3.1 Code coverage

Most papers on coverage describe a form of *code coverage*, defining coverage based on the implementation. An excellent overview can be found in [Mye79], or the more recent edition [MSBT04]. Some extensions to the traditional approached were proposed in [Bal05]. Here we give a short summary of some code coverage criteria.

Statement coverage is the weakest form of code coverage. It demands all statements to be executed at least once. *Decision coverage* is already a bit stronger. It requires a test suite to contain enough tests such that each decision evaluates at least once to `true` and once to `false`. The strongest code coverage criterion is *path coverage*. It requires that all possible sequences of statements are included in a test suite. Since the occurrence of loops often results in an infinite number of paths, path coverage is not very realistic in practice.

1.3.2 Specification coverage

The aforementioned methods are, unfortunately, not invariant under replacement of a system with a semantically equivalent one. Therefore, besides code coverage, also *specification coverage* has been described extensively in literature. Techniques such as equivalence partitioning and boundary value analysis are well-known, and are described in every text book on testing. They are, however, by far not sufficient to completely test a system.

For testing techniques applying finite state machines, a good overview is provided by [Ura92]. Furthermore, many details on the principles of testing finite state machines can be found in [LY96] and [Yan91]. Often, a distinction is made between state coverage and transition coverage.

State coverage makes sure that every state of the specification is visited by a test case at least once. Since finite state machines have a finite number of states, such a test suite is feasible. Transition coverage extends state coverage by not only requiring every state to occur in a test suite, but also every transition between them.

Although these coverage measures are based on the specification instead of the implementation, they are still of a syntactic nature. Coverage measures depending on the number of states that were visited or the number of transitions that were taken are focused on these syntactic issues, instead of the actual behaviour the system exhibits. Equivalent specifications might therefore yield different coverage values.

Furthermore, although finite state machines are a useful way to model systems, they have several disadvantages. First of all, nondeterminism is not allowed. This does not restrict their expressiveness, but it does restrict their ease of use. Also, it is not possible to specify several output actions occurring sequentially, since the formalism restricts us to an alternation between inputs and outputs. The solution is to consider a more modern formalism to express system behaviour formally: *labeled transition systems*. This formalism is also used in this thesis.

In papers discussing testing based on labeled transition system, coverage is often forgotten, ignored or 'solved' by making assumptions and restrictions that result in complete coverage [TPB95]. Since test suites for systems with infinite behaviour are inherently incomplete, this does not reflect reality very well.

An important tool in the area of model-based testing is Microsoft's Spec Explorer. Instead of labeled transition systems it uses *model programs*: a machine-executable specification formalism. According to [CGN$^+$05], coverage metrics are especially difficult in case of nondeterminism and have therefore not yet been implemented. In [NVS$^+$04] Markov Decision Processes are applied to optimize coverage when deriving test suites, but since coverage is defined based on final states these methods are syntactic as well.

1.3.3 Probabilistic approaches to testing

The use of probability in testing was already a subject of research more than twenty years ago. In 1985, [Wun85] described a tool used to estimate the detection probability for each fault in a digital circuit. These probabilities were used to describe the testability of circuits, and to predict how many random tests had to be generated to achieve some required fault coverage.

More recent approaches regarding software conformance testing apply formal methods [Tre96]. In [HT96], probabilities were added to the work of [Tre96], to describe how a system behaves under a certain test case. This paper argues that it is difficult, or even impossible, to be sure that all possible outcomes have really been observed. Based on this understanding, it proposes to consider test executions in a probabilistic setting. Furthermore, it includes the gravity of errors in implementations by assigning an error weight to each implementation. This differs significantly from the approach taken in [BBS06], where error weights were used to denote the severity of individual faults. Probabilities are assigned to implementations as well, indicating how likely the occurrence of each implementation is. Especially the probabilistic approach of [HT96] shows resemblances to our work, although it is much less thorough.

Based on [HT96], [Gog03] introduces a probabilistic coverage for on-the-fly test generation algorithms. Its coverage measure is very different from the measure we introduce. In [Gog03], coverage is defined as 'the weighted probability of being able to reject an implementation divided by the probability that an erroneous implementation occurs'. The main disadvantage of this notion is that it does not take into account *how many* faults are or might be detected during the testing process. Assuming that for complex systems every implementation has at least one detectable fault, they would always achieve 100% coverage.

A side-step from probabilistic testing is risk-based testing. It assessing the risk associated which each fault, and aims at detecting the more risky faults [Aml00]. We include this approach as an optional extension to our notions.

Chapter 2

Preliminaries

The main focus of this thesis is to extend and improve the semantic framework for test coverage introduced in [BBS06]. Therefore, this chapter will discuss this previous work. Furthermore, we introduce the basic mathematical notations that will be used in the subsequent chapters.

The framework of [BBS06] is based on input-output labeled transition system, and test cases for them constructed based on `ioco` theory. Weighted fault models are introduced as semantic structures for assigning error weights to erroneous traces. Using these models, coverage measures are defined.

To describe weighted fault models in a syntactically feasible way, fault automata have been defined. Two mechanisms have been provided for converting these fault automata into finite weighted fault models: a mechanism based on finite depths and another based on discounting.

Organisation of this chapter
First, Section 2.1 introduces some basic notation. Then, input-output labeled transition systems are covered in Section 2.2, followed by test cases for them in Section 2.3. Section 2.4 introduces weighted fault models, as well as coverage measures for test cases based on them. Finally, fault automata are covered in Section 2.5, and their two conversion mechanisms in Section 2.6.

2.1 Basic notations

Definition 2.1. *Let L be any set, then a* trace *over L is a finite sequence of elements from L. Traces will be denoted by their elements, separated by white spaces. The set of all traces over L is denoted by L^*. When a trace does not contain any elements, it is called the empty trace and denoted by ϵ. For any trace σ, its length $|\sigma|$ is defined as the number of elements it consists of. Finally, $L^+ = L^* \setminus \{\epsilon\}$ is the set of all non-empty traces.*

When describing traces, we use the notational convention that $a_i \dots a_{i-1} = \epsilon$, $a_i \dots a_i = a_i$, and $a_0 \dots a_2 = a_0 a_1 a_2$, etcetera.

The notation $\mathscr{P}(S)$ will be used to denote the powerset of a set S.

Example 2.2. Considering the alphabet $L = \{a, b, c, d\}$, we can identify among

others the traces $a\ d\ b\ c$, $b\ a$, $b\ c\ b\ d$ and ϵ. If $\sigma = a\ a\ d$, then $|\sigma| = 3$. Furthermore, although $\epsilon \in L^*$, we have $\epsilon \notin L^+$.

Definition 2.3. *Let $\sigma, \rho \in L^*$ be traces, then σ is a* prefix *of ρ (denoted $\sigma \sqsubseteq \rho$) if there exists a $\sigma' \in L^*$ such that $\sigma\sigma' = \rho$. When $\sigma' \in L^+$, σ is called a* proper prefix *of ρ (denoted $\sigma \sqsubset \rho$).*

A set of traces T is prefix-closed *if for each trace $\sigma \in T$, also all its prefixes are contained in T.*

Example 2.4. The set $T = \{a\ b, a\ b\ c, a\ b\ c\ d\}$ of traces over $L = \{a, b, c, d\}$ is *not* prefix-closed. After all, a is not contained in T, even though it is a prefix of all the other traces in T. Also, T does not contain the trace ϵ, which is by definition required to be in all non-empty prefix-closed trace sets. If we would add these two traces to T, it would become prefix-closed.

Definition 2.5. *Let T be a set a traces, then a trace $\sigma \in T$ is* maximal in T *if there does not exist a trace $\sigma' \in T$ such that $\sigma \sqsubset \sigma'$.*

The next definition provides the function *Distr*, which maps any sample space S on all possible probability distributions over S.

Definition 2.6. *Let S be a set, then Distr(S) is the set containing all functions $p : S \to \mathbb{R}^{\geq 0}$ such that*

$$\sum_{s \in S} p(s) = 1$$

Finally, we define the membership of a tuple.

Definition 2.7. *Let $E = (e_1, e_2, \ldots, e_n)$ be a tuple. Then, $e \in E$ is used as a shorthand notation for $\exists e_i : e_i = e$.*

2.2 Input-output labeled transition systems

The semantic framework uses input-output labeled transition systems (IOLTSs, shortened to LTSs) for specifying the behaviour of systems, following `ioco` testing theory [Tre96]. We first define LTSs in terms of sets and relations, and then define paths and traces in LTSs.

2.2.1 The basics

An input-output labeled transition system specifies the behaviour of a system in terms of states and transitions. Transitions from one state to another are always caused by either an input action (often explained as the user pressing a button) or an output action (often explained as the system giving information or goods to the user).

Definition 2.8. *An input-output labeled transition system $\mathcal{A}$ is given by a tuple $\langle S, s^0, L, \Delta \rangle$, where*

- *S is a finite set of states;*
- *s^0 is the initial state;*
- *L is a finite set of actions, partitioned into a set L^I of* input actions *and a set L^O of* output actions *($L = L^I \cup L^O$ and $L^I \cap L^O = \varnothing$);*

- $\Delta \subseteq S \times L \times S$ *is the transition relation, which is required to be deterministic, i.e. if* $(s, a, s') \in \Delta$ *and* $(s, a, s'') \in \Delta$*, then* $s' = s''$*.*

The components of $\mathcal{A}$ *are denoted by* $S_{\mathcal{A}}$*,* $s^0_{\mathcal{A}}$*,* $L_{\mathcal{A}}$*, and* $\Delta_{\mathcal{A}}$*. When the context makes it clear that a component belongs to* $\mathcal{A}$*, the subscript is omitted.*

An element $a \in L$ *is often denoted a? if it is an input action, and a! if it is an output action.*

LTSs modeling implementations are always assumed to be input-enabled*, meaning that all inputs can be provided from every state of the system.*

Definition 2.9. *Let* $\Delta \subseteq S \times L \times S$ *be a transition relation with* L *partitioned into* L^I *and* L^O*, and* $s \in S$*, then*

- Δ^I *is the restriction of* Δ *to* $S \times L^I \times S$
- Δ^O *is the restriction of* Δ *to* $S \times L^O \times S$
- $\Delta(s) = \{(a, s') \mid (s, a, s') \in \Delta\}$
- $\Delta^I(s) = \{(a, s') \mid (s, a, s') \in \Delta^I\}$
- $\Delta^O(s) = \{(a, s') \mid (s, a, s') \in \Delta^O\}$

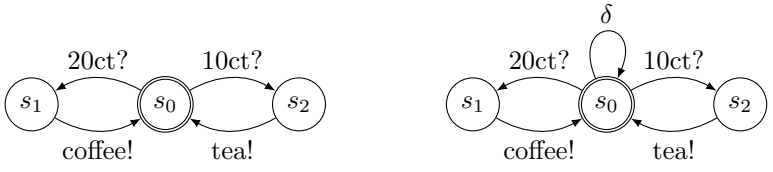
Example 2.10. Suppose we have a coffee machine providing coffee for 20 cents, and tea for 10 cents. The machine only accepts 10 cent coins and 20 cent coins. When a 10 cent coin is inserted, the machine does not wait for a second coin but immediately assumes the user wants tea. Inserting a 20 cent coin results in a cup of coffee.

Figure 2.1(a) shows the LTS $\mathcal{A}$ specifying the behaviour of this coffee machine. Formally, $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$, with

$$
\begin{aligned}
\mathrm{S} &= \{s_0, s_1, s_2\} \\
s^0 &= s_0 \\
L &= L^I \cup L^O, \text{with } L^I = \{\mathit{10ct?}, \mathit{20ct?}\} \text{ and } L^O = \{\mathit{coffee!}, \mathit{tea!}\} \\
\Delta &= \{(s_0, \mathit{10ct?}, s_2), (s_0, \mathit{20ct?}, s_1), (s_1, \mathit{coffee!}, s_0), (s_2, \mathit{tea!}, s_0)\}
\end{aligned}
$$

Applying Definition 2.9, we have

$$
\begin{aligned}
\Delta^I &= \{(s_0, \mathit{10ct?}, s_2), (s_0, \mathit{20ct?}, s_1)\} \\
\Delta^O &= \{(s_1, \mathit{coffee!}, s_0), (s_2, \mathit{tea!}, s_0)\} \\
\Delta(s_0) &= \{(\mathit{10ct?}, s_2), (\mathit{20ct?}, s_1)\}
\end{aligned}
$$



(a) The LTS (b) The LTS including quiescence

Figure 2.1: An LTS $\mathcal{A}$ of a coffee machine

Sometimes it is convenient to incorporate quiescence (the absence of outputs). To do this, we add a transition labeled δ from each quiescent state s ($\Delta^O(s) = \varnothing$) to itself. We define δ to be an output action. Figure 2.1(b) shows the resulting LTS for Example 2.10.

Definition 2.11. *We use a shorthand notation for an LTS almost equivalent to another, differing only in the start state. Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be an LTS and $s \in S$, then $\mathcal{A}[s] = \langle S, s, L, \Delta \rangle$.*

2.2.2 Traces and paths

Since LTSs define a structure with states and transitions between them, we can speak of paths within this structure. A *path* is a connected sequence of transitions in an LTS, specified by the states visited and the actions of the transitions that are taken. Such a path is based on a *trace* over the alphabet of the LTS; a list of actions to be processed (or produced) by the system consecutively.

A state is defined *reachable* from some other state if and only if a path exists between them.

Definition 2.12. *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be an LTS, then*

- *A path in $\mathcal{A}$ is a finite sequence $\pi = s_0 a_1 s_1 a_2 \dots a_n s_n$, with $s_0 = s^0$ and $\forall i \in [0..n-1] : (s_i, a_{i+1}, s_{i+1}) \in \Delta$. The set of all paths in $\mathcal{A}$ is denoted by $paths_{\mathcal{A}}$ and the last state of π is denoted by $last(\pi)$. The length of a path π is denoted by $|\pi|$ and is equal to the number of* states *visited by π (including the first and the last).*
- *Each path π has a trace associated with it, denoted by $trace(\pi)$ and given by the sequence of the actions of π. From all the paths in $\mathcal{A}$ we can easily deduce all the traces in $\mathcal{A}$: $traces_{\mathcal{A}} = \{trace(\pi) \mid \pi \in paths_{\mathcal{A}}\}$.*
- *For each trace $\sigma \in L^*$, $reach_{\mathcal{A}}(\sigma)$ is the set containing all the states that can be reached by following σ in $\mathcal{A}$. Formally, $s \in reach_{\mathcal{A}}$ if $\exists \pi \in paths_{\mathcal{A}} :$ $trace(\pi) = \sigma \wedge last(\pi) = s$. For a deterministic LTS, $final_{\mathcal{A}}(\sigma)$ is defined as the single state contained in $reach_{\mathcal{A}}(\sigma)$.*
- *The set of all states reachable in $\mathcal{A}$ is denoted by $reach_{\mathcal{A}}$, and is formally defined by $reach_{\mathcal{A}} = \bigcup_{\sigma \in L^*} reach_{\mathcal{A}}(\sigma)$.*

Note that we often write s_i for the states of an LTS, and we also use s_i in the definition of a path. However, these are not necessarily equal; the subscripts are for reference purposes only. It is not required that the second state of a path is state s_1, for example.

Furthermore, since $\Delta_{\mathcal{A}}$ is defined to be deterministic, $reach_{\mathcal{A}}(\sigma)$ contains exactly one state in case σ is a trace in $\mathcal{A}$, and zero states otherwise.

Again, the subscript $\mathcal{A}$ is omitted when it is clear from the context which LTS a concept is related to.

Looking once more at Example 2.10, we can identify the path $\pi = s_0$ *10ct?* s_2 *tea!* s_0 *10ct?* s_2 *tea!* s_0. By definition, $trace(\pi) =$ *10ct? tea! 10ct? tea!*. Note that the set of all traces over the LTS is given by the regular expression $((20ct?\ coffee!) \cup (10ct?\ tea!))^*$ [Sud97]. Finally, $reach_{\mathcal{A}} = S_{\mathcal{A}}$, since all states are reachable from the start state. (In fact, they are reachable from every state.)

2.3 Test cases for LTSs

As mentioned before, many mission-critical systems need to be thoroughly tested. In case such a system has been modeled by an input-output labeled transition system, we can also formally define its test cases.

Just as has been done in [BBS06], we use a test case model based on `ioco` testing theory [Tre96]. We assume — as does `ioco` — that tests can only fail based on an output action. Also, we require tests to be *fail fast*, meaning they stop directly after observing a failure.

Basically, for each state a test case visits, it chooses to either perform an input action $a?$ or observe which output action $b!$ the system provides.

Definition 2.13. *A test case t for an LTS $\mathcal{A}$ is a prefix-closed, finite subset of $L^*_{\mathcal{A}}$, such that*

- *if $\sigma a? \in t$, then $\forall b \in L_{\mathcal{A}} : b \neq a? \rightarrow \sigma b \notin t$*
- *if $\sigma a! \in t$, then $\forall b! \in L^O_{\mathcal{A}} : \sigma b! \in t$*
- *if $\sigma \notin traces_{\mathcal{A}}$, then $\forall \sigma' \in L^+_{\mathcal{A}} : \sigma\sigma' \notin t$*

A test suite T is a tuple of test cases, denoted by $(t_1, \ldots, t_n)$.

Several facts can be observed from this definition. First of all, a test case does not consist of a *single* trace, but of a *set* of traces. Because of the prefix-closure, we require that if a certain trace is contained in a test case t, also all its prefixes are (Definition 2.3).

Furthermore, if a certain trace σ is augmented by an *input action*, no other trace equal to σ augmented by one action is present in the test case. On the other hand, if σ is augmented by an *output action*, then also all traces obtained by augmenting σ with the other output actions are included in the test case. Finally, if a certain trace σ is not present in $\mathcal{A}$, then no trace of which σ is a proper prefix can be present in a test case.

The next definitions define the *executions* and *inner traces* of a test case. An execution of a test case is simply a single trace ending in a leaf of the test case tree; a situation where no further transitions are specified. This exactly corresponds to where we end up when a test case is executed in practice.

Inner traces are exactly the opposite; they lead to a state in which there are still outgoing transitions defined by the test case.

Definition 2.14. *Let t be a test case for an LTS $\mathcal{A}$, then an* execution *of t is a trace $\sigma \in t$, such that σ is maximal in t. The set of all executions of t is denoted by $exec_t$, and formally defined by $exec_t = \{\sigma \in t \mid \nexists \sigma' \in t : \sigma \sqsubset \sigma'\}$. A* correct execution *of t is an execution $\sigma \in exec_t \cap traces_{\mathcal{A}}$, and an* erroneous execution *of t is an execution $\sigma \in exec_t \setminus traces_{\mathcal{A}}$. The set of all erroneous executions of t is denoted by err_t.*

Definition 2.15. *Let t be a test case, then an* inner trace *of t is a trace σ such that σ is* not *maximal in t. The set of all inner traces of t is denoted by $inner_t$ and formally defined by $inner_t = \{\sigma \in t \mid \exists \sigma' \in t : \sigma \sqsubset \sigma'\}$.*

Finally, a verdict function is defined, assigning one of the verdicts *pass*, *fail* and *cont* (short for *continue*) to each trace in t. Test case executions corresponding to erroneous behaviour receive the verdict *fail*, while executions corresponding to correct behaviour receive the verdict *pass*. All inner traces receive the verdict *cont*.

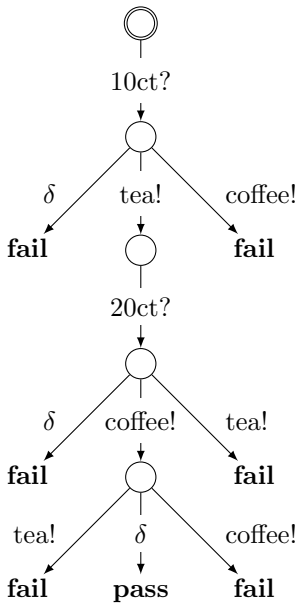


Figure 2.2: A test case for $\mathcal{A}$

Definition 2.16. *Let t be a test case for an LTS $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$, then the* verdict function *of t is the function $v_t : t \to \{pass, fail, cont\}$, defined by*

$$v_t(\sigma) = \begin{cases} pass & , if\ \sigma \in exec_t \cap traces_{\mathcal{A}} \\ fail & , if\ \sigma \in exec_t \setminus traces_{\mathcal{A}} \\ cont & , otherwise \end{cases}$$

Example 2.17. Figure 2.2 visually shows a possible test case for the LTS $\mathcal{A}$ of a coffee machine, introduced in Example 2.10. Observe how each time we choose between either one input action, or all the output actions (including the quiescence action δ). Furthermore, we added the verdict **fail** to all states resulting from a trace σ for which $v_t(\sigma) = fail$ (corresponding to erroneous executions), and the verdict **pass** to all states resulting from a trace σ for which $v_t(\sigma) = pass$ (corresponding to correct executions).

We can obtain the formal test case t by taking all the possible traces in this figure (including traces not ending in a verdict, to comply to the required prefix-closure). The test case is equal to the trace set $T = \{\epsilon, 10ct?, 10ct?\ \delta, 10ct?\ coffee!, 10ct?\ tea!, 10ct?\ tea!\ 20ct?, 10ct?\ tea!\ 20ct?\ \delta, 10ct?\ tea!\ 20ct?\ tea!, 10ct?\ tea!\ 20ct?\ coffee!, 10ct?\ tea!\ 20ct?\ coffee!\ \delta, 10ct?\ tea!\ 20ct?\ coffee!\ tea!, 10ct?\ tea!\ 20ct?\ coffee!\ coffee!\}$.

Seven executions can be identified ($10ct?\ \delta$, $10c?\ tea!\ 20ct?\ coffee!\ \delta$, and so on), six of which are erroneous.

2.4 Weighted fault models

To describe the seriousness of faults in an implementation of a system, it is useful to give not only the correct traces of the system, but also the severity of the erroneous ones. For this purpose, [BBS06] introduces the concept of a

weighted fault model.

A weighted fault model is independent of an input-output labeled transition system, since its definition is based on just an alphabet of actions L. For each possible trace over L, it defines its error weight. An error weight of 0 defines a correct trace. The higher the error weight, the worse we consider the presence of the erroneous trace in a particular implementation.

Definition 2.18. *Let L be a finite alphabet of actions, then a weighted fault model (WFM) over L is a function $f : L^* \to \mathbb{R}^{\geq 0}$, such that*

$$0 < \sum_{\sigma \in L^*} f(\sigma) < \infty$$

Based on this definition we can observe that a WFM should always contain at least one erroneous trace. Furthermore, the sum of all error weights should not be infinite. The first restriction makes sure that the relative potential coverage measure defined later on in this section is properly defined, and the second is necessary because otherwise any measure relative to the total error weight would yield a value of 0.

2.4.1 Coverage measures based on weighted fault models

Having defined weighted fault models, we can define coverage measures. These coverage measures indicate how many of the possible faults a certain trace set (or set of trace sets) can detect. We first define the absolute potential coverage of a set of traces t over a WFM f, which simply accumulates the error weights of all traces in t with respect to f. By looking at the total error weight with respect to f of all the traces over its alphabet, we can easily define the relative potential coverage of a trace set over a WFM as the fraction of the total error weight that is potentially covered by it.

The absolute and relative potential coverage of trace sets are extended in a natural way to the absolute and relative potential coverage of test suites (sets of trace sets) by first taking the union of their elements.

Definition 2.19. *Let $f : L^* \to \mathbb{R}^{\geq 0}$ be a WFM over some alphabet L, $t \subseteq L^*$ a set of traces, and $T \subseteq \mathscr{P}(L^*)$ a collection of these kind of sets, then we define*

- *absPotCov(t,f)* $= \sum_{\sigma \in t} f(\sigma)$
- *absPotCov(T,f)* $= absPotCov(\bigcup_{t \in T} t, f)$
- *totCov(f)* $= absPotCov(L^*, f)$
- *relPotCov(t,f)* $= \dfrac{absPotCov(t, f)}{totCov(f)}$
- *relPotCov(T,f)* $= \dfrac{absPotCov(T, f)}{totCov(f)}$

A test suite T is potentially complete *with respect to a WFM f if and only if $relPotCov(T, f) = 1$.*

2.4.2 Consistency of WFMs with LTSs

Although weighted fault models are defined independent of input-output labeled transition systems, they are meant to specify the desired *and* undesired behaviour of such systems. Since a test case or test suite is actually just a set of traces (with some extra properties), we can calculate their absolute and relative potential coverage with respect to a certain WFM based on Definition 2.19. That way, we obtain a measure for the quality of test cases and test suites.

However, not every WFM f is *consistent* with an input-output labeled transition system $\mathcal{A}$. For example, if some trace σ is in $\mathcal{A}$, then our interpretation requires that $f(\sigma) = 0$. Moreover, as mentioned in Section 2.3, we do not allow failures directly after an input action, and because of the *fail fast* property traces continuing after a failure are *not* considered erroneous. The next definition formally defines consistent weighted fault models based on these criteria.

Definition 2.20. *Let $\mathcal{A} = \langle S, s^0, L_1, \Delta \rangle$ be an LTS and $f : L_2^* \to \mathbb{R}^{\geq 0}$ be a WFM. Then, f is* consistent *with $\mathcal{A}$ if and only if they concern the same alphabet ($L_1 = L_2$) and for all $\sigma \in L_1^*$,*

- *if $\sigma \in traces_{\mathcal{A}}$, then $f(\sigma) = 0$*
- *$\forall a? \in L_1^I : f(\sigma a?) = 0$*
- *if $f(\sigma) > 0$ then for all $\sigma' \in L_1^+$ we have $f(\sigma\sigma') = 0$*

Example 2.21. Looking again at the LTS of Figure 2.1(b) (which we will refer to by $\mathcal{A}$ in this example), we will give a rough overview of how a possible WFM consistent with it would look like. We start by considering all traces that should have an error weight of zero, in order to comply to the consistency constraint.

First of all, for all traces σ in $\mathcal{A}$, $f(\sigma) = 0$. So, $f(\epsilon) = f(\delta) = f(10ct?) = f(10ct?\ tea!) = f(20ct?) = \cdots = 0$. Furthermore, for all traces σ *not* in $\mathcal{A}$ ending in an input action, $f(\sigma) = 0$, so also $f(10ct?\ 10ct?) = f(10ct?\ 20ct?) = f(20ct?\ 10ct?) = \cdots = 0$. Finally, for all traces σ *not* in $\mathcal{A}$ ending in an output action, but with a proper prefix that is also erroneous, $f(\sigma) = 0$, so $f(tea!\ tea!) = f(10ct?\ \delta\ \delta) = \cdots = 0$.

We are left with are all traces not in $\mathcal{A}$, ending in an output action and not having a proper prefix that is also erroneous. These traces may also be assigned an error weight of zero (as long as at least one of them is positive), but it is more logical to assign positive numbers. After all, these are the traces in a test case that would end up at a **fail** verdict. Using some of these traces as an example, f could be given a by:

$f(10ct?\ \delta) = 5$ (not producing anything after inserting a 10 cent coin)
$f(10ct?\ coffee!) = 3$ (producing coffee after inserting a 10 cent coin)
$f(10ct?\ tea!\ 10ct?\ \delta) = 4.9$ (not producing anything after a 10 cent coin)
etc...

The reason for choosing 4.9 instead of 5 as the error weight for the third trace is that a failure later on is considered less severe than the same failure earlier. Moreover, if we would not do this, we would get a WFM with an infinite error weight, which conflicts with the definition.

2.5 Fault automata

As Example 2.21 showed, it is quite some work to properly define a weighted fault model for an input-output labeled transition system. Even worse, since practically all LTSs contain an infinite number of traces, one has to define an infinite number of error weights. Although this could be accomplished by formulae in some cases, for more complicated systems it might be very difficult if not impossible. Therefore, [BBS06] introduces *fault automata*, which can be used to specify WFMs in a more manageable format.

In fact, a fault automaton is nothing more than an LTS $\mathcal{A}$ and a function r, specifying for each state the severity of producing the unexpected output actions.

Definition 2.22. *A fault automaton (FA) $\mathcal{F}$ is a pair $\langle \mathcal{A}, r \rangle$, where $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ is an LTS, and $r : S \times L^O \to \mathbb{R}^{\geq 0}$. We require that $r(s, a!) = 0$ if $\exists s' \in S : (s, a!, s') \in \Delta$.*

Notice that r is only defined over $S \times L^O$ and not over $S \times L$, since errors can only occur after an output action.

All concepts and notations defined for LTSs will also be used for fault automata, abstracting from the fact that a fault automaton *contains* an LTS instead of being one. For example, with a trace over a fault automaton $\mathcal{F}$ we will mean a trace over its LTS.

Definition 2.23. *Let $\mathcal{F} = \langle \mathcal{A}, r \rangle$ be a fault automaton, then $\bar{r} : S_{\mathcal{A}} \to \mathbb{R}^{\geq 0}$ assigns the accumulated weight of all erroneous outputs in a state to that state. Formally,*

$$\bar{r}(s) = \sum_{a \in L^O_{\mathcal{A}}(s)} r(s, a)$$

Example 2.24. Figure 2.3 shows a fault automaton for the LTS defined in Example 2.10. It specifies that producing coffee when no money has been inserted is considered to be of severity 9. If tea is produced, this is a bit less severe, since tea is cheaper than coffee.

If after inserting a 10 cent coin nothing is provided, this is defined to be of severity 5. If coffee is provided this is less severe, since the customer receives at least something, but of course it is still an error.

Formally, this fault automaton is defined as $\mathcal{F} = \langle \mathcal{A}, r \rangle$, with $\mathcal{A}$ the LTS given before, and r fully defined in Table 2.1.

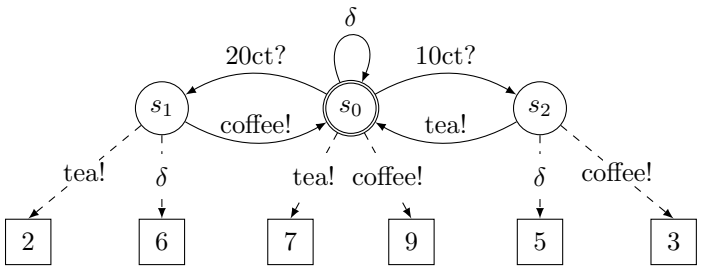


Figure 2.3: A fault automaton for $\mathcal{A}$

$r(s_0, \delta) = 0$	$r(s_0, tea!) = 7$	$r(s_0, coffee!) = 9$
$r(s_1, coffee!) = 0$	$r(s_1, tea!) = 2$	$r(s_1, \delta!) = 6$
$r(s_2, tea!) = 0$	$r(s_2, coffee!) = 3$	$r(s_2, \delta) = 5$

Table 2.1: The error weight function for $\mathcal{F}$

We immediately obtain $\bar{r}(s_0) = 7 + 9 = 16$, $\bar{r}(s_1) = 2 + 6 = 8$, and $\bar{r}(s_2) = 5 + 3 = 8$.

2.6 From FA to WFM

In Section 2.4 we explained how a weighted fault model describes the correct *and* erroneous behaviour of a system. Then, in Section 2.5 we defined fault automata as a syntactic format for specifying such a WFM. Since we defined the absolute and relative potential coverage of a test suite in terms of WFMs, it is desirable to construct a WFM f from an FA $\mathcal{F}$. Intuitively, we would like that for each trace σ ending in some state s, an erroneous trace $\sigma a!$ is assigned the error weight $r(s, a!)$. Since infinitely many traces may end up in s, however, this could have the effect that $totCov(f) = \infty$.

To construct a WFM based on an FA such that the total coverage is finite, two methods are proposed in [BBS06]. We will first consider finite depth weighted fault models, giving a positive error weight only to traces with a length smaller than some constant. Then, we will discuss discounted weighted fault models, decreasing the weight of traces based on their length.

2.6.1 Finite depth weighted fault models

A finite depth weighted fault model only assigns a positive error weight to traces σ for which $|\sigma| \leq k$. Since we require the alphabet of a WFM to be finite, this restriction results in a finite number of traces with a positive error weight. That way, the accumulated error weight remains finite, even though the number of traces itself is infinite.

Definition 2.25. *Let $\mathcal{F}$ be a fault automaton and $k \in \mathbb{N}$, then the function $f^k_{\mathcal{F}} : L^* \to \mathbb{R}^{\geq 0}$ is defined by*

$$f^k_{\mathcal{F}}(\epsilon) = 0$$

$$f^k_{\mathcal{F}}(\sigma a) = \begin{cases} r(s,a) & \text{if } |\sigma| < k \wedge a \in L^O \wedge \\ & \quad \exists \pi \in paths_{\mathcal{F}} : trace(\pi) = \sigma \wedge last(\pi) = s \\ 0 & \text{otherwise} \end{cases}$$

Since LTSs were defined to be deterministic, there can only be one path in L associated with each trace σ over L. Therefore, the function $f^k_{\mathcal{F}}$ is uniquely defined.

Proposition 2.26. *Let $\mathcal{F}$ be a fault automaton and $k \in \mathbb{N}$. If there exists at least one state s reachable in $k-1$ steps for which $\bar{r}(s) > 0$, then $f^k_{\mathcal{F}}$ is a WFM consistent with the LTS of $\mathcal{F}$.*

For a proof, we refer to [Bri07].

2.6.2 Discounted weighted fault models

While a finite depth WFM considers only traces limited in length by some constant, a discounted WFM considers all traces. However, it reduces the error weight of each trace based on the trace length. This construction is supported by the assumption that failures in the near future are worse than failures in the far future. Of course, discounting has to be applied in such a way that the accumulated weight of all traces is less than ∞.

The basic idea is to introduce a function $\alpha : S \times L \times S \to \mathbb{R}^{\geq 0}$ for each LTS $\langle S, s^0, L, \Delta \rangle$, assigning a discount factor to each transition. Then, the trace $\sigma = a_1 a_2 \dots a_k$, belonging to the path $s_0 a_1 s_1 a_2 s_2 \dots s_{k-1} a_k s_k$, is discounted by $\alpha(s_0, a_1, s_1)\alpha(s_1, a_2, s_2) \cdots \alpha(s_{k-1}, a_k, s_k)$.

We need the following definition for a restriction on α, making sure the total coverage will not be infinite.

Definition 2.27. *Let $\mathcal{F}$ be an FA, then $Inf_{\mathcal{F}} \subseteq S_{\mathcal{F}}$ is the set of all states with at least one outgoing infinite path. Formally, $Inf_{\mathcal{F}} = \{s \in S \mid \exists \pi \in paths_{\mathcal{F}[s]} : |\pi| > |S|\}$.*

The formal part of Definition 2.27 corresponds to the intuition of a infinite path, since a trace visiting more states than the total number of states must contain at least one cycle. This cycle can be repeated infinitely many times, obtaining an infinite path.

Now we can precisely define discount functions and the way to apply them on paths.

Definition 2.28. *Let $\mathcal{F}$ be an FA, then a* discount function *for $\mathcal{F}$ is a function $\alpha : S_{\mathcal{F}} \times L_{\mathcal{F}} \times S_{\mathcal{F}} \to \mathbb{R}^{\geq 0}$, such that*

- $\forall s, s' \in S_{\mathcal{F}}, a \in L_{\mathcal{F}} : \alpha(s, a, s') = 0 \Leftrightarrow (s, a, s') \notin \Delta_{\mathcal{F}}$
- $\forall s \in S_{\mathcal{F}} : \sum_{a \in L, s' \in Inf_{\mathcal{F}}} \alpha(s, a, s') < 1.$

For a detailed explanation of the second restriction, which makes sure that the total coverage will not be infinite, see [BBS06].

Definition 2.29. *Let α be a discount function for $\mathcal{F}$ and let $\pi = s_0 a_1 \dots a_n s_n$ be a path in $\mathcal{F}$, then $\alpha(\pi) = \prod_{i=1}^{n} \alpha(s_{i-1}, a_i, s_i)$.*

Using all the above definitions, we can define a weighted fault model based on discounting.

Definition 2.30. *Let $\mathcal{F}$ be an FA and α as discount function, then the function $f^{\alpha}_{\mathcal{F}} : L^* \to \mathbb{R}^{\geq 0}$ is defined by*

$f^{\alpha}_{\mathcal{F}}(\epsilon) = 0$

$$f^{\alpha}_{\mathcal{F}}(\sigma a) = \begin{cases} \alpha(\pi) \cdot r(s, a) & \text{if } a \in L^O \wedge \\ & \quad \exists \pi \in paths_{\mathcal{F}} : trace(\pi) = \sigma \wedge last(\pi) = s \\ 0 & \text{otherwise} \end{cases}$$

Because of determinism, there is at most one path π corresponding to σ. Therefore, the function $f^{\alpha}_{\mathcal{F}}$ is uniquely defined.

Proposition 2.31. *Let $\mathcal{F}$ be a fault automaton and α a discount function for $\mathcal{F}$. Then, if there exists at least one state s that is reachable for which $\bar{r}(s) > 0$, $f^{\alpha}_{\mathcal{F}}$ is a WFM consistent with $\mathcal{F}$.*

For a proof, we again refer to [Bri07].

Chapter 3

Nondeterministic fault automata

In this chapter, we extend the framework on potential coverage from [BBS06] to nondeterministic specifications. The reason for this is that many systems are modeled, or can be modeled much easier, using nondeterministic LTSs. Since it is well-known from literature that every nondeterministic LTS has a trace equivalent deterministic LTS, the theory of Chapter 2 can also be applied to these kinds of systems.

We first construct nondeterministic FAs based on nondeterministic LTSs, still using an error weight function similar to the one used before. However, the interpretation changes slightly, since there might be several different error weights assigned to the same trace. To enable the use of all definitions and propositions of the existing test coverage framework, our aim is to transform such nondeterministic FAs into deterministic FAs. This transformation is performed in two parts.

First, the underlying LTSs have to be determinised. The familiar subset construction can be applied, but a difficulty arises concerning the interplay with the special quiescence action δ. We show that removing nondeterminism first and adding quiescence afterwards results in nonequivalent systems. On the other hand, adding quiescence before determinising results in systems that do not comply to the definition of quiescence by `ioco` theory. We solve this difficulty by extending the definition of quiescence, integrating it with the concept of nondeterminism.

Second, the error weight function has to be adapted. The error weight of an erroneous output in the determinised FA is defined as the *lowest* error weight assigned to it by the nondeterministic FA.

Organisation of this chapter
Section 3.1 first defines nondeterministic LTSs, and describes the subset construction. Then, FAs based on nondeterministic LTSs and the way to determinise them are described in Section 3.2. Finally, Section 3.3 discusses the problem considering quiescence, and its solution. An algorithm for applying the transformation, including a detailed example and a discussion on its complexity, can be found in Appendix A.

3.1 Nondeterministic LTSs

The difference between nondeterministic and deterministic LTSs is that a state s and an action a do not uniquely identify the next state of the system anymore, since there might be several a-transitions from s to different target states. Therefore, an observer just seeing the external behaviour of a system might not be able to know precisely in which state the system is at a certain point during execution.

Note that, for simplicity, we ignore the possible existence of τ-transitions. However, the transformations needed to incorporate this unobservable behaviour are already known from literature [Sud97]. One could substitute these methods for ours without any consequences.

Definition 3.1. *A nondeterministic input-output labeled transition system $\mathcal{N}$ is given by a tuple $\langle S, s^0, L, \Delta \rangle$, where*

- *S is a finite set of states*
- *s^0 is the initial state*
- *L is a finite set of actions, partitioned into a set L^I of* input actions *and a set L^O of* output actions *($L = L^I \cup L^O$ and $L^I \cap L^O = \varnothing$)*
- *$\Delta \subseteq S \times L \times S$ is the transition relation*

Note that Definition 3.1 only differs from Definition 2.8 on its fourth constraint. For nondeterministic LTSs we drop the requirement that $s' = s''$ if $(s, a, s') \in \Delta$ and $(s, a, s'') \in \Delta$.

All definitions about LTSs given in Chapter 2, including paths and traces, are still valid for nondeterministic LTSs. However, for nondeterministic LTS there can now be multiple paths associated with one single trace. Following `ioco` theory [Tre96] and just applying Definition 2.20, we consider a trace over a nondeterministic LTS correct if there is at least one path corresponding to it.

3.1.1 Determinising LTSs

It is well-known that each nondeterministic LTS $\mathcal{N}$ is equivalent to a deterministic LTS $\mathcal{D}_\mathcal{N}$, such that both have exactly the same traces. For this purpose we can use the so-called *subset construction*, also called *powerset construction*, which is described in most elementary textbooks on automaton theory [Sud97]. The following definition and proposition describe the transformation and its properties.

Definition 3.2. *Let $\mathcal{N} = \langle S, s^0, L, \Delta_\mathcal{N} \rangle$ be a nondeterministic LTS. Then $\mathcal{D}_\mathcal{N}$ is an LTS, defined as $\mathcal{D}_\mathcal{N} = \langle \mathscr{P}(S) \setminus \varnothing, \{s^0\}, L, \Delta_\mathcal{A} \rangle$, with*

$$\Delta_\mathcal{A} = \{(s, a, t) \mid s \in \mathscr{P}(S) \wedge a \in L \wedge t = \{t' \in S \mid \exists s' \in s : (s', a, t') \in \Delta_\mathcal{N}\}\}$$

Proposition 3.3. *Let $\mathcal{N} = \langle S, s^0, L, \Delta_\mathcal{N} \rangle$ be a nondeterministic LTS. Then $\mathcal{D}_\mathcal{N}$ is deterministic and trace equivalent to $\mathcal{N}$.*

The idea behind the automaton $\mathcal{D}_\mathcal{N}$ is as follows. Its states (called *superstates*) are sets of states of $\mathcal{N}$. A transition from a superstate s to a superstate t by an action a exists if for all states in t a state in s exists that can reach t by an a transition. Moreover, no superset of t satisfying this condition should

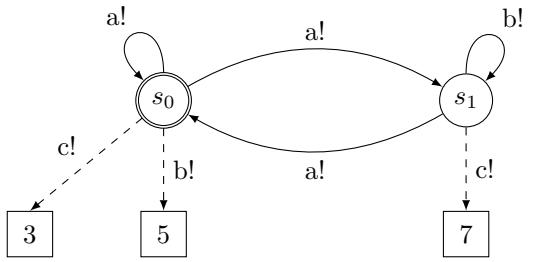


Figure 3.1: A fault automaton based on a nondeterministic LTS

exist. In this way the states of a superstate exactly represent the states that $\mathcal{N}$ can possibly be in based on the transitions that have occurred thus far.

Initially, the only possible state $\mathcal{N}$ can be in is s^0, so the initial state of $\mathcal{D}_\mathcal{N}$ is $\{s^0\}$. Furthermore, $\mathcal{N}$ and $\mathcal{D}_\mathcal{N}$ obviously have to have the same alphabet to be trace equivalent.

3.2 Nondeterministic FAs

In Definition 2.22 we defined a fault automaton as a pair $\langle \mathcal{A}, r \rangle$, where $\mathcal{A}$ is a deterministic LTS, and r is a function assigning error weights to the occurrence of output actions.

For a *nondeterministic* FA we drop the assumption made previously that $\mathcal{A}$ is a deterministic LTS. Since in both deterministic and nondeterministic LTSs an output action *a!* is erroneous in case there is *no a!* transition, the existing semantics and interpretation for the function r can be preserved.

Example 3.4. Figure 3.1 shows an example of a fault automaton based on a nondeterministic LTS. This FA is nondeterministic, because the occurrence of an *a!* transition in state s_0 can either result in a move to s_1, or a self-loop to s_0. This choice determines what can happen next. If we enter s_1, a *b!* can be observed. However, if we remain in the initial state, this output is specified to be incorrect. Furthermore, the error weight assigned to the occurrence of *c!* depends on the transition that is taken.

3.2.1 Determinising FAs

To determinise an FA, both the LTS and the error weight function have to be dealt with. For determinising the LTS, we can simply use the subset construction described in Section 3.1.1. By applying this construction, a new LTS with a different structure is obtained. Therefore, a new error weight function has to be constructed as well.

Nondeterministic FAs may assign different error weights to the same trace. It could even be the case that an output is considered correct when following one path, but considered erroneous when following another path corresponding to the same trace.

These situations both occur in the FA shown in Figure 3.1. Starting in state s_0 the trace *a! b!* either has an error weight of 5, or is considered correct. As the trace *is* present, it will not be considered erroneous, and the *a! b!* failure should

not be visible anymore in the determinised FA. To accomplish this, an output action *a!* from some superstate s of a determinised LTS is only considered erroneous in case none of the states s contains can perform *a!*.

In case an output action indeed *is* erroneous from a certain superstate, an error weight should be given. However, since the original FA provides an error weight for this output action for every of the corresponding states, there might be different values. In Figure 3.1 the error weight of *a! c!* is either 3 or 7, depending on the transition that was taken by the *a!* action. We have chosen to use the minimum value of these error weights for the determinised FA. This is in line with the interpretation that a trace is correct when there is at least one path justifying it. In this case the occurrence of the trace *a! c!* therefore has an error weight of 3.

This results in the following definition, describing for each nondeterministic fault automaton the deterministic fault automaton we considered equivalent.

Definition 3.5. *Let $\mathcal{F} = \langle \mathcal{N}, r \rangle$ be a nondeterministic fault automaton, based on the nondeterministic LTS $\mathcal{N} = \langle S, s^0, L, \Delta_{\mathcal{N}} \rangle$. Then, $\mathcal{D}_{\mathcal{F}} = \langle \mathcal{D}_{\mathcal{N}}, r' \rangle$ is its corresponding deterministic fault automaton. The error weight function r' is given by*

$$r'(s,a) = \begin{cases} 0 & , \text{ if } \exists s' \in S_{\mathcal{A}} : (s,a,s') \in \Delta_{\mathcal{A}} \\ \min_{s' \in s} r(s',a) & , \text{ otherwise} \end{cases} \tag{3.1}$$

For an algorithm applying the subset construction while incorporating the error weight function, see Appendix A.

3.3 Dealing with quiescence

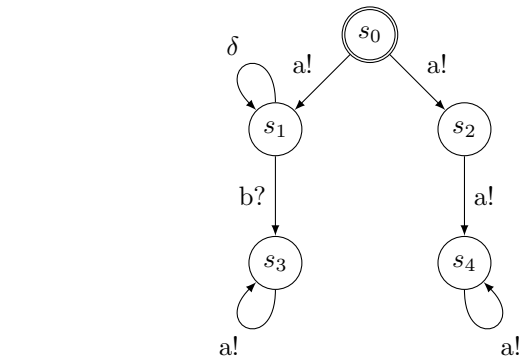
In the previous section we discussed determinising LTSs, mentioning nothing about the special quiescence action δ. Adding quiescence is, however, vital to support testing based on fault automata, since it enables us to give an error weight to a state being erroneously quiescent. Therefore, we want our theory to allow the nondeterministic fault automata to incorporate quiescence.

As indicated in [BBS06], quiescence is not preserved under determinisation, so it is not possible to just consider it as one of the output actions. [BBS06] recommends to remove nondeterminism first and then add quiescence in the conventional manner. Adding quiescence afterwards, however, results in an LTS representing different behaviour than the original nondeterministic LTS.

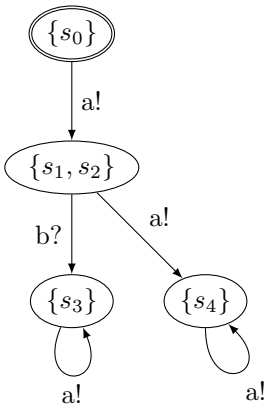
Figure 3.2 illustrates the problem at hand. Suppose we have the LTS depicted in Figure 3.2(a). After it does an *a!*, it can either enter the quiescent state s_1, or the state s_2 from which another *a!* can occur.

If we add quiescence *after* determinisation we obtain the LTS depicted in Figure 3.2(b). It shows that after an *a!*, either an *a!* or a *b?* transition can take place. Because of this output action *a!*, the composed state $\{s_1, s_2\}$ *does not* allow quiescence. However, the nondeterministic automaton we started with *does* allow quiescence after the first *a!*. Therefore, adding quiescence after determinisation does not preserve the behaviour of a nondeterministic LTS.

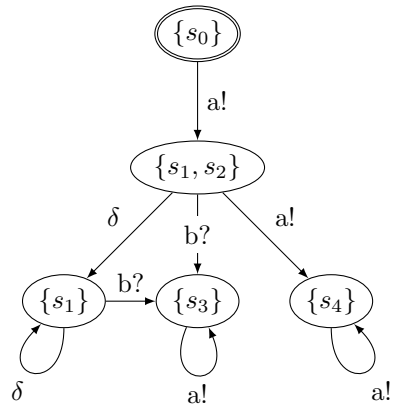
According to `ioco` theory, however, a δ transition can only be added to a state with no outgoing output actions. Furthermore, it is required to be a self-loop. This makes it impossible to specify that a state from which an output action can occur is also allowed to be quiescent.



(a) An LTS



(b) Quiescence afterwards



(c) Quiescence beforehand

Figure 3.2: Handling quiescence when removing non-determinism

As a solution, we propose to extend the meaning of δ, making it possible to transform a nondeterministic LTS into a deterministic one without loosing any information or changing its meaning.

Definition 3.6. *Let $\mathcal{A}$ be an LTS, and $s, s' \in S_{\mathcal{A}}$. A transition (s, δ, s') may be added if $\Delta^{O}_{\mathcal{A}}(s') = \varnothing$. It signifies that when $\mathcal{A}$ is in s, it is allowed to do nothing until an input arrives, and that it continues waiting in s'.*

To explain the restriction put on δ transitions, observe the LTS in Figure 3.3. In this case the target state of the δ transitions *does* have an outgoing output transition. This means that a trace consisting of a δ followed by an *a!* can occur, which does not seem sensible. After all, a δ transition means that the system has to wait for input, before it can do an output action again.

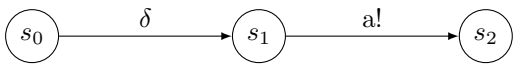


Figure 3.3: Quiescence wrongly applied

Now that quiescence is defined in such a way that it is possible to have δ transitions with different source and target states, it is possible to add quiescence *prior* to determinisation. The result of determinising the LTS of Figure 3.2(a) is shown in Figure 3.2(c). Now it is clear that after an *a!* is observed, both the output of another *a! and* quiescence are correct behaviour. After observing quiescence, the *a!* cannot be observed anymore before an input action is given, as required by Definition 3.6.

Using the new definition of quiescence, the suspension traces (traces possibly including one or more δ) of the determinised LTS are equal to the suspension traces of the original LTS. After all, δ can now be added prior to determinisation and will therefore just be considered as an action. Since it is known from theory that determinisation yields trace equivalence, adding quiescence beforehand yields suspension trace equivalence.

It is immediate that the suspension traces were not preserved when using the old definition and adding quiescence afterwards, since Figure 3.2 shows a counter example.

Chapter 4

From potential to actual coverage

Having described the framework on potential coverage of [BBS06], an important limitation can be observed: it only describes how many faults are *potentially detected*. Since a finite number of test executions might not be able to detect all erroneous traces [HT96], this only correctly describes the faults that are covered for an *infinite* number of executions.

This remainder of this thesis extends the theory on semantic test coverage by defining *actual coverage*: a notion that basically not only takes into account which faults are contained in a test case, but also how many will actually be covered during one or more test executions. Actual coverage will be evaluated given a sequence of executions, and predicted based on a test case or test suite and a probabilistic execution model of a system.

This chapter describes the notion of actual coverage in detail, explaining it first intuitively and then discussing the formal ingredients of our framework. We explain the main concepts that will be developed in the subsequent chapters (a probabilistic execution model, the evaluation of actual coverage and the predicting of actual coverage), giving a broad overview of the purpose and cohesion of these concepts.

Organisation of this chapter
First, Section 4.1 discusses the limitations of potential coverage. Then, the requirements of the coverage notion to be developed are explained in Section 4.2. The resulting notion, actual coverage, is then intuitively explained in Section 4.3. Finally, Section 4.4 introduces the formal ingredients that will be defined in the next chapters.

4.1 The limitation of potential coverage

As explained in Chapter 1, many different definitions of coverage can be found in literature on testing. The framework of [BBS06], discussed in Chapter 2, made an initial attempt to define coverage from a *semantic* point of view. It defines coverage as the number of *potential faults* that are *potentially detected*, weighted by the severity of each fault.

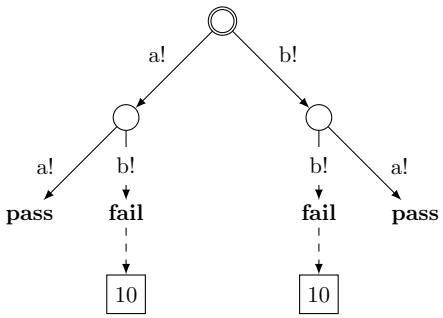


Figure 4.1: A test case t

There is an important limitation to the approach taken in [BBS06]: the fact that it only describes which faults are *potentially* covered. When a test case is executed just once, however, not all traces over it are actually traversed by the system (unless the test case does not branch). An immediate consequence is that during a single execution of a test case, not all faults that were *potentially* covered are *actually* covered. Since test case executions might overlap, every finite number of executions might fail to cover all faults. Therefore, the notion of potential coverage only correctly describes the faults that were covered for an *infinite* number of test case executions.

As an example, observe Figure 4.1. Although the coverage measure of [BBS06] would deem the absolute coverage of this test case to be 20, a single execution will only be able to detect at most one of the faults. This shows that potential coverage does not draw conclusions about what will happen when a test case or test suite is executed a *finite* number of times, or how many times it *should be* executed to obtain on average a certain coverage.

This chapter extends the framework of [BBS06], introducing a new notion of coverage that does deal with these issues.

4.2 Requirements for a new notion of coverage

The main requirement of our notion of coverage is that it improves potential coverage by taking into account which faults were *actually* shown present or absent during a certain execution or sequence of executions. Furthermore, we want to be able to *predict* how many faults will *actually* be covered during a certain number of executions. Therefore, our notion is called *actual coverage*.

Our definition of actual coverage has been directed by several subrequirements, listed below. For every requirement a motivation is provided. When applicable, we also discuss the technical implications for the framework to be developed. Note that the third requirement is partly implied by the fourth.

1. When the number of executions of a test case approaches infinity, its actual coverage should approach its potential coverage.

   Motivation:

   The potential coverage measure denotes the error weight of all faults that potentially can be detected by a test execution. Although in a

single execution not everything can actually be detected, for infinitely many executions this *will* be the case (assuming all faults are reachable).

2. The actual coverage of a sequence of test case or test suite executions E should be larger than or equal to the actual coverage of a sequence of executions $E' \subseteq E$.

   Motivation:

   By testing more thoroughly we can obviously only learn more about a system, therefore obtaining more coverage (or an equal amount, if we learned nothing new).

3. Correct executions might have a nonzero actual coverage value.

   Motivation:

   When no failures occur from any of the intermediate states of a correct execution, our confidence in the absence of faults increases. Therefore, observing a correct execution should yield a nonzero actual coverage value (unless only states were visited in which all behaviour is correct, naturally).

   In other words:

   The framework should consider a fault σa not only covered when it is observed, but also when an execution increases the confidence in its absence. This is the case when σ is observed, but the system provides an output action different from a.

4. Observing the same correct execution more often should increase actual coverage, with an amount depending on the probability with which potential failures on its path occur. It should depend on the error weights of the corresponding faults as well.

   Motivation:

   A fault σa that *is* present in a given implementation might nonetheless *not* be observed after every observed σ. For example, faults corresponding to threading might result in failures only sporadically. Therefore, a single correct execution does not exclude all faults on its path. Covering a fault more often should thus increase the total actual coverage value (unless it is certain that, in case it is present, it occurs every time its source state is visited). For faults that — in case they are present — result in failures only sporadically, more executions covering them are necessary to achieve the same certainty of absence than for faults that can be observed very often.

   In other words:

   The framework should incorporate estimations of failure occurrence probabilities, given that the corresponding faults are present. That way, it can define the actual coverage of a fault based on the number of correct executions showing its presence or absence and the probability of resulting in a failure in case it is present.

5. After an execution terminated by failure, later executions observing either the presence or absence of the corresponding fault should not influence its actual coverage anymore.

Motivation:
When a failure is observed, we know with certainty that the corresponding fault *is* present. Therefore, we consider it completely covered.

We will use the notion of actual coverage for two different purposes: to *evaluate* the obtained coverage after testing has completed, and to *predict* the actual coverage a test case will yield. This way, actual coverage can be applied to decide how useful a testing process has been, and to compare test cases or test suites in advance such that the best one can be selected before testing has started.

To be able to reason about the expected outcome of a test execution in advance, it is unavoidable to make assumptions on the behaviour of the system under test. This justifies our choice to include a thorough discussion on probabilities.

4.3 The intuition behind actual coverage

Actual coverage is defined for executions of test cases. It resembles the notion of potential coverage of [BBS06] in the sense that it accumulates the error weights of faults in a test case. However, a fault σa is only considered actually covered if an execution informed us about whether the fault is present or absent. When σa has been observed we know it is present, and when σb has been observed we increased our confidence in the absence of σa. Traces not starting with σ do not actually cover the fault σa.

Since the absence of a fault can in general not be determined with certainty, we take only a fraction of each error weight. This fraction is equal to the *coverage probability* of the corresponding fault, given an execution or sequence of executions.

The coverage probability of a fault indicates how certain we know whether it is present or absent. If a fault was observed, no doubts remain, so its coverage probability is 1. On the other hand, if a fault has not been observed, there might have been one or more executions that showed its absence. The coverage probability then describes how likely it is that the fault would have been observed in case it was present. After all, the higher this probability, the more certain we are of its absence.

We define the *actual coverage distribution* of a test case to be the actual coverage it yields. This is modeled as a random variable, since several executions might occur. Earlier work followed the same approach [HT96]. We provide formulae for the calculation of the expected actual coverage, that way predicting the quality of a test case.

4.4 The formal ingredients of actual coverage

Our framework on actual coverage consists of several components, based on the concepts of the framework on potential coverage. We introduce a probabilistic execution model and, based on this model, we provide the notion of actual coverage and the notion of an actual coverage distribution.

The fact that we need probabilistic information about the system is immediate from the purpose of actual coverage. As it has to predict which faults

will be covered, the behaviour of the system needs to be estimated. Because many modern systems are highly nondeterministic, this behaviour can only be specified by equipping it with probabilities.

- The *probabilistic execution model* consists of two functions: p^{br} and p^{cbr} (in which the superscripts stand for *branching* and *conditional branching*). The function p^{br} describes the expected probabilistic transition behaviour of a test case, by assigning probabilities to its branches. It is needed to estimate how often the state from which a potential failure can occur is visited during test executions. The function p^{cbr} describes the expected probabilities with which failures occur given that they are present. It is needed to estimate how certain we can be of the absence of a fault, after we have observed a given number of executions that did not fail. This certainty is expressed as the fault's *coverage probability* p^{cov}.

  The probabilistic execution model is defined over *test cases*, since this makes it finite. Moreover, the transition behaviour of a system depends on whether or not we choose to provide input actions; this is specified by a test case. However, we will also define *probabilistic fault automata* (PFAs) to specify the probabilities over a *system*. The reason for this is to provide an easy syntactic means to specify all relevant probabilities just once, instead of having to specify them again for every test case. A PFA is actually just an FA, augmented with generalised functions p^{br} and p^{cbr}. A transformation is provided to obtain the probabilistic execution model over given test cases automatically from such a PFA.

  As a possible extension we discount the error weights based on the probabilistic execution model, obtaining risk-based testing. The discounted error weights then describe the *importance of preventing the existence* of the corresponding faults, instead of the *severity of their occurrence*.

  The probabilistic execution model is described in Chapter 5.

- The *actual coverage* of a specific execution or sequence of executions is defined as the sum of all the error weights of all failures that can occur from any observing state visited during such an execution or sequence of executions, each multiplied by its coverage probability. Calculating actual coverage after testing has completed is well-suited to evaluate whether or not more testing is required.

  We show how to calculate how many executions covering a fault are needed to achieve a certain coverage probability.

  In the calculation of actual coverage we only need the function p^{cbr} of the probabilistic execution model. Therefore, a simplified version of the probabilistic execution model (omitting p^{br}) can be used if the framework is only applied to evaluate the actual coverage of given executions.

  The evaluation of actual coverage is described in Chapter 6.

- The *actual coverage distribution* of a test case, also computed based on the probabilistic execution model, is represented by a random variable describing the actual coverage obtained by executing a test case (parameterised by the number of executions). It can be used for test selection, as it predicts in advance what will happen during test executions. Clearly,

to calculate this, it is necessary to estimate how the system will behave during the executions. For this estimation the function p^{br} is used, so the complete probabilistic execution model is needed. However, we propose an approximation that uses a simplified version of the probabilistic execution model.

The most important property of the actual coverage distribution is its *expected value*. This gives an expectation of the actual coverage that will be achieved. We show that when the number of executions approaches infinity, the expected value of actual coverage approaches potential coverage. Besides the expectation of actual coverage we also investigate its variance. Unfortunately, it will turn out that the calculation of variance is not possible in polynomial time.

The *prediction* of actual coverage for test cases is described in Chapter 7.

In Chapter 8, all notions described above are generalised to also be used on test suites instead of only test cases.

4.5 An introductory example

To acquire a basic feeling about the applicability of actual coverage, we provide an example in which an important part of the theory of the subsequent chapters is applied informally.

Estimating the probabilistic behaviour
Consider the test case of Figure 4.1 again. As explained before, we describe the behaviour of the implementation under test by estimating its *probabilistic execution model*. It consists of the probability with which each branch is chosen (the *branching probabilities* p^{br}) and the probability of the occurrence of each failure given that the corresponding fault is present (the *conditional branching probabilities* p^{cbr}).

Let us first assume a flawless system. It provides an *a!* with probability 0.75, and a *b!* with probability 0.25. Incorporating these probabilities in the model, we obtain the test case depicted in Figure 4.2 on the left.

Also taking possible failures into account, we might for example estimate that the conditional branching probability of both faults is 0.75. Moreover, we

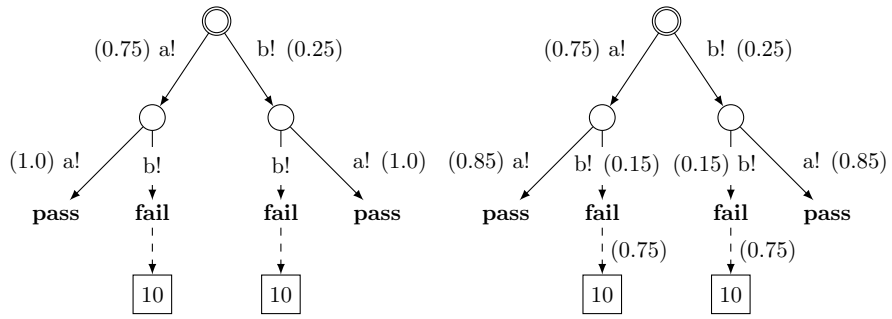


Figure 4.2: Probabilistic models for t

can estimate the probability that the faults are present (later called the *presence probabilities*), in this case for example 0.2. Multiplying these probabilities we obtain the unconditional probability of a *b!* after an *a!* or *b!*, resulting in 0.15. Note that this does not mean that the faults are known to be present and always occur fifteen times every one hundred executions. It does mean that if we would take ten random implementations and execute each ten times, we expect to see approximately fifteen failures.

Together, the branching probabilities p^{br} and the conditional branching probabilities p^{cbr} constitute the *probabilistic execution model*. Depicting this model visually, we obtain the test case shown in Figure 4.2 on the right.

Note that the conditional branching probabilities are included in the test case only as an easy way to represent the probabilistic behaviour in one figure. They can be interpreted as possible substitutions for the branching probabilities above them: the conditional branching probability 0.75 on the *a! b!* branch denotes that the probability of a *b!* after an *a!* is *either* 0.0 *or* 0.75. The branching probability 0.15 is the weighted average of these two probabilities, each weighted by the probability that it applies. As the presence probability was estimated to be 0.2, we observe that indeed $0.75 \cdot 0.2 = 0.15$.

Evaluating actual coverage

Using the probabilistic execution model obtained thus far, *actual coverage* can be evaluated. For simplicity we ignore relative actual coverage for the moment, and just focus on absolute actual coverage: *absCov*.

Assuming we have observed the execution σ = *a! b!*, we obtain an actual coverage of 10 (since one of the faults has been shown to be present). If we have observed the sequence of executions E = (*a! b!, b! b!*), we obtained an actual coverage of 20 (since both faults have been shown to be present).

If, however, we have observed the sequence of executions E = (*a! a!, a! a!*), we did not show the presence of any fault. We did increase our confidence in the absence of *a! b!*, although we still do not know whether it is indeed absent. The actual coverage is therefore just a fraction of its error weight. As indicated before, the fraction is called the *coverage probability* p^{cov} of *a! b!*. It is defined as the probability that the erroneous execution *a! b!* is observed at least once, given the number of times we observed *a!* and given the presence of *a! b!*.

In this case, we observed *a!* twice, and the probability of a *b!* after an *a!* is 0.75 in case it is present. Therefore, the probability of *not* seeing *a! b!* both times is $(1 - 0.75)^2$. This immediately yields that the coverage probability is $1 - (1 - 0.75)^2 = 0.9375$. Multiplying by the error weight, this gives an actual coverage of 9.375.

Predicting actual coverage

Instead of evaluating the actual coverage of a given execution or sequence of executions, we might also be interested in predicting the actual coverage a test case *will* obtain. As we saw before, executing a test case once or several times might result in different executions each time this 'experiment' is performed. Since each execution has its own actual coverage, the actual coverage a certain number of test case executions yields can be described by a probability distribution. This distribution is called an *actual coverage distribution*, denoted by the random variable $A_{t,f}$ (t referring to the test case, f to its weighted fault

Execution	p^{to}	*absCov*
a! b!	$0.75 \cdot 0.15 = 0.1125$	10
b! b!	$0.25 \cdot 0.15 = 0.0375$	10
a! a!	$0.75 \cdot 0.85 = 0.6375$	7.5
b! a!	$0.25 \cdot 0.85 = 0.2125$	7.5

Table 4.1: Probabilities of executions and their actual coverage

model).

For the test case of Figure 4.2, a single execution can yield four different traces. For each trace, the probability of its occurrence (later called its *trace occurrence probability* p^{to}) and its absolute actual coverage are shown in Table 4.1.

It is not difficult to see that the actual coverage distribution of the test case is therefore given by

$$\mathbb{P}[A_{t,f} = 10] = 0.1125 + 0.0375 = 0.15$$
$$\mathbb{P}[A_{t,f} = 7.5] = 0.6375 + 0.2125 = 0.85$$

From this we derive its expected value by

$$\mathbb{E}(A_{t,f}) = 10 \cdot 0.15 + 7.5 \cdot 0.85 = 7.875$$

Note that the number of possible executions rises exponentially with the number of test executions that will be performed, making the calculation method applied above infeasible for larger systems. However, we will present a derivation that can still be calculated in polynomial time.

Moreover, the methods used above are generalised to test suites.

Chapter 5

Probabilities in test case executions

If it is unknown how a system behaves, estimating how many faults one of its test cases will unveil is impossible. Therefore, several probabilities have to be known, estimated or measured.

This chapter discusses all the probabilities necessary for the coverage analyses in the next chapters. We will first describe the random experiment of executing a test case t, and the induced random variable X_t corresponding to the trace obtained this way. Then, the function p^{to} is defined, assigning to every trace the probability that it will occur during an execution. Also, we define the function p^{br}, assigning to each sequence of actions the probability distribution of the next action. We show that both p^{to} and the probability distribution function of X_t are derivable from p^{br}. Then, conditional branching probabilities are introduced, described by the function p^{cbr}. It specifies for each fault how often it results in a failure, given its presence.

We define the *probabilistic execution model* for each test case as a pair consisting of p^{br} and p^{cbr}, together describing all relevant probabilities. Probabilistic fault automata are introduced as a semantic way to specify such probabilistic execution models over automata, instead of over test cases. To assist the user of this framework in specifying probabilistic execution models, we describe in detail the process of obtaining the necessary probabilities.

We discuss an immediate application of the probabilities derived in this chapter: to transform error weights into error risks, based on the theory of risk-based testing [Aml00].

Organisation of this chapter
First, Section 5.1 and Section 5.2 briefly recall the theory on probability spaces, random variables and conditional probabilities. These concepts are used in Section 5.3 to describe the random variable X_t. Then, Section 5.4 defines the functions p^{to} and p^{br}, and proves that p^{br} is sufficient to derive p^{to} and the probability distribution function of X_t. Section 5.5 defines p^{cbr}, after which Section 5.6 formally defines the probabilistic execution model. Section 5.7 discusses probabilistic fault automata, and Section 5.8 describes the process of

obtaining the relevant probabilities. Section 5.9 concludes this chapter by discussing risk-based testing.

5.1 Probability spaces and random variables

A *random experiment* is an experiment whose outcome cannot be predicted in advance. Every time the experiment is performed (a *trial*) its outcome is one of a set of possible outcomes. This set is called its *sample space*. A *basic event* is an element of the sample space. An *event* is a statement about the outcome of a random experiment; formally it is a subset of the sample space. The probability of an event E is the probability that the outcome of the experiment is in E.

Formally, a random experiment is a pair (Ω, P), called a *probability space*[1]. In this pair, the element Ω is a set identifying the sample space. Every $\omega \in \Omega$ corresponds to a certain basic event in the physical world. The element P is a function from $\mathscr{P}(\Omega)$ to $[0, 1]$, assigning a probability to each event in $\mathscr{P}(\Omega)$. It is called the probability distribution over Ω, and should be such that the probability axioms are satisfied. This entails $P(\Omega) = 1$, and $P(E_1 \cup \ldots \cup E_n) = P(E_1) + \cdots + P(E_n)$ for any countable sequence of pairwise disjoint events $E_i \in \mathscr{P}(\Omega)$.

From a mathematical perspective, a probability space defines all relevant properties of a random experiment; we are only interested in which events are possible, and the probability of each event.

To avoid having to work directly with probability spaces, *random variables* are used. Often, a random variable X on a probability space (Ω, P) is a function from Ω to $\mathbb{R}$. We will, however, also use random variables with a different codomain (denoted by S_X). The probability distribution of a random variable X on a probability space (Ω, P) is given by $\mathbb{P}[X = x] = P(\{w \in \Omega \mid X(w) = x\})$. We extend this standard notation with $\mathbb{P}[\sigma \sqsubseteq X]$, denoting the probability that σ is a prefix of the valuation of X. Obviously, this definition only applies to random variables with a codomain consisting of traces.

Definition 5.1. *Let X be a random variable with a codomain S_X. If there is a set L such that $S_X \subseteq L^*$, then we define*

$$\mathbb{P}[\sigma \sqsubseteq X] = \sum_{\substack{\sigma' \in S_X \\ \sigma \sqsubseteq \sigma'}} \mathbb{P}[X = \sigma']$$

A random variable X on a sample space Ω is called *discrete* if its *value range*, the set $\{X(\omega) \mid \omega \in \Omega\}$, is finite or countably infinite. Since we only deal with finite sample spaces, all random variables in this thesis will be discrete.

For an extensive overview of probability theory we refer to textbooks such as [AL06] and [ADD99].

[1] In fact this is a simplification, since formally a probability space is a triple $(\Omega, \mathcal{F}, P)$, with $\mathcal{F} \subseteq \mathscr{P}(\Omega)$ a σ-field over Ω and P a function from $\mathcal{F}$ to [0..1]. $\mathcal{F}$ then defines the *event space*: all events for which a probability has to be specified. Since in our case all sample spaces are finite, $\mathcal{F}$ is always equal to the powerset of Ω. Hence, we omit $\mathcal{F}$ from the probability space. For more details on probability spaces we refer to [Hal50].

5.2 Conditional probabilities

In the remainder of this chapter, we often use the notion of *conditional probabilities*. These probabilities are defined as follows in literature.

Definition 5.2. *Let (Ω, P) be a probability space and $E_1, E_2 \in \mathscr{P}(\Omega)$ two events. Then, if $P(E_2) > 0$, the* conditional probability of E_1 given E_2 *is defined by*

$$P(E_1 \mid E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

If X, Y and two discrete random variables, the conditional probability distribution of X given $Y = y$ *is defined by*

$$\mathbb{P}[X = x \mid Y = y] = \frac{\mathbb{P}[X = x \wedge Y = y]}{\mathbb{P}[Y = y]}$$

The following lemma is known from literature and will be useful in many proofs.

Lemma 5.3. *Let (Ω, P) be a probability space and $E_1, E_2, E_3 \in \mathscr{P}(\Omega)$ three events. Then, if $P(E_2 \cap E_3) > 0$, the* conditional probability of E_1 given E_2 and E_3 *is defined by*

$$P(E_1 \mid E_2 \cap E_3) = \frac{P(E_1 \cap E_2 \mid E_3)}{P(E_2 \mid E_3)}$$

5.3 The test case execution experiment

Consider the random experiment of executing a test case t once, without any knowledge on fault presence. We define the random variable $X_{t,\mathcal{I}}$ as the trace that this experiment yields given an implementation $\mathcal{I}$.

Definition 5.4. *For every test case t, the random variable $X_{t,\mathcal{I}} : \Omega \to exec_t$ is the trace obtained when executing t once given an implementation $\mathcal{I}$, without any knowledge on fault presence.*

Conform Definition 2.14, the outcomes of $X_{t,\mathcal{I}}$ are also called *executions* of t. The subscript $\mathcal{I}$ will from now on be omitted, assuming an implicit implementation.

Note that we should formally define the underlying probability space (Ω, P). Recall that every $\omega \in \Omega$ corresponds to the occurrence of a basic event in the physical world, so in this case to the occurrence of a certain execution. The function P maps each of these events on the probability of its occurrence. The random variable X_t then maps every $\omega \in \Omega$ on the trace in $exec_t$ whose occurrence it represents. Therefore, $\mathbb{P}[X_t = \sigma]$ is the probability of the occurrence of the execution σ.

For this, and all other random variables and function that will be defined, the subscript t may be omitted in case it is clear from the context.

Example 5.5. Figure 5.1 shows a test case t with an estimated probability distribution of X_t. For example, in this case $\mathbb{P}[X_t = a?\ e!\ b?\ d!] = 0.0062$.

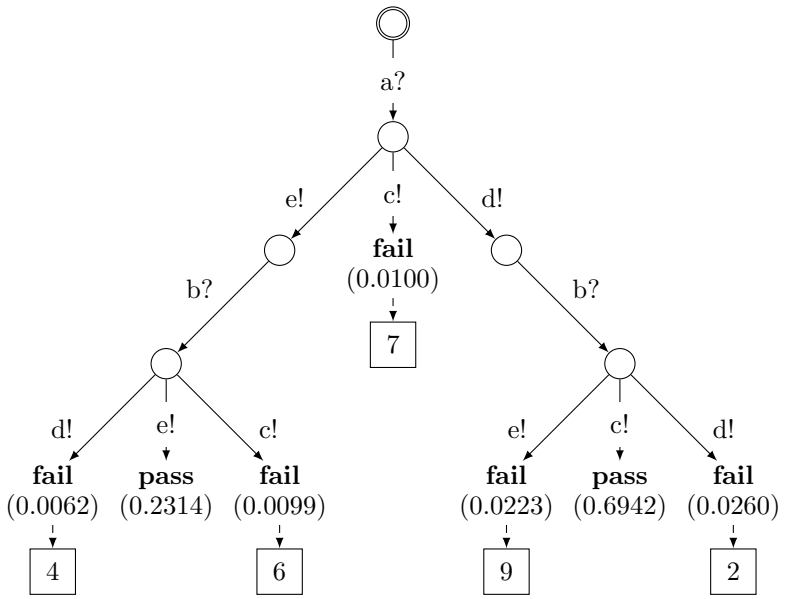


Figure 5.1: A test case t with the probability distribution of X_t shown

Besides the experiment of executing a test case t once, we are also interested in the experiment of executing it n times. The next definition formalises this experiment, by applying the standard probability theory concepts of random vectors and joint probability distributions.

Definition 5.6. *The random vector $X_t^n = (X_t, X_t, \ldots, X_t)$ is the sequence of executions that n trials of X_t yield.*

Obviously, the value range of X_t^n is $exec_t^n$. We assume that the probability distributions of all trials are independent, resulting in the following proposition.

Proposition 5.7. *Let t be a test case, and $\sigma_1, \sigma_2, \ldots \sigma_n \in t$, then*

$$\mathbb{P}[X_t^n = (\sigma_1, \sigma_2, \ldots, \sigma_n)] = \mathbb{P}[X_t = \sigma_1] \cdot \mathbb{P}[X_t = \sigma_2] \cdots \mathbb{P}[X_t = \sigma_n]$$

Example 5.8. Using the test case of Figure 5.1 once more, observe that $\mathbb{P}[X_t^3 = (a?\ c!, a?\ e!\ b?\ d!, a?\ d!\ b?\ c!)] = 0.01 \cdot 0.0062 \cdot 0.6942 = 0.000043$.

Note that this is the probability of obtaining these traces in this specific order; the probability of obtaining them in any order is six times as high (six being the number of permutations of the three traces).

To shorten the definitions that use sequences of executions, we introduce the following notation.

Definition 5.9. *Let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of some test case t. Let $\sigma \in t$. Then*

$$\sigma \sqsubseteq_\exists E \stackrel{def}{=} \exists e_i : \sigma \sqsubseteq e_i$$

$$\sigma \sqsubseteq_\forall E \stackrel{def}{=} \forall e_i : \sigma \sqsubseteq e_i$$

$$\sigma \sqsubseteq_\nexists E \stackrel{def}{=} \nexists e_i : \sigma \sqsubseteq e_i$$

Example 5.10. Consider the sequence of executions $E = (a?\ c!,\ a?\ e!\ b?\ d!)$. Using Definition 5.9, we have $a?\ e!\ b? \sqsubseteq_\exists E$, $a? \sqsubseteq_\forall E$, and $b? \sqsubseteq_\nexists E$.

5.4 Trace occurrence and branching

It will prove useful to estimate how often test executions visit the state from which a potential failure can occur. For a potential failure $\sigma a!$ this corresponds to the probability of an execution starting with σ. To support these kinds of calculations, the next definition gives the function p^{to}, called the t*race* o*ccurrence function*, that assigns to every trace σ its occurrence probability.

Definition 5.11. *Let t be a test case. Then, the* trace occurrence function *of t is the function $p_t^{\text{to}} : t \to [0..1]$ given for every $\sigma \in t$ by*

$$p_t^{\text{to}}(\sigma) = \mathbb{P}[\sigma \sqsubseteq X_t]$$

Note that p^{to} is not a distribution function, because an execution results in the occurrence of several (partly overlapping) traces.

Example 5.12. Considering the test case of Figure 5.1 again, we can calculate the function p^{to} (using Definition 5.1). For instance,

$$\begin{aligned}
&p_t^{\text{to}}(a?\ e!\ b?) \\
&\quad = \mathbb{P}[a?\ e!\ b? \sqsubseteq X_t] \\
&\quad = \sum_{\substack{\sigma \in exec_t \\ a?\ e!\ b? \sqsubseteq \sigma}} \mathbb{P}[X_t = \sigma] \\
&\quad = \mathbb{P}[X_t = a?\ e!\ b?\ d!] + \mathbb{P}[X_t = a?\ e!\ b?\ e!] + \mathbb{P}[X_t = a?\ e!\ b?\ c!] \\
&\quad = 0.0062 + 0.2314 + 0.0099 = 0.2475
\end{aligned}$$

Besides defining the occurrence probability of each trace, we can also define the probability distribution of the next output at each branching point in the test case. The following definition formalises this by means of the br*anching probability function* p^{br}. For every inner trace σ it gives the probability distribution of the next action from $\mathit{final}(\sigma)$.

Definition 5.13. *Let t be a test case. Then, the* branching probability function *of t is the function $p_t^{\text{br}} : \mathit{inner}_t \to \mathit{Distr}(L)$ given for every $\sigma \in \mathit{inner}_t$, $a \in L$ by*

$$p_t^{\text{br}}(\sigma)(a) = \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t]$$

We will denote $p_t^{\text{br}}(\sigma)(a)$ by $p_t^{\text{br}}(a \mid \sigma)$, to remind the reader that it is the probability of an a-action under the condition that we already saw the trace σ.

Example 5.14. Considering the test case of Figure 5.1 once more, we can calcu-

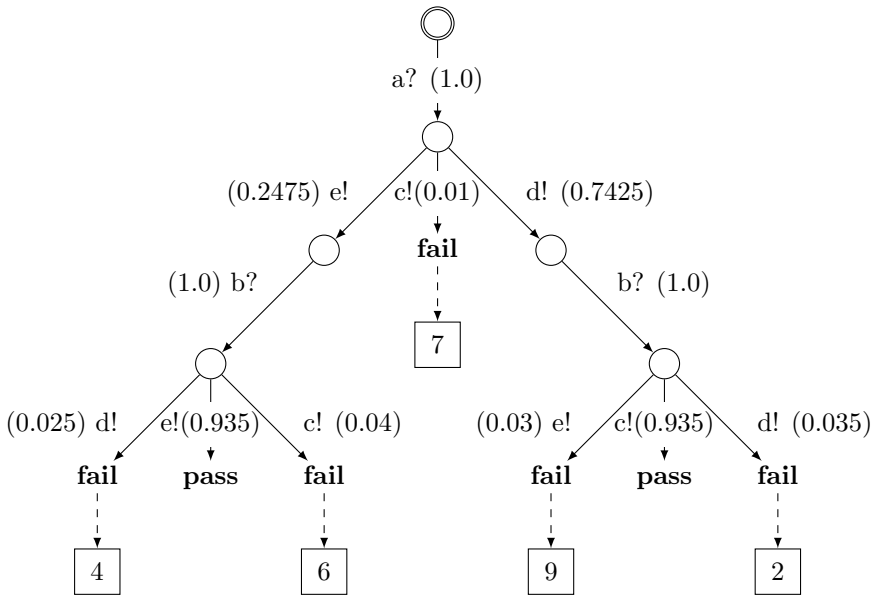


Figure 5.2: A test case with its branching probabilities p^{br} shown

late the function p^{br}. For instance,

$$
\begin{aligned}
&p_t^{\mathrm{br}}(d! \mid a?\ e!\ b?) \\
&\quad = \mathbb{P}[a?\ e!\ b?\ d! \sqsubseteq X_t \mid a?\ e!\ b? \sqsubseteq X_t] \\
&\quad = \frac{\mathbb{P}[a?\ e!\ b?\ d! \sqsubseteq X_t \wedge a?\ e!\ b? \sqsubseteq X_t]}{\mathbb{P}[a?\ e!\ b? \sqsubseteq X_t]} \\
&\quad = \frac{\mathbb{P}[a?\ e!\ b?\ d! \sqsubseteq X_t]}{\mathbb{P}[a?\ e!\ b? \sqsubseteq X_t]} \\
&\quad = \frac{\displaystyle\sum_{\substack{\sigma \in exec_t \\ a?\ e!\ b?d! \sqsubseteq \sigma}} \mathbb{P}[X_t = \sigma]}{\displaystyle\sum_{\substack{\sigma \in exec_t \\ a?\ e!\ b? \sqsubseteq \sigma}} \mathbb{P}[X_t = \sigma]} \\
&\quad = \frac{\mathbb{P}[X_t = a?\ e!\ b?\ d!]}{\mathbb{P}[X_t = a?\ e!\ b?\ d!] + \mathbb{P}[X_t = a?\ e!\ b?\ e!] + \mathbb{P}[X_t = a?\ e!\ b?\ c!]} \\
&\quad = \frac{0.0062}{0.0062 + 0.2314 + 0.0099} = 0.025
\end{aligned}
$$

Figure 5.2 shows the values of p_t^{br} for all branches.

Although the functions p^{to} and p^{br} and the probability distribution of X_t will all be useful in the subsequent chapters, it is not necessary to specify them separately. Obviously, only specifying the distribution of X_t would be sufficient, since p^{to} and p^{br} are defined based on it. However, it might be more intuitive to specify p^{br}. As the next two propositions show that both p^{to} and the complete probability distribution of X_t are derivable from p^{br}, this would indeed suffice.

Proposition 5.15. *Let t be a test case, and $\sigma = a_0 a_1 \dots a_n \in t$. Then, given the function p_t^{br}, it is possible to derive $p_t^{\text{to}}(\sigma)$. Specifically,*

$$p_t^{\text{to}}(\sigma) = \prod_{i=0}^{n} p_t^{\text{br}}(a_i \mid a_0 \dots a_{i-1})$$

Proof. Let $\sigma = a_0 a_1 \dots a_n \in t$. Then

$$\begin{aligned}
&\prod_{i=0}^{n} p_t^{\text{br}}(a_i \mid a_0 \dots a_{i-1}) \\
&= \prod_{i=0}^{n} \mathbb{P}[a_0 \dots a_{i-1} a_i \sqsubseteq X_t \mid a_0 \dots a_{i-1} \sqsubseteq X_t] && \{\text{Def. of } p^{\text{br}}\} \\
&= \prod_{i=0}^{n} \frac{\mathbb{P}[a_0 \dots a_{i-1} a_i \sqsubseteq X_t \wedge a_0 \dots a_{i-1} \sqsubseteq X_t]}{\mathbb{P}[a_0 \dots a_{i-1} \sqsubseteq X_t]} && \{\text{Def. of cond. prob.}\} \\
&= \prod_{i=0}^{n} \frac{\mathbb{P}[a_0 \dots a_{i-1} a_i \sqsubseteq X_t]}{\mathbb{P}[a_0 \dots a_{i-1} \sqsubseteq X_t]} && \{a_0 \dots a_{i-1} \sqsubseteq a_0 \dots a_{i-1} a_i\} \\
&= \frac{\mathbb{P}[a_0 \dots a_n \sqsubseteq X_t]}{\mathbb{P}[a_0 \dots a_{-1} \sqsubseteq X_t]} && \{\text{Basic rewriting}\} \\
&= \frac{\mathbb{P}[\sigma \sqsubseteq X_t]}{\mathbb{P}[\epsilon \sqsubseteq X_t]} && \{\sigma = a_0 \dots a_n,\ a_0 \dots a_{-1} = \epsilon\} \\
&= \mathbb{P}[\sigma \sqsubseteq X_t] && \{\text{Def. of } \sqsubseteq\} \\
&= p_t^{\text{to}}(\sigma) && \{\text{Def. of } p^{\text{to}}\}
\end{aligned}$$

□

Example 5.16. Considering Figure 5.2, we calculate

$$\begin{aligned}
p_t^{\text{to}}(a?\ e!\ b?\ e!) &= p_t^{\text{br}}(a? \mid \epsilon) \cdot p_t^{\text{br}}(e! \mid a?) \cdot p_t^{\text{br}}(b? \mid a!\ e!) \cdot p_t^{\text{br}}(e! \mid a!\ e!\ b?) \\
&= 1 \cdot 0.2475 \cdot 1.0 \cdot 0.935 = 0.2314
\end{aligned}$$

Proposition 5.17. *Let t be a test case. Given the function p_t^{to}, it is possible to derive the probability distribution function of X_t. Specifically, for every $\sigma \in exec_t$*

$$\mathbb{P}[X_t = \sigma] = p_t^{\text{to}}(\sigma)$$

Proof. Immediate from the definition of p^{to}. □

Proposition 5.15 and Proposition 5.17 result in the following corollary.

Corollary 5.18. *Let t be a test case. Given the valuations of p_t^{br}, it is possible to derive both the probability distribution of X_t and the valuations of p_t^{to}.*

5.5 Conditional branching probabilities

To calculate how certain we are of the absence of a fault after we have observed a given number of executions that did not fail, we need to estimate the probability that the fault results in a failure given its presence.

We first define the random variable of the state of presence of a fault. Every fault is either present or absent, but we do not know this in advance.

Definition 5.19. *Let $\mathcal{A}$ be an LTS. Then, for each $\sigma \in \{\sigma' a \mid \sigma' \in traces_{\mathcal{A}} \wedge a \in L^O_{\mathcal{A}} \wedge \sigma' a \notin traces_{\mathcal{A}}\}$, the random variable $SoP_t(\sigma)$ is the* state of presence *of σ in the system under test. The value range of $SoP_t(\sigma)$ is $\{present, absent\}$.*

Based on *SoP*, we can now define the c*onditional* br*anching probability function* p^{cbr}. It is similar to the previously defined branching probability function, except that it now assumes the presence of the action for which it defines the branching probability.

Definition 5.20. *Let t be a test case for an LTS $\mathcal{A}$. Then, the* conditional branching probability function *of t is the function $p^{\text{cbr}}_t : inner_t \times L^O_{\mathcal{A}} \to [0..1]$ given for each $\sigma \in inner_t$, $a \in L^O_{\mathcal{A}}$ by*

$$p^{\text{cbr}}_t(\sigma, a) = \begin{cases} \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a) = present] & , \text{ if } \sigma a \notin traces_{\mathcal{A}} \\ 0 & , \text{ otherwise} \end{cases}$$

We will denote $p^{\text{cbr}}_t(\sigma, a)$ by $p^{\text{cbr}}_t(a \mid \sigma)$, to remind the reader that it is the probability of an a-action under the condition that we already saw the trace σ (given the presence of σa).

The conditional branching probabilities will only be used for erroneous traces, so $p^{\text{cbr}}(\sigma, a)$ need only be defined if $\sigma a \notin traces_{\mathcal{A}}$. For technical reasons, it has been defined 0 *for all correct traces.*

Note that the probability of the occurrence of an erroneous trace might not only depend on its own presence, but also on the presence of other faults. The conditional branching probability function abstracts from this, and just specifies the probability given its own presence. This probability is actually the average of the occurrence probabilities given all combinations of present erroneous traces, each weighted by the probability of that situation. Section 5.8.3 will elaborate more on this matter.

5.6 The probabilistic execution model

Since we have shown that the probability distribution of X_t and the function p^{to} can be derived from p^{br}, the pair $\langle p^{\text{br}}, p^{\text{cbr}} \rangle$ contains the probabilities that are necessary to derive all others discussed in the previous sections. Therefore, we define the probabilistic execution model as follows.

Definition 5.21. *Let t be a test case, then its* probabilistic execution model *is the pair $\langle p^{\text{br}}_t, p^{\text{cbr}}_t \rangle$, with p^{br}_t its branching probability function and p^{cbr}_t its conditional branching probability function.*

Analysing systems based on these two characterisations has been done in literature before; for example, [RRS05] considers the 'probability of transitions between scenarios' (corresponding to our p^{br}) and the reliability of components (comparable to our p^{cbr}, more precisely to $1 - p^{\text{cbr}}$). Using these concepts, the reliability of an entire system is calculated.

Example 5.22. Figure 5.3 shows the test case of Figure 5.2, now also showing the conditional branching probabilities. For example, the probability of the occurrence of *c!* from the state on the bottom left is 0.8 in case the corresponding fault is present. Obviously, it is 0 in case the fault is absent. Not

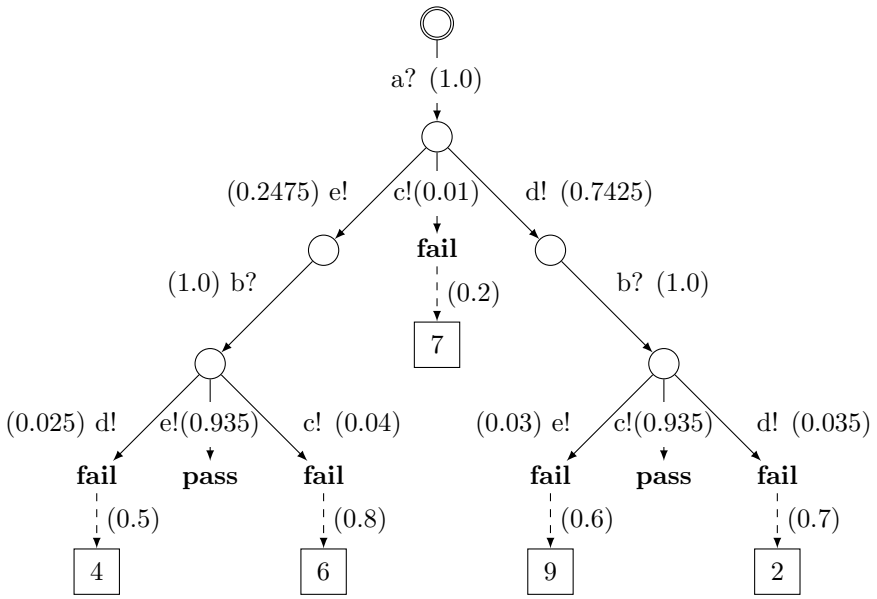


Figure 5.3: A test case with its probabilistic execution model shown

knowing whether or not it is present, the probability of the occurrence is 0.04. Apparently, the probability of the presence of the fault has been estimated at $0.04/0.8 = 0.05$.

5.7 Probabilistic fault automata

Instead of defining a probabilistic execution model $\langle p^{\mathrm{br}}, p^{\mathrm{cbr}} \rangle$ for every test case of a system, it is more practical to define it directly on its FA. To do so, we assume that the probability distribution of output actions has not changed when the system returns to a certain state after it has done some transitions. Furthermore, we assume that observing a fault from some state s after a trace σ does not necessarily mean that the same fault is also present from s after a trace $\sigma' \neq \sigma$. Future work attempting to incorporate potential dependencies might be an interesting extension.

Probabilistic automata (PAs) have been defined by [Seg95], but a new notion is needed for our purposes. After all, PAs assume a probability distribution over *all* actions, while in our case a distinction is made between input and output actions. Since input actions can always be performed with probability 1 when specified by a test case, we only need to specify the probability distribution the system employs 'choosing' its *output* actions. Furthermore, we need conditional branching probabilities as well.

Definition 5.23. *Let $\mathcal{F} = \langle \mathcal{A}, r \rangle$ be an FA, then a* branching probability function *for $\mathcal{F}$ is a function $p^{\mathrm{br}}_{\mathcal{F}} : S_{\mathcal{A}} \to \mathit{Distr}(L^O_{\mathcal{A}})$.*

Definition 5.24. *Let $\mathcal{F} = \langle \mathcal{A}, r \rangle$ be an FA, then a* conditional branching probability function *for $\mathcal{F}$ is a function $p^{\mathrm{cbr}}_{\mathcal{F}} : S_{\mathcal{A}} \times L^O_{\mathcal{A}} \to [0..1]$.*

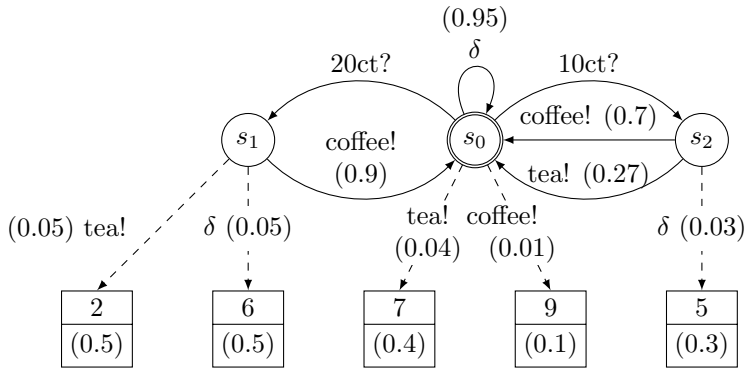


Figure 5.4: A probabilistic fault automaton

Definition 5.25. *A* probabilistic fault automaton *(PFA) $\mathcal{P}$ is a triple $\langle \mathcal{F}, p^{\mathrm{br}}_{\mathcal{F}}, p^{\mathrm{cbr}}_{\mathcal{F}} \rangle$, where $\mathcal{F}$ is an FA, $p^{\mathrm{br}}_{\mathcal{F}}$ a branching probability function for $\mathcal{F}$ and $p^{\mathrm{cbr}}_{\mathcal{F}}$ a conditional branching function for $\mathcal{F}$.*

Example 5.26. Figure 5.4 shows an example of a PFA. The branching probabilities of the output transitions, given by p^{br}, are shown on the transition arrows. The conditional branching probabilities, given by p^{cbr}, are shown in the error weight boxes.

The following definition states when a branching probability function and a conditional branching function for a *test case* are considered consistent with a PFA.

Definition 5.27. *Let $\mathcal{P} = \langle \langle \mathcal{A}, r \rangle, p^{\mathrm{br}}_{\mathcal{F}}, p^{\mathrm{cbr}}_{\mathcal{F}} \rangle$ be a PFA, t a test case for $\mathcal{A}$, p^{br}_t a branching probability function for t and p^{cbr}_t a conditional branching function for t. Then, p^{br}_t and p^{cbr}_t are* consistent *with $\mathcal{P}$ if for all $\sigma \in inner_t$, $a \in L_{\mathcal{A}}$*

$$p^{\mathrm{cbr}}_t(a \mid \sigma) = p^{\mathrm{cbr}}_{\mathcal{F}}(\mathit{final}(\sigma), a)$$

$$p^{\mathrm{br}}_t(a \mid \sigma) = \begin{cases} 1 & , \text{ if } a \in L^I \wedge \sigma a \in t \\ p^{\mathrm{br}}_{\mathcal{F}}(\mathit{final}(\sigma), a) & , \text{ if } a \notin L^I \wedge \sigma a \in t \\ 0 & , \text{ otherwise} \end{cases}$$

Example 5.28. Figure 5.5 gives a test case for the LTS corresponding to the PFA of Figure 5.4, with probability values assigned consistent with the PFA.

5.8 Deriving p^{cbr} and p^{br}

So far, we assumed that the user of our framework is able to provide the functions p^{cbr} and p^{br}. It might, however, not be trivial to derive them. This section therefore gives assistance in the process of obtaining p^{cbr} and p^{br}.

The deriving process consists of four steps. First, we estimate the branching probabilities assuming a flawless system. Second, we estimate for each fault the probability of its presence. Third, we estimate the conditional branching probabilities of erroneous traces. Fourth, the branching probabilities can be calculated based on the above.

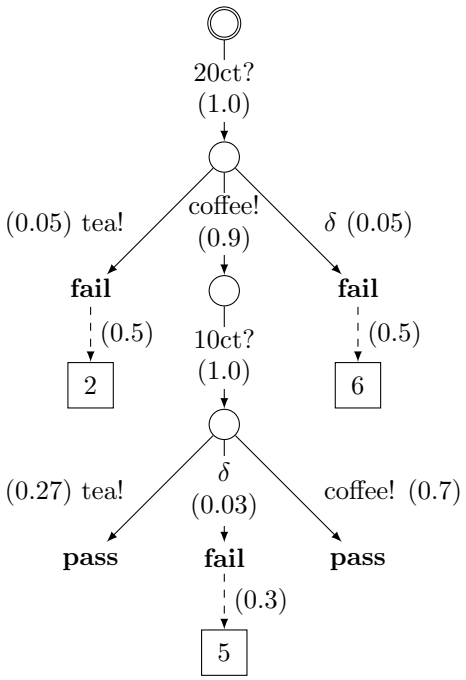


Figure 5.5: A test case consistent with the PFA of Figure 5.4

5.8.1 Branching probabilities assuming a flawless system

The first step in deriving p^{cbr} and p^{br} is obtaining the branching probabilities assuming a flawless system. We therefore define the function p^{fbr}, assigning to each trace its branching probability given the absence of all faults.

Definition 5.29. *Let t be a test case for an LTS $\mathcal{A}$. Then, the* flawless branching probability function *of t is the function $p_t^{\text{fbr}} : inner_t \to Distr(L)$ given for every $\sigma \in inner_t$, $a \in L$ by*

$$p_t^{\text{fbr}}(\sigma)(a) = \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \forall \sigma' \in t \setminus traces_{\mathcal{A}} : SoP_t(\sigma') = absent]$$

Similar to our notation for p^{br} and p^{cbr}, we will denote $p_t^{\text{fbr}}(\sigma)(a)$ by $p_t^{\text{fbr}}(a \mid \sigma)$.

Note that the flawless branching probability function is defined for all traces, but that obviously for every erroneous trace σa we have $p^{\text{fbr}}(a \mid \sigma) = 0$.

The flawless branching probabilities might be extracted from the specification, known by the implementers or measured by running the system prior to the testing phase.

Example 5.30. As an example for Definition 5.29, Figure 5.6 shows a test case with flawless branching probabilities assigned. Since we assume a flawless system, the erroneous transitions have been assigned probability 0.

5.8.2 Presence probabilities

The second step is to estimate the probability of the presence of each erroneous trace. For this purpose we define the *presence probability function* p^{pr}.

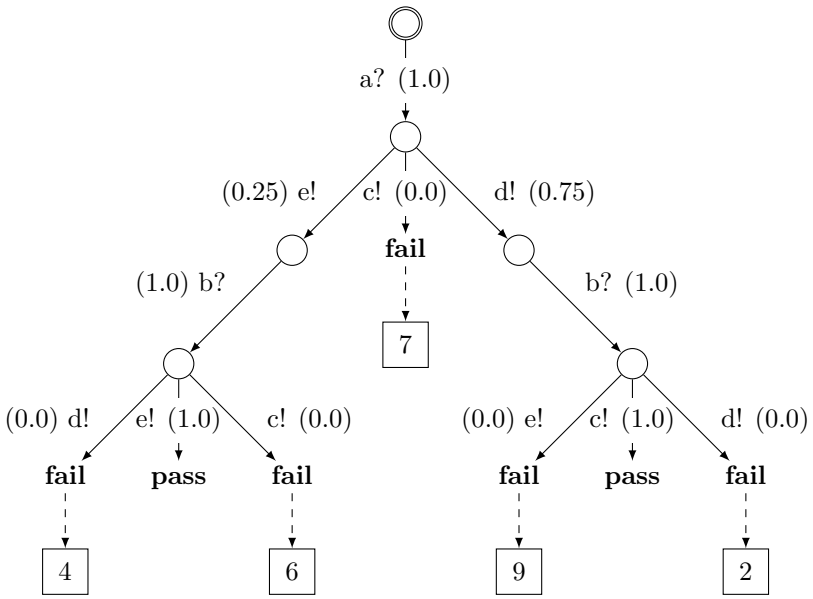


Figure 5.6: A test case including flawless branching probabilities

Definition 5.31. *Let t be a test case, then a* presence probability function *for t is a function $p_t^{\mathrm{pr}} : err_t \to [0..1]$. It maps each erroneous execution σ on the probability that it is present in the implementation under test. For all $\sigma \in err_t$*

$$p_t^{\mathrm{pr}}(\sigma) = \mathbb{P}[SoP_t(\sigma) = present]$$

These probabilities might result from experience of the programmers, or can be estimated based on the difficulty of the corresponding code (for instance by means of McCabe's cyclomatic complexity number [McC76]).

5.8.3 Deriving p^{cbr}

The third step is to derive the valuation of p^{cbr} for all erroneous traces in a test case.

Estimating the conditional branching probability of a certain erroneous trace could be difficult, since most of the times it will depend on the presence of other faults as well. There might even be a different probability distribution for every possible combination of present faults.

As an example, observe Figure 5.7. It gives a visual representation of the four possible probability distributions from a state from which one correct output action *a!* can take place, and two erroneous output actions *b!* and *c!* *might* be able to take place. Normally the system chooses the *a!* action with probability 1. When an error has been made enabling the *b!* output, this occurs with probability 0.7, and when *c!* is enabled it occurs 80 percent of the time. However, when both are present together, the *c!* action has a higher priority and 'steals' some of the probability mass of *b!*.

Note that in general the distribution of the case where several failures are present cannot be calculated based on the individual occurrences, since more information is needed about their mutual dependence.

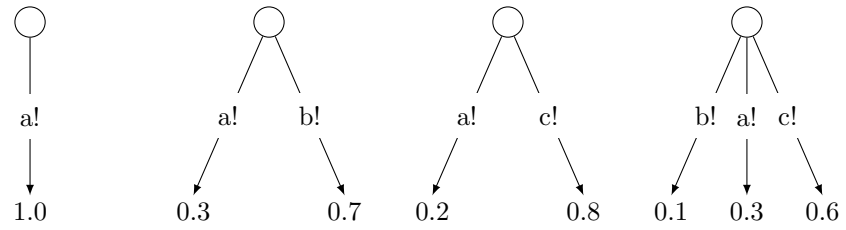


Figure 5.7: Failure probabilities

Now we define the *specialised conditional branching probability function* p^{scbr}, specifying for each erroneous trace its occurrence probability given exactly the state of presence of all other faults.

Definition 5.32. *Let t be a test case for an LTS $\mathcal{A}$. Then, the* specialised conditional branching probability function *of t is the function $p_t^{\mathrm{scbr}} : inner_t \times L^O \times \mathscr{P}(L^O) \to [0..1]$ given for every $\sigma \in inner_t$, $a \in L^O$, and $F \in \mathcal{P}(L^O)$ by*

$$p_t^{\mathrm{scbr}}(\sigma, a, F) = \begin{cases} \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \\ \qquad Errors_t = \{\sigma F\}] & \text{, if } \sigma a \in err_t \wedge \\ & \qquad \forall b \in F : \sigma b \in err_t \\ 0 & \text{, otherwise} \end{cases}$$

where the notation $\{\sigma F\}$ is used to denote the set $\{\sigma b \mid b \in F\}$ and the notation $Errors_t = Y$ is used as a shorthand for $\forall \sigma' \in err_t \backslash Y : SoP_t(\sigma') = absent \wedge \forall \sigma' \in Y : SoP_t(\sigma') = present$].

Specifying this function will probably be the most difficult part of the derivation process. However, we will show later on that an easy — and quite accurate — approximation exists.

Example 5.33. The probabilities of the situation given by Figure 5.7, assuming some trace σ leads to the top state, are described by the following specialised conditional branching probability function:

$p^{\mathrm{scbr}}(\sigma, a!, \varnothing) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \varnothing) = 0.0$	$p^{\mathrm{scbr}}(\sigma, c!, \varnothing) = 0.0$
$p^{\mathrm{scbr}}(\sigma, a!, \{a!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \{a!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, c!, \{a!\}) = 0.0$
$p^{\mathrm{scbr}}(\sigma, a!, \{b!\}) = 0.0$	$\boldsymbol{p^{\mathrm{scbr}}(\sigma, b!, \{b!\}) = 0.7}$	$p^{\mathrm{scbr}}(\sigma, c!, \{b!\}) = 0.0$
$p^{\mathrm{scbr}}(\sigma, a!, \{c!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \{c!\}) = 0.0$	$\boldsymbol{p^{\mathrm{scbr}}(\sigma, c!, \{c!\}) = 0.8}$
$p^{\mathrm{scbr}}(\sigma, a!, \{a!, b!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \{a!, b!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, c!, \{a!, b!\}) = 0.0$
$p^{\mathrm{scbr}}(\sigma, a!, \{a!, c!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \{a!, c!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, c!, \{a!, c!\}) = 0.0$
$p^{\mathrm{scbr}}(\sigma, a!, \{b!, c!\}) = 0.0$	$\boldsymbol{p^{\mathrm{scbr}}(\sigma, b!, \{b!, c!\}) = 0.3}$	$\boldsymbol{p^{\mathrm{scbr}}(\sigma, c!, \{b!, c!\}) = 0.6}$
$p^{\mathrm{scbr}}(\sigma, a!, \{a!, b!, c!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, b!, \{a!, b!, c!\}) = 0.0$	$p^{\mathrm{scbr}}(\sigma, c!, \{a!, b!, c!\}) = 0.0$

When all distributions are known, the normal conditional branching probabilities can be obtained. The following proposition states how this is done.

Proposition 5.34. *Let t be a test case for some LTS $\mathcal{A}$ and $p_t^{\mathrm{scbr}} : inner_t \times L^O \times \mathscr{P}(L^O) \to [0..1]$ the specialised conditional branching probability function for t. Then, for all $\sigma \in inner_t$, $b! \in L^O$ such that $\sigma b! \notin traces_{\mathcal{A}}$,*

$$p_t^{\mathrm{cbr}}(b! \mid \sigma) = \frac{\displaystyle\sum_{\substack{F \in \mathscr{P}(L^O) \\ \forall a \in F : \sigma a \notin traces_{\mathcal{A}}}} p_t^{\mathrm{scbr}}(\sigma, b!, F) \cdot \prod_{a \in F} p_t^{\mathrm{pr}}(\sigma a) \prod_{\substack{a \in L^O \backslash F \\ \sigma a \notin t}} (1 - p_t^{\mathrm{pr}}(\sigma a))}{p_t^{\mathrm{pr}}(\sigma b!)}$$

Proof. As mentioned before, $\mathit{Errors}_t = \{\sigma F\}$ is used to denote the event that exactly all faults in $\{\sigma F\}$ are present, while all other faults are absent. Note that given some trace σ, the events $\{\mathit{Errors}_t = \{\sigma F\} \mid F \in \mathscr{P}(L^O) \wedge \forall a \in F : \sigma a \notin \mathit{traces}_{\mathcal{A}}\}$ together form a partition of the total probability mass.

Assume $\sigma \in \mathit{inner}_t$, and $b! \in L^O$ such that $\sigma b! \notin \mathit{traces}_{\mathcal{A}}$. Then

$$
\begin{aligned}
& p_t^{\mathrm{cbr}}(b! \mid \sigma) \\
& \{\text{Def. of } p^{\mathrm{cbr}},\ \sigma b! \notin \mathit{traces}_{\mathcal{A}}\} \\
& = \mathbb{P}[\sigma b! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \mathit{SoP}_t(\sigma b!) = \mathit{present}] \\
& \{\text{Lemma 5.3}\} \\
& = \frac{\mathbb{P}[\sigma b! \sqsubseteq X_t \wedge \mathit{SoP}_t(\sigma b!) = \mathit{present} \mid \sigma \sqsubseteq X_t]}{\mathbb{P}[\mathit{SoP}_t(\sigma b!) = \mathit{present} \mid \sigma \sqsubseteq X_t]} \\
& \{\text{Independence of } \mathit{SoP}_t(\sigma b!) = \mathit{present} \text{ and } \sigma \sqsubseteq X_t\} \\
& = \frac{\mathbb{P}[\sigma b! \sqsubseteq X_t \wedge \mathit{SoP}_t(\sigma b!) = \mathit{present} \mid \sigma \sqsubseteq X_t]}{\mathbb{P}[\mathit{SoP}_t(\sigma b!) = \mathit{present}]} \\
& \{\sigma b! \sqsubseteq X_t \text{ implies } \mathit{SoP}_t(\sigma b!) = \mathit{present}\} \\
& = \frac{\mathbb{P}[\sigma b! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t]}{\mathbb{P}[\mathit{SoP}_t(\sigma b!) = \mathit{present}]} \\
& \{\text{Law of total probability, the events } \mathit{Errors}_t = \{\sigma F\} \text{ form a partition}\} \\
& = \sum_{\substack{F \in \mathscr{P}(L^O) \\ \forall a \in F : \sigma a \notin \mathit{traces}_{\mathcal{A}}}} \frac{\mathbb{P}[\sigma b! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \mathit{Errors}_t = \{\sigma F\}] \cdot \mathbb{P}[\mathit{Errors}_t = \{\sigma F\}]}{\mathbb{P}[\mathit{SoP}_t(\sigma b!) = \mathit{present}]} \\
& \{\text{Def. 5.32, restrictions on } F, \text{ assumption that } \sigma b! \notin \mathit{traces}_{\mathcal{A}}\} \\
& = \sum_{\substack{F \in \mathscr{P}(L^O) \\ \forall a \in F : \sigma a \notin \mathit{traces}_{\mathcal{A}}}} \frac{p^{\mathrm{scbr}}(\sigma, b!, F) \cdot \mathbb{P}[\mathit{Errors}_t = \{\sigma F\}]}{\mathbb{P}[\mathit{SoP}_t(\sigma b!) = \mathit{present}]} \\
& \{\text{Def. of } \mathbb{P}[\mathit{Errors}_t = \{\sigma F\}],\ \text{Def. of } p^{\mathrm{pr}}\} \\
& = \sum_{\substack{F \in \mathscr{P}(L^O) \\ \forall a \in F : \sigma a \notin \mathit{traces}_{\mathcal{A}}}} \frac{p_t^{\mathrm{scbr}}(\sigma, b!, F) \cdot \prod_{a \in F} p_t^{\mathrm{pr}}(\sigma a) \prod_{\substack{a \in L^O \setminus F \\ \sigma a \notin t}} (1 - p_t^{\mathrm{pr}}(\sigma a))}{p_t^{\mathrm{pr}}(\sigma b!)}
\end{aligned}
$$

□

Example 5.35. We use Figure 5.7 once more, and assume again that some trace σ leads to the top state. Furthermore, assume that for example $p^{\mathrm{pr}}(\sigma b!) = 0.02$ and $p^{\mathrm{pr}}(\sigma c!) = 0.05$. We are now able to calculate the conditional branching probabilities of $b!$ and $c!$. All terms of $p^{\mathrm{cbr}}(b! \mid \sigma)$ for which $b!$ is not included in F are omitted for brevity, since these terms are zero anyway. The same has been done for $p^{\mathrm{cbr}}(c! \mid \sigma)$.

$$
\begin{aligned}
p^{\mathrm{cbr}}(b! \mid \sigma) = \big(& p^{\mathrm{scbr}}(\sigma, b!, \{b!\}) \cdot p^{\mathrm{pr}}(\sigma b!) \cdot (1 - p^{\mathrm{pr}}(\sigma c!)) \\
& + p^{\mathrm{scbr}}(\sigma, b!, \{b!, c!\}) \cdot p^{\mathrm{pr}}(\sigma b!) \cdot p^{\mathrm{pr}}(\sigma c!)\big) / p^{\mathrm{pr}}(\sigma b!) \\
= & \left(0.7 \cdot 0.02 \cdot (1 - 0.05) + 0.3 \cdot 0.02 \cdot 0.05\right) / 0.02 = 0.68
\end{aligned}
$$

$$\begin{aligned} p^{\mathrm{cbr}}(c! \mid \sigma) = \big(&p^{\mathrm{scbr}}(\sigma, c!, \{c!\}) \cdot p^{\mathrm{pr}}(\sigma\, c!) \cdot (1 - p^{\mathrm{pr}}(\sigma\, b!)) \\ &+ p^{\mathrm{scbr}}(\sigma, c!, \{b!, c!\}) \cdot p^{\mathrm{pr}}(\sigma\, b!) \cdot p^{\mathrm{pr}}(\sigma\, c!)\big) / p^{\mathrm{pr}}(\sigma\, c!) \\ = &\left(0.8 \cdot 0.05 \cdot (1 - 0.02) + 0.6 \cdot 0.02 \cdot 0.05\right) / 0.05 = 0.796 \end{aligned}$$

As indicated by the example, the conditional branching probability of a fault in general seems to be fairly well approximated by the occurrence probabilities given no other faults are present. Put formally, $p^{\mathrm{cbr}}(a! \mid \sigma) \approx p^{\mathrm{scbr}}(\sigma, a!, \{a!\})$.

This is indeed true when the p^{pr}-values are small, because in that case the probability to have multiple faults present is very small. It also holds when the conditional branching probabilities in all fault presence scenarios are similar to the individual occurrence probabilities, because in that case it does not matter whether one or more faults are present. In practice, one might therefore choose to use these approximations, instead of estimating all the possible probability distributions.

5.8.4 Deriving p^{br}

Using the previously obtain values of p^{pr} and p^{cbr} for the erroneous traces and p^{fbr} for the correct traces, the values of p^{br} are derivable.

Proposition 5.36. *Let t be a test case for an LTS $\mathcal{A}$. Then, for all $\sigma \in inner_t$, $a \in L$,*

$$p^{\mathrm{br}}(a \mid \sigma) = \begin{cases} p^{\mathrm{cbr}}(a \mid \sigma) \cdot p^{\mathrm{pr}}(\sigma a) & , \text{ if } \sigma a \in t \setminus traces_{\mathcal{A}} \\ p^{\mathrm{fbr}}(a \mid \sigma) \cdot \left(1 - \sum\limits_{\substack{b! \in L^O \\ \sigma b! \notin traces_{\mathcal{A}}}} p^{\mathrm{br}}(b! \mid \sigma)\right) & , \text{ if } \sigma a \in t \cap traces_{\mathcal{A}} \\ 0 & , \text{ otherwise} \end{cases}$$

Proof.

- $\sigma a \in t \setminus traces_{\mathcal{A}}$

$$\begin{aligned} & p^{\mathrm{br}}(a \mid \sigma) \\ & \{\text{Def. of } p^{\mathrm{br}}\} \\ & = \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t] \\ & \{\text{Def. of conditional probabilities}\} \\ & = \frac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t]} \\ & \{\text{Basic rewriting}\} \\ & = \frac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t] \cdot \mathbb{P}[SoP_t(\sigma a) = present]} \cdot \mathbb{P}[SoP_t(\sigma a) = present] \\ & \{\text{Independence of } \mathbb{P}[\sigma \sqsubseteq X_t] \text{ and } \mathbb{P}[SoP_t(\sigma a) = present]\} \\ & = \frac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t \wedge SoP_t(\sigma a) = present]} \cdot \mathbb{P}[SoP_t(\sigma a) = present] \\ & \{\sigma a \sqsubseteq X_t \text{ implies } SoP_t(\sigma a) = present\} \\ & = \frac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a) = present]}{\mathbb{P}[\sigma \sqsubseteq X_t \wedge SoP_t(\sigma a) = present]} \cdot \mathbb{P}[SoP_t(\sigma a) = present] \end{aligned}$$

$\{$Def. of conditional probabilities$\}$
$= \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a) = present] \cdot \mathbb{P}[SoP_t(\sigma a) = present]$
$\{$Def. of p^{cbr}, $\sigma a \notin traces_{\mathcal{A}}$ (because of the case distinction)$\}$
$= p^{\mathrm{cbr}}(a \mid \sigma) \cdot \mathbb{P}[SoP_t(\sigma) = present]$
$\{$Def. of $p^{\mathrm{pr}}\}$
$= p^{\mathrm{cbr}}(a \mid \sigma) \cdot p^{\mathrm{pr}}(\sigma a)$

- $\sigma a \in t \cap traces_{\mathcal{A}}$

$p^{\mathrm{br}}(a \mid \sigma)$
$\{$Def. of $p^{\mathrm{br}}\}$
$= \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t]$
$\{$Def. of conditional probabilities$\}$
$= \dfrac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t]}$
$\{$Basic rewriting$\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot \dfrac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t] \cdot \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t]}$
$\{$Def. of conditional probabilities$\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot \dfrac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t]}{\mathbb{P}[\sigma \sqsubseteq X_t \wedge \sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}}]}$
$\{\sigma a \sqsubseteq X_t$ implies $\sigma a' \sqsubseteq X_t$ such that $\sigma a' \in traces_{\mathcal{A}}\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot \dfrac{\mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t \wedge \sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}}]}{\mathbb{P}[\sigma \sqsubseteq X_t \wedge \sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}}]}$
$\{$Def. of conditional probabilities$\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}}]$
$\{$Assuming $\sigma \sqsubseteq X_t \wedge \sigma a' \sqsubseteq X_t$ such that $\sigma a' \in traces_{\mathcal{A}}$ is equivalent to assuming $\sigma \sqsubseteq X_t$ and the absence of all errors, because $\sigma \sqsubseteq X_t$ already implies that no errors occur *until* the last step, and $\sigma a' \sqsubseteq X_t$ such that $\sigma a' \in traces_{\mathcal{A}}$ implies no errors occur *in* the last step$\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge \forall \sigma' \in t \setminus traces_{\mathcal{A}} : SoP_t(\sigma') = absent]$
$\{$Def. of $p^{\mathrm{fbr}}\}$
$= \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \in traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t] \cdot p^{\mathrm{fbr}}(a \mid \sigma)$
$\{$Basic probability theory$\}$
$= (1 - \mathbb{P}[\sigma a' \sqsubseteq X_t \text{ such that } \sigma a' \notin traces_{\mathcal{A}} \mid \sigma \sqsubseteq X_t]) \cdot p^{\mathrm{fbr}}(a \mid \sigma)$

{Basic rewriting}

$$= \left(1 - \sum_{\substack{b! \in L^O \\ \sigma b! \notin traces_{\mathcal{A}}}} \mathbb{P}[\sigma b! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t]\right) \cdot p^{\text{fbr}}(a \mid \sigma)$$

{Def. of p^{br}}

$$= \left(1 - \sum_{\substack{b! \in L^O \\ \sigma b! \notin traces_{\mathcal{A}}}} p^{\text{br}}(b! \mid \sigma)\right) \cdot p^{\text{fbr}}(a \mid \sigma)$$

- *Otherwise*

  By ioco theory, traces not included in a test case will never occur during an execution. Therefore, their branching probability is obviously 0. □

Example 5.37. As an example consider the test case of Example 5.30 again. Figure 5.8 shows it once more, now with estimations of the conditional branching probabilities of the erroneous output actions. These probabilities have been denoted between parentheses next to the dashed arrows. Furthermore, the presence of every fault is assumed to be 0.05. Based on these values and the probabilities of the correct behaviour (shown in Figure 5.6), the branching probability function has been derived and shown in the figure. The probabilities in this test case together form the probabilistic execution model.

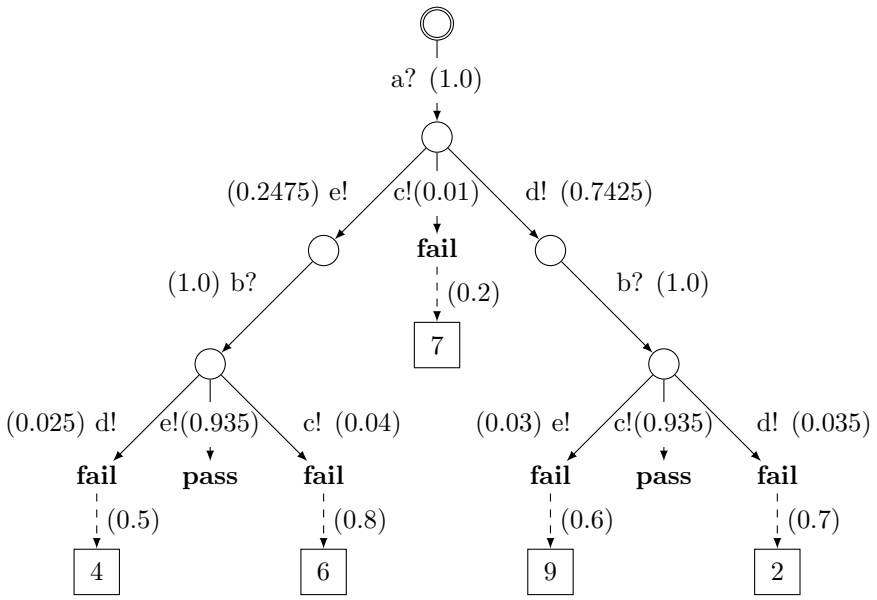


Figure 5.8: A test case with p^{br} and p^{cbr} shown

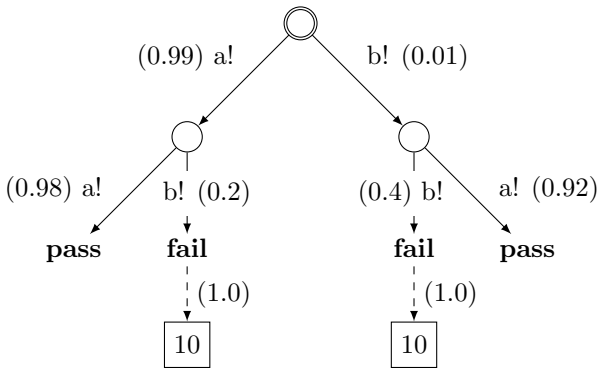


Figure 5.9: A test case t

5.9 Risk-based testing

In the framework of [BBS06] error weights represent the severity of the occurrence of a failure. However, as is known from literature on risk-based testing [Aml00], the risk of a fault is the severity of its occurrence *multiplied by the probability of its occurrence*. Since precisely these probabilities were derived in this chapter, we are now able to transform error weights into error risks.

First, an example is given to motivate this extension.

Example 5.38. Let t be the test case depicted in Figure 5.9. The occurrences of the erroneous traces *a! b!* and *b! b!* are considered equally bad, but the probability that *a! b!* actually occurs is much higher. It therefore seems reasonable to assume that it is more important to cover *a! b!* in a test case than to cover *b! b!*.

The important observation is that a fault that would cause severe problems in case it occurs, but that can approximately never occur in practice, can still be considered less important to fix than a fault with a lower severity but occurring all the time.

One simple way to incorporate these concerns in our framework is to let the user assign error weights in such a way that they directly represent the error risk. Stated differently, he should specify *the importance of preventing the existence of a fault*, instead of *the severity of its occurrence*. However, this process can also be automated by the function given in the next definition, discounting error weights by the probability of the occurrence of the associated error. Applying this function, the user can just assign weights in such a way that they represent how bad it would be if the error would occur, not worrying about the probability of occurrence yet.

Definition 5.39. *Let t be a test case for an LTS $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ and f a weighted fault model consistent with $\mathcal{A}$. Then, the weighted fault model based on f and discounted by p_t^{to} is the function $f_{p_t^{\text{to}}} : L^* \to \mathbb{R}^{\geq 0}$, such that for all $\sigma \in L^*$*

$$f_{p_t^{\text{to}}}(\sigma) = p_t^{\text{to}}(\sigma) f(\sigma)$$

Obviously this changes the total coverage by some factor, but if desirable this could easily be annulled by multiplying each error weight by that factor.

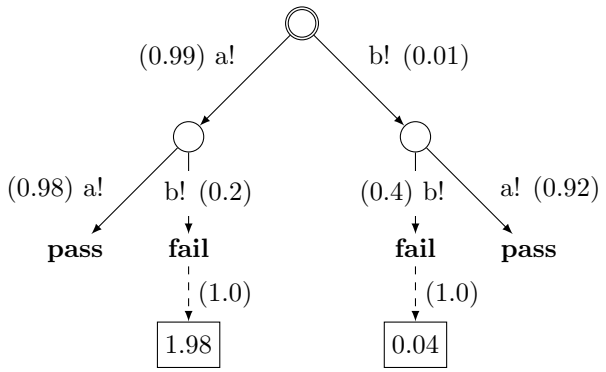


Figure 5.10: The test case of Figure 5.9 with error risks

However, since the absolute magnitude of error weights was already quite arbitrary and does not matter at all for the relative coverage, this does not seem necessary.

Example 5.40. Again consider the test case of Figure 5.9. The error weights when using $f_{p_t^{\text{to}}}$ instead of f are visualized in Figure 5.10. Clearly, the weights now support the believe that it is more important to prevent the existence of the erroneous trace *a! b!* than to prevent the existence of *b! b!*.

Discussion

When our aim is to let a high relative coverage correspond with covering most errors that we definitely need to fix, using $f_{p_t^{\text{to}}}$ clearly gives better results. However, if we want to really address how many faults are actually covered by a test case, independent of whether or not they appear often, it is better to use f.

Since all methods developed in this framework just expect some weighted fault model, they can be used no matter whether f or $f_{p_t^{\text{to}}}$ is appropriate. We choose to use the non-discounted version in our definitions, but by replacing each f by $f_{p_t^{\text{to}}}$ one directly obtains the same methods applied to the situation where error weights are discounted by a trace occurrence function.

Note that it seems appropriate to at least set the error weight of all unreachable faults to zero. That way, actual coverage is not influenced by faults that would never occur anyway.

Chapter 6

Evaluating actual coverage

Having defined all important probabilities by means of the probabilistic execution model, we now define our measures of coverage.

First, we discuss a rudimentary definition of fault coverage. It states that an erroneous trace σa is *covered by an execution* in case that execution starts with σ.

Second, we extend this definition to incorporate how certain we are of the absence of faults we have not observed. We define the *coverage probability* of some trace after a sequence of executions as the probability that we would have observed it in case it was present. *Fault coverage* is defined as the product of error weight and coverage probability. The *actual coverage of an execution* or sequence of executions is then defined as the sum of all fault coverages obtained by it. An appealing property of the definition of actual coverage is that the only probabilities it requires are the conditional branching probabilities of erroneous traces. Therefore, we do not need to estimate presence probabilities if we only want to evaluate actual coverage.

Finally, we look at an alternative definition of the coverage probabilities, defining them as the probability that the state of presence of a fault is equal to its observed state of presence. This way, however, we actually calculate the reliability of the system instead of coverage. We leave it to the user of the framework to choose which method to apply.

Organisation of this chapter
Section 6.1 describes the basic notion of fault coverage, followed by the more involved coverage probabilities in Section 6.2. Section 6.3 applies these probabilities to define fault coverage and actual coverage. The alternative definition of coverage probabilities is discussed in Section 6.4.

6.1 Basic notion of fault coverage

When executing a test case once, either one failure occurs or no failures occur. In the first case, obviously, the presence of a fault has been proved. In the second case, however, we have also learned something about the faults that *did not* result in a failure. After all, the fact that some failures did not occur even

though they were enabled, increases the confidence in their absence. Note that this also holds for traces that do result in a failure, up until the failure occurs.

Our most rudimentary form of coverage states that given an execution exactly every fault that either resulted in a failure or could have resulted in a failure is covered by that execution. Technically, a fault σa is *covered by an execution* in case that execution starts with σ.

Definition 6.1. *Let t be a test case and $\sigma \in exec_t$. Then, a trace $\sigma' a \in t$ is* covered *by σ if $\sigma' \sqsubseteq \sigma$.*

If $E = (e_1, e_2, \ldots, e_n)$ is an n-tuple of executions of t, then a trace $\sigma' a \in t$ is covered *by E if $\sigma' \sqsubseteq_\exists E$.*

Example 6.2. Looking again at Figure 5.3, consider the sequence of executions E = (*a? e! b? e!*, *a? c!*). Applying Definition 6.1 we find that E covers the erroneous traces *a? c!*, *a? e! b? d!* and *a? e! b? c!*. After all, *a?* and *a? e! b?* are prefixes of *a? e! b? e!*. On the other hand we see that *a? d! b? e!* and *a? d! b? d!* are *not* covered by *a? e! b? e!*. This is because *a? d! b?* is not a prefix of either *a? e! b? e!* or *a? c!*.

6.2 Coverage probabilities

Although the notion provided by Definition 6.1 gives a basic idea of the coverage of an execution or sequence of executions, it does not take into account *how certain* we are of the actual absence of each fault that was covered by appearing absent.

The following definition provides a notion of coverage that *does* take this into account. Instead of just being covered or not covered by an n-tuple of executions $E = (e_1, e_2, \ldots, e_n)$, every fault $\sigma a!$ is assigned a *coverage probability* p^{cov}. The definition is based on the number of executions c that reached the state from which the fault could occur but did not, so technically $c = |\{i \mid \sigma \sqsubseteq e_i\}|$.

The coverage probability $p_t^{\text{cov}}(\sigma a!, E)$ is defined as the probability that $\sigma a!$ is observed at least once, given that *final*(σ) is reached c times.

If a fault *has been observed*, we are certain of its presence, so we assign it a coverage probability of 1.

Definition 6.3. *Let t be a test case and $\sigma a! \in err_t$. Let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of t. Then, the* coverage probability p_t^{cov} of $\sigma a!$ after E is defined by

$$p_t^{\text{cov}}(\sigma a!, E) = \begin{cases} 1 & , \text{ if } \sigma a! \sqsubseteq_\exists E \\ 0 & , \text{ if } \sigma \sqsubseteq_\nexists E \\ \mathbb{P}[\sigma a! \sqsubseteq_\exists X_t^c \mid \sigma \sqsubseteq_\forall X_t^c \wedge SoP_t(\sigma a!) = present] & , \text{ otherwise, with } c = |\{i \mid \sigma \sqsubseteq e_i\}| \end{cases}$$

For single executions $\sigma' \in exec_t$, this degenerates to

$$p_t^{\text{cov}}(\sigma a!, \sigma') = \begin{cases} 1 & , \text{ if } \sigma a! \sqsubseteq \sigma' \\ 0 & , \text{ if } \sigma \not\sqsubseteq \sigma' \\ \mathbb{P}[\sigma a! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a!) = present] & , \text{ otherwise} \end{cases}$$

The next definition provides a shorthand notation, indicating the coverage probability after *any* sequence of executions with a certain number of executions covering a fault but none showing its presence.

Definition 6.4. *Let t be a test case and $\sigma a! \in err_t$. Let c be any natural number. Then we define*

$$p_t^{\mathrm{cov}}(\sigma a!, c) = \mathbb{P}[\sigma a! \sqsubseteq_{\exists} X_t^c \mid \sigma \sqsubseteq_{\forall} X_t^c \wedge SoP_t(\sigma a!) = present]$$

Corollary 6.5. *Let t be a test case and $\sigma a! \in err_t$. Then, the coverage probability p_t^{cov} of $\sigma a!$ after* any *sequence of executions $E = (e_1, e_2, \ldots, e_n)$ with $c = |\{i \mid \sigma \sqsubseteq e_i\}|$ and $\sigma a! \sqsubseteq_{\nexists} E$ is given by $p_t^{\mathrm{cov}}(\sigma a!, c)$.*

Proof. Immediate from Definition 6.3 and Definition 6.4. □

The next proposition provides a formula for the calculation of $p_t^{\mathrm{cov}}(\sigma a!, c)$.

Proposition 6.6. *Let t a test case and $\sigma a! \in err_t$. Then, for every $c \in \mathbb{N}$*

$$p_t^{\mathrm{cov}}(\sigma a!, c) = 1 - \left(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma)\right)^c$$

Proof. Let t a test case, $\sigma a! \in err_t$, and $c \in \mathbb{N}$. Then

$$\begin{aligned}
&p_t^{\mathrm{cov}}(\sigma a!, c)\\
&\{\text{Def. of } p^{\mathrm{cov}}\}\\
&= \mathbb{P}[\sigma a! \sqsubseteq_{\exists} X_t^c \mid \sigma \sqsubseteq_{\forall} X_t^c \wedge SoP_t(\sigma a!) = present]\\
&\{\text{Basic prob. theory and logic}\}\\
&= 1 - \mathbb{P}[\sigma a! \sqsubseteq_{\nexists} X_t^c \mid \sigma \sqsubseteq_{\forall} X_t^c \wedge SoP_t(\sigma a!) = present]\\
&\{\text{Def. of conditional probabilities}\}\\
&= 1 - \frac{\mathbb{P}[\sigma a! \sqsubseteq_{\nexists} X_t^c \wedge \sigma \sqsubseteq_{\forall} X_t^c \wedge SoP_t(\sigma a!) = present]}{\mathbb{P}[\sigma \sqsubseteq_{\forall} X_t^c \wedge SoP_t(\sigma a!) = present]}\\
&\{\text{Prop. 5.7}\}\\
&= 1 - \frac{(\mathbb{P}[\sigma a! \not\sqsubseteq X_t \wedge \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a!) = present])^c}{(\mathbb{P}[\sigma \sqsubseteq X_t \wedge SoP_t(\sigma a!) = present])^c}\\
&\{\text{Def. of conditional probabilities}\}\\
&= 1 - (\mathbb{P}[\sigma a! \not\sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a!) = present])^c\\
&\{\text{Basic probability theory}\}\\
&= 1 - (1 - \mathbb{P}[\sigma a! \sqsubseteq X_t \mid \sigma \sqsubseteq X_t \wedge SoP_t(\sigma a!) = present])^c\\
&\{\text{Def. of } p^{\mathrm{cbr}},\ \sigma a \in err_t\}\\
&= 1 - \left(1 - p^{\mathrm{cbr}}(a! \mid \sigma)\right)^c
\end{aligned}$$

□

Note that the number of times $\sigma a!$ occurs given c executions starting with σ is easily seen to be binomially distributed with parameters $n = c$ and $p = p^{\mathrm{cbr}}(a! \mid \sigma)$. The proposition then follows immediately from basic probability theory.

Example 6.7. Looking again at Figure 5.3, consider the sequence of executions $E = \{a?\ e!\ b?\ d!, a?\ d!\ b?\ e!\}$. We want to derive the coverage probability of the erroneous trace $a?\ c!$. Since both executions cover $a?\ c!$, we have

$c = 2$. Therefore, $p^{\text{cov}}(a?\ c!, E) = p^{\text{cov}}(a?\ c!, 2) = 1 - (1 - p^{\text{cbr}}(c! \mid a?))^2 = 1 - (1 - 0.2)^2 = 0.36$.

Furthermore, $p^{\text{cov}}(a?\ e!\ b?\ d!, E) = 1$, because this erroneous trace is shown to be present by the first execution of E.

6.2.1 Obtaining a certain coverage probability

Besides calculating the coverage probability of a fault based on a given sequence of executions, one might also need to know how often to test to obtain a certain coverage probability p. Fortunately, it is not difficult to obtain the inverse of $p_t^{\text{cov}}(\sigma a!, E)$. In this calculation we assume that the fault to be proved absent does not result in a failure, because such a failure already shows its presence. It is therefore in that case not possible anymore to prove its absence with a given certainty.

Proposition 6.8. *Let t be a test case and $\sigma a! \in err_t$. Let c be the number of times $\sigma a!$ is observed absent. Then*

$$p_t^{\text{cov}}(\sigma a!, E) \geq p \text{ iff } c \geq \left\lceil \frac{\log(1-p)}{\log(1 - p_t^{\text{cbr}}(a! \mid \sigma))} \right\rceil$$

Proof.

$$p_t^{\text{cov}}(\sigma a!, E) = 1 - (1 - p_t^{\text{cbr}}(a! \mid \sigma))^c \geq p$$

if and only if

$$(1 - p_t^{\text{cbr}}(a! \mid \sigma))^c \leq 1 - p$$

if and only if

$$c \geq \left\lceil \frac{\log(1-p)}{\log(1 - p_t^{\text{cbr}}(a! \mid \sigma))} \right\rceil \qquad \square$$

Example 6.9. Assume some erroneous output $\sigma a!$ has an occurrence probability of $p^{\text{cbr}}(a! \mid \sigma) = 0.2$. We want to obtain a coverage probabilities of 0.95. Using the formula derived above, we apparently need at least $\left\lceil \frac{\log(1-0.95)}{\log(1-p^{\text{cbr}}(a!|\sigma))} \right\rceil = \left\lceil \frac{\log(1-0.95)}{\log(1-0.2)} \right\rceil = \lceil 13.4 \rceil = 14$ executions that cover $\sigma a!$ (but not show its presence).

6.3 Fault coverage and actual coverage

Based on the coverage probability, we can now define a more involved notion of fault coverage. The coverage of a fault is simply defined as its error weight multiplied by the coverage probability. This way, the fault coverage of a fault that was observed to be present is just equal to its error weight. For all other faults, however, the error weights are weighed themselves to incorporate the fact that some are expected to be absent with more certainty than others.

Definition 6.10. *Let $\mathcal{A}$ be an LTS, t a test case for $\mathcal{A}$, σ an execution of t and f a weighted fault model consistent with $\mathcal{A}$. Furthermore, let $\sigma' a! \in err_t$. Then, the fault coverage of $\sigma' a!$ by σ, denoted by $faultCov_t(\sigma' a!, \sigma, f)$, is defined by*

$$faultCov_t(\sigma' a!, \sigma, f) = f(\sigma' a!) \cdot p_t^{\text{cov}}(\sigma' a!, \sigma)$$

Let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of t, then we define

$$faultCov_t(\sigma' a!, E, f) = f(\sigma' a!) \cdot p_t^{\text{cov}}(\sigma' a!, E)$$

When executing a test case, several faults might be covered. The notion of the *actual coverage* of an execution deals with this fact. It defines the coverage of a given execution or sequence of executions as the sum of the fault coverages of all faults in the system.

Similar to the framework on potential coverage, we define absolute and relative coverage measures.

Definition 6.11. *Let $\mathcal{A}$ be an LTS, t a test case for $\mathcal{A}$, σ an execution of t and f a weighted fault model consistent with $\mathcal{A}$. Then, the* absolute actual coverage *of σ, denoted by $absCov_t(\sigma, f)$, is defined by*

$$absCov_t(\sigma, f) = \sum_{\sigma' a! \in err_t} faultCov_t(\sigma' a!, \sigma, f)$$

The absolute actual coverage *of an n-tuple of executions E, denoted by $absCov_t(E, f)$, is defined by*

$$absCov_t(E, f) = \sum_{\sigma' a! \in err_t} faultCov_t(\sigma' a!, E, f)$$

The relative actual coverage *of an execution σ, denoting the fraction of the total error weight that is actually covered, is written $relCov_t(\sigma, f)$. Similarly, the relative actual coverage of an n-tuple of executions E is written $relCov_t(E, f)$. These coverages are defined by*

$$relCov_t(\sigma, f) = \frac{absCov_t(\sigma, f)}{totCov(f)}$$

$$relCov_t(E, f) = \frac{absCov_t(E, f)}{totCov(f)}$$

Recall that *totCov* was introduced in Definition 2.19 on page 15.

Example 6.12. Consider the test case depicted in Figure 6.1, with its probabilistic execution model shown. For this test case, one possible execution σ is *a? e! b? d!* (denoted by the thick arrows). The erroneous trace *a? c!* is covered by σ, since *a?* is one of its prefixes. Also, σ covers *a? e! b? d!* and *a? e! b? c!*. Applying the definitions of actual coverage yields

$$\begin{aligned} absCov(\sigma, f) &= 7 \cdot p^{\text{cov}}(a?\ c!, \sigma) + 4 \cdot p^{\text{cov}}(a?\ e!\ b?\ d!, \sigma) \\ &\quad + 6 \cdot p^{\text{cov}}(a?\ e!\ b?\ c!, \sigma) \\ &= 7 \cdot p^{\text{cov}}(a?\ c!, 1) + 4 \cdot 1 \\ &\quad + 6 \cdot p^{\text{cov}}(a?\ e!\ b?\ c!, 1) \\ &= 7 \cdot 0.2 + 4 \cdot 0.5 + 6 \cdot 0.8 = 8.2 \end{aligned}$$

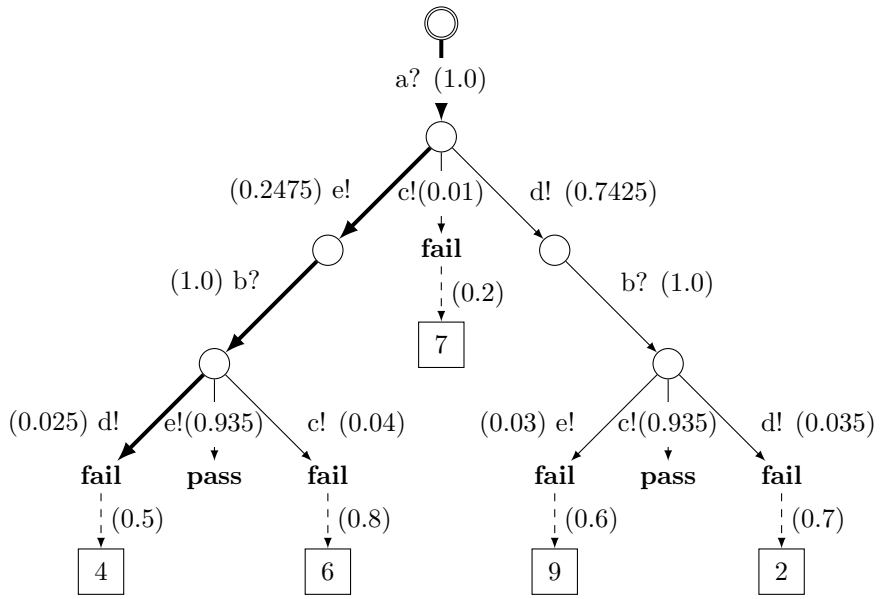


Figure 6.1: Test case with its probabilistic execution model shown

Note that the faults that were not covered by σ have been omitted from the calculation, since these have a coverage probability of zero anyway.

Using this result, we immediately obtain $\mathit{relCov}(\sigma, f) = \frac{8.2}{28} = 0.29$ (assuming the error weights shown are the only ones in the system).

For the execution sequence $E = (a?\ e!\ b?\ e!, a?\ e!\ b?\ e!, a?\ d!\ b?\ c!)$, applying the definitions yields

$$\begin{aligned}
\mathit{absCov}(E, f) &= 7 \cdot p^{\mathrm{cov}}(a?\ c!, E) \\
&\quad + 4 \cdot p^{\mathrm{cov}}(a?\ e!\ b?\ d!, E) \\
&\quad + 6 \cdot p^{\mathrm{cov}}(a?\ e!\ b?\ c!, E) \\
&\quad + 9 \cdot p^{\mathrm{cov}}(a?\ d!\ b?\ e!, E) \\
&\quad + 2 \cdot p^{\mathrm{cov}}(a?\ d!\ b?\ d!, E) \\
&= 7 \cdot p^{\mathrm{cov}}(a?\ c!, 3) \\
&\quad + 4 \cdot p^{\mathrm{cov}}(a?\ e!\ b?\ d!, 2) \\
&\quad + 6 \cdot p^{\mathrm{cov}}(a?\ e!\ b?\ c!, 2) \\
&\quad + 9 \cdot p^{\mathrm{cov}}(a?\ d!\ b?\ e!, 1) \\
&\quad + 2 \cdot p^{\mathrm{cov}}(a?\ d!\ b?\ d!, 1) \\
&= 18.976
\end{aligned}$$

Furthermore, $\mathit{relCov}(E, f) = \frac{18.976}{28} = 0.68$.

6.4 Another approach to coverage probabilities

Our approach to coverage probabilities has a significant advantage: it only uses the function p^{cbr}, not the function p^{br}. The evaluation of actual coverage can

therefore be performed without having to estimate presence probabilities or branching probabilities.

However, in case we do not mind also specifying p^{br}, an alternative approach is possible. We can then define the coverage probability of a fault as the probability that its state of presence is equal to its observed state of presence. In the examples of the remaining chapters, we will still use the original coverage probabilities.

To define the alternative coverage probabilities, we first formally define the *observed state of presence* of an erroneous execution.

Definition 6.13. *Let t a test case for an LTS $\mathcal{A}$, $\sigma' \in err_t$, and $\sigma \in exec_t$. Then, the* observed state of presence of σ' after σ*, $observedSoP(\sigma', \sigma)$, is defined by*

$$observedSoP(\sigma', \sigma) = \begin{cases} present & , if\ \sigma' \sqsubseteq \sigma \\ absent & , otherwise \end{cases}$$

Let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of t, then we define

$$observedSoP(\sigma', E) = \begin{cases} present & , if\ \sigma' \sqsubseteq_{\exists} E \\ absent & , otherwise \end{cases}$$

Note that the *observed* state of presence does not necessarily coincide with the *actual* state of presence. If we observed the presence of a fault, however, it does.

Example 6.14. Looking again at Figure 5.3, consider the sequence of executions $\mathrm{E} = (a?\ e!\ b?\ e!,\ a?\ c!)$. Applying Definition 6.13 we immediately obtain that

$$observedSoP(a?, E) = present$$
$$observedSoP(a?\ e!\ b?\ c!, E) = absent$$

Definition 6.15. *Let t be a test case and $\sigma a! \in err_t$. Let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of t. Then, the* coverage probability p_t^{cov} of $\sigma a!$ after E is defined by

$$p_t^{\mathrm{cov}}(\sigma a!, E) = \mathbb{P}[SoP_t(\sigma a!) = observedSoP(\sigma' a!, E) \mid X_t^n = E]$$

For single executions $\sigma' \in exec_t$ this degenerates to

$$p_t^{\mathrm{cov}}(\sigma a!, \sigma') = \mathbb{P}[SoP_t(\sigma a!) = observedSoP(\sigma a!, \sigma') \mid X_t = \sigma']$$

In case an execution shows the presence of a fault, p_t^{cov} is obviously 1. The next proposition provides a formula for calculating p^{cov} for a fault whose presence was *not* yet shown.

Proposition 6.16. *Let t be a test case, and $\sigma a! \in err_t$. Furthermore, let $E = (e_1, e_2, \ldots, e_n)$ be an n-tuple of executions of t, such that $\sigma a! \sqsubseteq_{\nexists} E$. Let $c = |\{i \mid \sigma \sqsubseteq e_i\}|$. Then*

$$p_t^{\mathrm{cov}}(\sigma a!, E) = \frac{1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}}{1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)} + (1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c \cdot \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}}$$

Proof. Let $\sigma\, a! \in \mathit{err}_t$. For layout purposes, let H_1 denote $\mathit{SoP}_t(\sigma\, a!) = \mathit{absent}$, and let H_2 denote $\mathit{SoP}_t(\sigma\, a!) = \mathit{present}$. Let H_3 denote the event $X_t^c = (e_1, \ldots, e_c)$ such that $|\{i \mid \sigma\, a! \sqsubseteq e_i\}| = 0$ and $|\{i \mid \sigma \sqsubseteq e_i\}| = c$.

The definition of p^{pr} and the proof of Proposition 5.36 immediately lead to

$$\mathbb{P}[H_2] = p_t^{\mathrm{pr}}(\sigma\, a!) = \frac{p_t^{\mathrm{br}}(a! \mid \sigma)}{p_t^{\mathrm{cbr}}(a! \mid \sigma)}$$

and because H_1 and H_2 form a partition it also leads to

$$\mathbb{P}[H_1] = 1 - \mathbb{P}[H_2] = 1 - \frac{p_t^{\mathrm{br}}(a! \mid \sigma)}{p_t^{\mathrm{cbr}}(a! \mid \sigma)}$$

Furthermore, the following conditional probabilities are relevant.

$$\mathbb{P}[H_3 \mid H_1] = p_t^{\mathrm{to}}(\sigma)^c \cdot 1$$
$$\mathbb{P}[H_3 \mid H_2] = p_t^{\mathrm{to}}(\sigma)^c \cdot (1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c$$

The factor $p_t^{\mathrm{to}}(\sigma)^c$ is just the probability of $|\{i \mid \sigma \sqsubseteq e_i\}| = c$, since H_1 and H_2 do not have any influence on this.

Then, given H_1, $\sigma\, a!$ is impossible, so the probability of $|\{i \mid \sigma\, a! \sqsubseteq e_i\}| = 0$ is 1. Given H_2, this probability is evidently $(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c$, since all c executions should not take the $a!$ branch of which we know it is present.

Now

$$\begin{aligned}
&p^{\mathrm{cov}}(\sigma\, a!, E)\\
&\{\text{Def. of } p^{\mathrm{cov}}\}\\
&= \mathbb{P}[H_1 \mid X_t^n = E]\\
&\{\text{Executions not covering } \sigma\, a! \text{ do not influence the probability of } H_1\}\\
&= \mathbb{P}[H_1 \mid H_3]\\
&\{\text{Bayes' theorem}\}\\
&= \frac{\mathbb{P}[H_3 \mid H_1] \cdot \mathbb{P}[H_1]}{\mathbb{P}[H_3 \mid H_1] \cdot \mathbb{P}[H_1] + \mathbb{P}[H_3 \mid H_2] \cdot \mathbb{P}[H_2]}\\
&\{\text{The probabilities derived above}\}\\
&= \frac{p_t^{\mathrm{to}}(\sigma)^c \cdot \left(1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}\right)}{p_t^{\mathrm{to}}(\sigma)^c \cdot \left(1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}\right) + p_t^{\mathrm{to}}(\sigma)^c \cdot \left(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma)\right)^c \cdot \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}}\\
&\{\text{Basic rewriting}\}\\
&= \frac{1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}}{1 - \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)} + \left(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma)\right)^c \cdot \frac{p_t^{\mathrm{br}}(a!|\sigma)}{p_t^{\mathrm{cbr}}(a!|\sigma)}}
\end{aligned}$$

□

Example 6.17. Looking again at Figure 5.3, consider the sequence of executions $E = \{a?\ e!\ b?\ d!, a?\ d!\ b?\ e!\}$. We want to calculate $p^{\mathrm{cov}}(a?\ c!, E)$. Notice

first that both executions cover $a?\ c!$, so $k = 2$. Therefore,

$$\begin{aligned} p^{\mathrm{cov}}(a?\ c!, E) &= p^{\mathrm{cov}}(a?\ c!, 2) \\ &= \frac{1 - \frac{p^{\mathrm{br}}(c!|a?)}{p^{\mathrm{cbr}}(c!|a?)}}{1 - \frac{p^{\mathrm{br}}(c!|a?)}{p^{\mathrm{cbr}}(c!|a?)} + (1 - p^{\mathrm{cbr}}(c! \mid a?))^2 \cdot \left(\frac{p^{\mathrm{br}}(c!|a?)}{p^{\mathrm{cbr}}(c!|a?)}\right)} \\ &= \frac{1 - \frac{0.01}{0.2}}{1 - \frac{0.01}{0.2} + (1 - 0.2)^2 \cdot \left(\frac{0.01}{0.2}\right)} \\ &= \frac{0.95}{0.95 + 0.032} = 0.9674 \end{aligned}$$

As another example, $p^{\mathrm{cov}}(a?\ e!\ b?\ d!, E) = 1$, since this erroneous trace is shown to be present by the first execution of E.

6.4.1 Obtaining a certain coverage probability

Again, we can calculate how often to test to obtain a certain coverage probability p^{cov}.

Proposition 6.18. *Let t be a test case and $\sigma a! \in err_t$. Let c be the number of times $\sigma a!$ is observed absent. Let $p_t^{\mathrm{pr}}(\sigma a!)$ be a shorthand for $\frac{p_t^{\mathrm{br}}(\sigma, a!)}{p_t^{\mathrm{cbr}}(\sigma, a!)}$. Then*

$$p^{\mathrm{cov}}(\sigma a!, E) \geq k \text{ iff } c \geq \left\lceil \frac{\log\left(\frac{\frac{1-p_t^{\mathrm{pr}}(\sigma a!)}{k} - 1 + p_t^{\mathrm{pr}}(\sigma a!)}{(p_t^{\mathrm{pr}}(\sigma a!))}\right)}{\log(1 - p_t^{\mathrm{cbr}}(\sigma, a!))} \right\rceil$$

Proof.

$$p_t^{\mathrm{cov}}(\sigma a!, E) = \frac{1 - p_t^{\mathrm{pr}}(\sigma a!)}{1 - p_t^{\mathrm{pr}}(\sigma a!) + (1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c \cdot p_t^{\mathrm{pr}}(\sigma, a!)} \geq k$$

if and only if

$$1 - p_t^{\mathrm{pr}}(\sigma a!) + (1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c \cdot p_t^{\mathrm{pr}}(\sigma a!) \leq \frac{1 - p_t^{\mathrm{pr}}(\sigma a!)}{k}$$

if and only if

$$(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c \cdot p_t^{\mathrm{pr}}(\sigma a!) \leq \frac{1 - p_t^{\mathrm{pr}}(\sigma a!)}{k} - 1 + p_t^{\mathrm{pr}}(\sigma a!)$$

if and only if

$$(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))^c \leq \frac{\frac{1-p_t^{\mathrm{pr}}(\sigma, a!)}{k} - 1 + p_t^{\mathrm{pr}}(\sigma a!)}{p_t^{\mathrm{pr}}(\sigma a!)}$$

if and only if

$$c \geq \left\lceil \frac{\log\left(\frac{\frac{1-p_t^{\mathrm{pr}}(\sigma\, a!)}{k} - 1 + p_t^{\mathrm{pr}}(\sigma\, a!)}{p_t^{\mathrm{pr}}(\sigma\, a!)}\right)}{\log(1 - p_t^{\mathrm{cbr}}(a! \mid \sigma))} \right\rceil$$

□

Example 6.19. Assume some erroneous output $\sigma\, a!$ has a conditional branching probability $p^{\mathrm{cbr}}(a! \mid \sigma) = 0.2$, and a ratio $p^{\mathrm{pr}}(\sigma\, a!) = \frac{p^{\mathrm{br}}(a!|\sigma)}{p^{\mathrm{cbr}}(a!|\sigma)} = 0.05$. We want to limit the probability of its presence to 1 percent, so formally $p^{\mathrm{cov}}(\sigma\, a!, E) \geq 0.99$. Using the formula derived above, we apparently need at least $\left\lceil \frac{\log(19/99)}{\log(0.8)} \right\rceil \approx \lceil 7.4 \rceil = 8$ executions that cover $\sigma\, a!$ (but not show its presence).

Discussion

We described two different ways of defining coverage probabilities. Although they seem quite similar, the impact of choosing one is significantly large.

Using the first definition of coverage probabilities, our notions of coverage indicate how many faults we actually covered. No matter how likely the presence of a fault is, we need to show its absence before we consider it covered. Moreover, the certainty of their absence is solely based on how likely it was to observe the fault in case it is present. A major advantage of this approach is that only the conditional branching probabilities are needed for our calculations.

Using the second definition of coverage probabilities, our notions of coverage indicate the reliability of the system. Because it incorporates the probability that faults are present, a high coverage probability is achieved for all sporadic faults without even having to test. After all, if the presence probability of a certain fault is low, the probability that it is absent is already high before testing has started.

Although both methods might be useful for different purposes, for our work it seems more appropriate to choose the first. That way, actual coverage denotes which faults were actually covered. As a pleasant side effect, less probabilities have to be estimated.

Chapter 7

Predicting actual coverage

This chapter discusses the actual coverage distribution of test cases, using the notion of actual coverage of the previous chapter. The distribution is represented by a random variable, denoting the actual coverage obtained by running a test case once or several times.

After describing the probability distribution of actual coverage we provide formulae for the calculation of its expected value and variance. We first present the case where a test case is executed just once, and then discuss the case where it is executed a certain number of times (since the former yields much easier formulae). We show that the expected value for both cases can be calculated in polynomial time, while this does not appear possible for the variance.

Furthermore, we show the important property that the expected value of the actual coverage of n executions of a test case is equal to its potential coverage if we take the limit of n to infinity.

We conclude by proposing an approximation for the calculation of expected actual coverage, chosen such that the only probabilities necessary are the conditional branching probabilities. Therefore, using this approximation we do not need to estimate presence probabilities anymore.

Organisation of this chapter
Section 7.1 states the probability distribution of the actual coverage of test cases, followed by its expected value in Section 7.2. Section 7.3 provides formulae for the variance. Finally, Section 7.4 discusses the approximation mentioned above.

7.1 Probability distributions for actual coverage

We first define the random variables for the absolute and relative actual coverage for a single execution, and derive their probability distribution functions. Then, we do the same for a *sequence* of executions. Important properties such as their expected value and variance and postponed to later sections.

Definition 7.1. *Let t be a test case for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Then, the random variable $A_{t,f}$ is the absolute actual coverage of X_t given the weighted fault model f. Furthermore, $R_{t,f}$ is its relative*

actual coverage. Formally, we define

$$A_{t,f} = absCov_t(X_t, f)$$
$$R_{t,f} = relCov_t(X_t, f)$$

Proposition 7.2. *The probability distribution functions of the random variables $A_{t,f}$ and $R_{t,f}$ are given by*

$$\mathbb{P}[A_{t,f} = x] = \sum_{\substack{\sigma \in exec_t \\ absCov_t(\sigma, f) = x}} \mathbb{P}[X_t = \sigma]$$
$$\mathbb{P}[R_{t,f} = x] = \mathbb{P}[A_{t,f} = x \cdot totCov(f)]$$

Proof.

$$\begin{aligned}
&\mathbb{P}[A_{t,f} = x] \\
&= \mathbb{P}[absCov_t(X_t, f) = x] && \{\text{Def. of } A\} \\
&= \sum_{\substack{\sigma \in exec_t \\ absCov_t(\sigma, f) = x}} \mathbb{P}[X_t = \sigma] && \{\text{Basic prob. theory}\}
\end{aligned}$$

The probability distribution function of $R_{t,f}$ follows directly from the definition of relative actual coverage. □

Definition 7.1 defines the actual coverage of a test case for only *one* execution. In practice, however, one might execute a test case several times to reveal more faults. The following definition deals with this fact, defining actual coverage for such a *sequence* of executions of a test case.

Definition 7.3. *Let t be a test case for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times t is executed. Then we define*

$$A^n_{t,f} = absCov_t(X^n_t, f)$$
$$R^n_{t,f} = relCov_t(X^n_t, f)$$

Proposition 7.4. *The probability distribution functions of the random variables $A^n_{t,f}$ and $R^n_{t,f}$ are given by*

$$\mathbb{P}[A^n_{t,f} = x] = \sum_{\substack{E=(e_1,\ldots,e_n) \in exec^n_t \\ absCov_t(E, f) = x}} \left(\prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \right)$$
$$\mathbb{P}[R^n_{t,f} = x] = \mathbb{P}[A^n_{t,f} = x \cdot totCov(f)]$$

Proof.

$$\begin{aligned}
&\mathbb{P}[A^n_{t,f} = x] \\
&\{\text{Def. of } A^n\} \\
&= \mathbb{P}[absCov_t(X^n_t, f) = x]
\end{aligned}$$

{Basic probability theory}

$$\sum_{\substack{E \in exec_t^n \\ absCov_t(E,f)=x}} \mathbb{P}[X_t^n = E]$$

{Prop. 5.7}

$$\sum_{\substack{E=(e_1,\ldots,e_n) \in exec_t^n \\ absCov_t(E,f)=x}} \left(\prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \right) \qquad \square$$

The probability distribution function of $R_{t,f}^n$ follows directly from the definition of relative actual coverage.

7.2 Expected actual coverage

The exact probability distribution functions derived in the previous section could be useful for some purposes, but are often more detailed than we actually need. It *is* interesting to know some statistical properties such as their expected value and variance. These provide quantities for an easy comparison of two or more different test cases.

In this section formulae are derived for the expected value of the random variables describing the absolute and relative actual coverage for both one and a sequence of executions. We also show that the expected value of the actual coverage of n executions of a test case is equal to its potential coverage if we take the limit of n to infinity.

Applying the definition of expected value, we immediately obtain

$$\mathbb{E}(A_{t,f}) = \sum_{\sigma \in exec_t} absCov_t(\sigma, f) \cdot \mathbb{P}[X_t = \sigma] \tag{7.1}$$

$$\mathbb{E}(R_{t,f}) = \frac{\mathbb{E}(A_{t,f})}{totCov(f)} \tag{7.2}$$

$$\mathbb{E}(A_{t,f}^n) = \sum_{E=(e_1,\ldots,e_n) \in exec_t^n} absCov_t(E, f) \cdot \prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \tag{7.3}$$

$$\mathbb{E}(R_{t,f}^n) = \frac{\mathbb{E}(A_{t,f}^n)}{totCov(f)} \tag{7.4}$$

Complexity

The formula for $\mathbb{E}(A_{t,f}^n)$ obtained this way is, unfortunately, very complex from a computational point of view. The number of terms to be summed is equal to the number of possible executions of the test case to the power of the number of times the test case is executed, making it unusable, even for small test cases with a small number of executions.

To be specific, we will express the complexity in terms of the basic operations (addition, multiplication, comparison).

The number of summands is m^n, with $m = |exec_t|$. Each summand requires n multiplications, and the calculation of *absCov*. To calculate *absCov*, we range over all erroneous traces (worst-case of order m). For each of them we calculate the number of executions covering it (n trace comparisons, so a maximum of nd

basic comparisons where d represents the maximal depth of the test case) and we perform an exponentiation (worst-case $\log(n)$ multiplications). In conclusion, the number of basic operations is in $O(m^n(n+m(nd+\log(n)))) = O(m^{n+1}nd)$.

The following theorem gives a much more efficient formula.

Theorem 7.5. *Let t be a test case for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times t is executed. Then*

$$\mathbb{E}(A^n_{t,f}) = \sum_{\sigma a \in err_t} f(\sigma a) \cdot \Bigg((1-(1-p^{\mathrm{to}}_t(\sigma a))^n) \cdot 1 + \sum_{k=0}^{n} \binom{n}{k} p^{\mathrm{to}}_t(\sigma)^k (1-p^{\mathrm{to}}_t(\sigma))^{n-k} \cdot (1-p^{\mathrm{br}}_t(a \mid \sigma))^k p^{\mathrm{cov}}_t(\sigma a, k) \Bigg)$$

Before proving this theorem formally, we will first intuitively explain it. Basically, the formula calculates how much each erroneous trace contributes to the expected actual coverage. To do so, two situations are distinguished.

First, consider the situation when a fault σa is observed during at least one execution. In this case, the fault is complete covered. Clearly, the number of occurrences of the trace σa during n executions is binomially distributed with parameters n and $p = p^{\mathrm{to}}(\sigma a)$. It immediately follows that the probability of complete coverage of σa is $1-(1-p^{\mathrm{to}}(\sigma a))^n$. The contribution is therefore $f(\sigma a) \cdot (1-(1-p^{\mathrm{to}}(\sigma a))^n)$.

Second, consider the situation when a fault σa is *not* observed during any of the n executions. It might still contribute to the expected actual coverage if σ is observed. The number of times σ is observed during n executions is obviously between 0 and n, and for each of these possibilities a different contribution to the expected actual coverage is made. For k observations of σ this contribution is equal to the error weight $f(\sigma a)$ multiplied by the coverage probability $p^{\mathrm{cov}}(\sigma a, k)$.

Since we do not know the number of times k a fault will be covered by an execution in advance, we sum over all possible k. For each k, we multiply the contribution by the probability of the occurrence of k covering executions. That probability is equal to k successes in a binomial distribution $B(n, p^{\mathrm{to}}(\sigma))$. Moreover, we need to exclude the scenario that during one of the k times σ is observed the trace continues with the erroneous output a, since that scenario was already covered. The probability of *not* observing σa when σ occurs k times is easily seen to be equal to $(1-p^{\mathrm{br}}(a \mid \sigma))^k$.

The next proof mathematically shows the correctness of the derivation above.

Proof. First observe that

$$\sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \exists e_i:\sigma a\sqsubseteq e_i}} p_t^{\mathrm{cov}}(\sigma a, E)\prod_{i=1}^{n}\mathbb{P}[X_t = e_i]$$

{Def. of p^{cov}, $\exists e_i : \sigma a \sqsubseteq e_i$}

$$= \sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \exists e_i:\sigma a\sqsubseteq e_i}} \prod_{i=1}^{n}\mathbb{P}[X_t = e_i]$$

{Complementary probability}

$$= 1 - \sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i:\sigma a\not\sqsubseteq e_i}} \prod_{i=1}^{n}\mathbb{P}[X_t = e_i]$$

{Def. 5.9, basic probability theory}

$$= 1 - \mathbb{P}[\sigma a \sqsubseteq_{\not\exists} X_t^n]$$

{Prop. 5.7}

$$= 1 - (\mathbb{P}[\sigma a \not\sqsubseteq X_t])^n$$

{Basic rewriting}

$$= 1 - (1 - \mathbb{P}[\sigma a \sqsubseteq X_t])^n$$

{Def. of p^{to}}

$$= 1 - \big(1 - p_t^{\mathrm{to}}(\sigma a)\big)^n$$

Furthermore, observe that

$$\sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i:\sigma a\not\sqsubseteq e_i}} p_t^{\mathrm{cov}}(\sigma a, E)\prod_{i=1}^{n}\mathbb{P}[X_t = e_i]$$

{Partitioning based on the number of e_i that cover σa}

$$= \sum_{k=0}^{n}\left(\sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i:\sigma a\not\sqsubseteq e_i \\ |\{i|\sigma\sqsubseteq e_i\}|=k}} p_t^{\mathrm{cov}}(\sigma a, E)\prod_{i=1}^{n}\mathbb{P}[X_t = e_i]\right)$$

{Corollary 6.5}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k)\left(\sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i:\sigma a\not\sqsubseteq e_i \\ |\{i|\sigma\sqsubseteq e_i\}|=k}} \prod_{i=1}^{n}\mathbb{P}[X_t = e_i]\right)$$

{Independence of $e_1, \ldots, e_n$}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} \sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i : \sigma a \not\sqsubseteq e_i \\ \sigma \sqsubseteq e_1, \ldots, \sigma \sqsubseteq e_k \\ \sigma \not\sqsubseteq e_{k+1}, \ldots, \sigma \not\sqsubseteq e_n}} \prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \right)$$

{Basic probability theory}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} \mathbb{P}[X_t^n = (e_1, \ldots, e_n) \text{ such that } \forall e_i : \sigma a \not\sqsubseteq e_i \wedge \sigma \sqsubseteq e_1, \ldots, \sigma \sqsubseteq e_k \wedge \sigma \not\sqsubseteq e_{k+1}, \ldots, \sigma \not\sqsubseteq e_n] \right)$$

{Prop. 5.7}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} (\mathbb{P}[\sigma a \not\sqsubseteq X_t \wedge \sigma \sqsubseteq X_t])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_t \wedge \sigma \not\sqsubseteq X_t])^{n-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} (\mathbb{P}[\sigma \sqsubseteq X_t] - \mathbb{P}[\sigma a \sqsubseteq X_t \wedge \sigma \sqsubseteq X_t])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_t \wedge \sigma \not\sqsubseteq X_t])^{n-k} \right)$$

{Def. of conditional probabilities}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} (\mathbb{P}[\sigma \sqsubseteq X_t] - \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t] \cdot \mathbb{P}[\sigma \sqsubseteq X_t])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_t \wedge \sigma \not\sqsubseteq X_t])^{n-k} \right)$$

{Basic rewriting, $\sigma \not\sqsubseteq X_t$ implies $\sigma a \not\sqsubseteq X_t$}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} (\mathbb{P}[\sigma \sqsubseteq X_t] - \mathbb{P}[\sigma a \sqsubseteq X_t \mid \sigma \sqsubseteq X_t] \cdot \mathbb{P}[\sigma \sqsubseteq X_t])^k \cdot (1 - \mathbb{P}[\sigma \sqsubseteq X_t])^{n-k} \right)$$

{Definition of p^{to} and p^{br}}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} \left(p_t^{\mathrm{to}}(\sigma) - p_t^{\mathrm{br}}(a \mid \sigma) p_t^{\mathrm{to}}(\sigma)\right)^k \left(1 - p_t^{\mathrm{to}}(\sigma)\right)^{n-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{n} p_t^{\mathrm{cov}}(\sigma a, k) \left(\binom{n}{k} p_t^{\mathrm{to}}(\sigma)^k (1 - p_t^{\mathrm{br}}(a \mid \sigma))^k (1 - p_t^{\mathrm{to}}(\sigma))^{n-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{n} \binom{n}{k} p_t^{\text{to}}(\sigma)^k (1 - p_t^{\text{to}}(\sigma))^{n-k} (1 - p_t^{\text{br}}(a \mid \sigma))^k p_t^{\text{cov}}(\sigma a, k)$$

Now

$$\mathbb{E}(A_{t,f}^n)$$

{Equation 7.3}

$$= \sum_{E=(e_1,\ldots,e_n)\in exec_t^n} absCov_t(E, f) \cdot \prod_{i=1}^{n} \mathbb{P}[X_t = e_i]$$

{Def. of *absCov*, Def. of *faultCov*}

$$= \sum_{E=(e_1,\ldots,e_n)\in exec_t^n} \left(\sum_{\sigma a \in err_t} f(\sigma a) \cdot p_t^{\text{cov}}(\sigma a, E) \right) \cdot \prod_{i=1}^{n} \mathbb{P}[X_t = e_i]$$

{Basic rewriting}

$$= \sum_{\sigma a \in err_t} f(\sigma a) \left(\sum_{E=(e_1,\ldots,e_n)\in exec_t^n} p_t^{\text{cov}}(\sigma a, E) \prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \right)$$

{Splitting the summation}

$$= \sum_{\sigma a \in err_t} f(\sigma a) \left(\sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \exists e_i : \sigma a \sqsubseteq e_i}} p_t^{\text{cov}}(\sigma a, E) \prod_{i=1}^{n} \mathbb{P}[X_t = e_i] + \right.$$

$$\left. \sum_{\substack{E=(e_1,\ldots,e_n)\in exec_t^n \\ \forall e_i : \sigma a \not\sqsubseteq e_i}} p_t^{\text{cov}}(\sigma a, E) \prod_{i=1}^{n} \mathbb{P}[X_t = e_i] \right)$$

{The equalities derived above}

$$= \sum_{\sigma a \in err_t} f(\sigma a) \cdot \left((1 - (1 - p_t^{\text{to}}(\sigma a))^n) \cdot 1 + \right.$$

$$\sum_{k=0}^{n} \binom{n}{k} p_t^{\text{to}}(\sigma)^k (1 - p_t^{\text{to}}(\sigma))^{n-k} \cdot$$

$$\left. (1 - p_t^{\text{br}}(a \mid \sigma))^k p_t^{\text{cov}}(\sigma a, k) \right) \qquad \square$$

Complexity

Observe that the number of summands is now equal to the number of possible erroneous outputs (which is less than the number of executions) *multiplied* by the number of times the test case is executed.

To be specific, the outer summation yields $m = |err_t|$ summands (which is less than the number of executions). Each summand requires an exponentiation to the power of n (complexity $\log(n)$), and another summation. This inner summation yields n summands. Each of these summands needs four exponentiations

(worst-case complexity $4\log(n)$). Moreover, the binomial coefficients $\binom{n}{0} \dots \binom{n}{n}$ have to be calculated, but this can be done in advance. Since each of these needs at most $2n$ multiplications, the calculation of all necessary coefficients is in $O(n^2)$.

In conclusion, the number of basic operations is in $O(n^2 + m(\log(n) + n(4\log(n)))) = O(n^2 + mn\log(n))$. Clearly this is a major improvement in comparison to the other formula for the expected value.

Example 7.6. To demonstrate calculating the expected value of the actual coverage of both a single execution and a sequence of executions, Figure 5.3 is used once more. As can be observed by the relation between the values for p^{br} and p^{cbr}, every error is assumed to be present with probability 0.05.

Since these and many of the calculations to come have been performed by Matlab, not all intermediate results are shown.

$$
\begin{aligned}
\mathbb{E}(A_{t,f}) = {} & absCov_t(a?\ e!\ b?\ d!, f) \cdot p_t^{\text{to}}(a?\ e!\ b?\ d!) \\
& + absCov_t(a?\ e!\ b?\ e!, f) \cdot p_t^{\text{to}}(a?\ e!\ b?\ e!) \\
& + absCov_t(a?\ e!\ b?\ c!, f) \cdot p_t^{\text{to}}(a?\ e!\ b?\ c!) \\
& + absCov_t(a?\ d!\ b?\ e!, f) \cdot p_t^{\text{to}}(a?\ d!\ b?\ e!) \\
& + absCov_t(a?\ d!\ b?\ c!, f) \cdot p_t^{\text{to}}(a?\ d!\ b?\ c!) \\
& + absCov_t(a?\ d!\ b?\ d!, f) \cdot p_t^{\text{to}}(a?\ d!\ b?\ d!) \\
& + absCov_t(a?\ c!, f) \cdot p_t^{\text{to}}(a?\ c!) \\
= {} & (7 \cdot 0.2 + 4 \cdot 1 + 6 \cdot 0.8) \cdot 0.2475 \cdot 0.025 \\
& + (7 \cdot 0.2 + 4 \cdot 0.5 + 6 \cdot 0.8) \cdot 0.2475 \cdot 0.935 \\
& + (7 \cdot 0.2 + 4 \cdot 0.5 + 6 \cdot 1) \cdot 0.2475 \cdot 0.04 \\
& + (7 \cdot 0.2 + 9 \cdot 1 + 2 \cdot 0.7) \cdot 0.7425 \cdot 0.03 \\
& + (7 \cdot 0.2 + 9 \cdot 0.6 + 2 \cdot 0.7) \cdot 0.7425 \cdot 0.935 \\
& + (7 \cdot 0.2 + 9 \cdot 0.6 + 2 \cdot 1) \cdot 0.7425 \cdot 0.035 \\
& + (7 \cdot 1) \cdot 0.01 \\
= {} & 8.3080
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}(A_{t,f}^5) = {} & 7 \cdot \Big((1 - (1 - 0.01)^5) \cdot 1 \\
& \quad + \sum_{i=0}^{5} \left(\binom{5}{i} 1^i \cdot 0^{5-i} \cdot (1 - 0.01)^i \cdot (1 - (1 - 0.2)^i) \right) \Big) \\
& + 4 \cdot \Big((1 - (1 - 0.2475 \cdot 0.025)^5) \cdot 1 \\
& \quad + \sum_{i=0}^{5} \left(\binom{5}{i} 0.2475^i \cdot (1 - 0.2475)^{5-i} \cdot (1 - 0.025)^i \cdot (1 - (1 - 0.5)^i) \right) \Big) \\
& + \\
& \vdots \\
& + 2 \cdot \Big((1 - (1 - 0.7425 \cdot 0.035)^5) \cdot 1 \\
& \quad + \sum_{i=0}^{5} \left(\binom{5}{i} 0.7425^i \cdot (1 - 0.7424)^{5-i} \cdot (1 - 0.035)^i \cdot (1 - (1 - 0.7)^i) \right) \Big) \\
= {} & 21.34
\end{aligned}
$$

This example shows the effect of multiple executions of the same test case.

Compared to the potential coverage defined in Chapter 2, we obtained much more information about the test case executions.

However, since potential coverage describes the coverage of a test case in terms of all the faults it can potentially cover, this *should* coincide with our measure for an infinite number of executions. The next theorem shows that this is indeed the case, assuming all potential faults are reachable.

Theorem 7.7. *Let t be a test case for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times t is executed. If for all erroneous traces $\sigma a!$ it holds that $p_t^{\text{to}}(\sigma a!) > 0$, then*

$$\lim_{n\to\infty} \mathbb{E}(A_{t,f}^n) = absPotCov(t, f)$$

Proof.

$$\text{Let } g_{t,f,n} = \Bigg((1 - (1 - p_t^{\text{to}}(\sigma a))^n) \cdot 1 + \sum_{k=0}^{n} \binom{n}{k} p_t^{\text{to}}(\sigma)^k (1 - p_t^{\text{to}}(\sigma))^{n-k} \cdot (1 - p_t^{\text{br}}(a \mid \sigma))^k p_t^{\text{cov}}(\sigma a, k)\Bigg)$$

then

$$\begin{aligned}
&\lim_{n\to\infty} \mathbb{E}(A_{t,f}^n) \\
&\{\text{Theorem 7.5, Def. of } g\} \\
&= \lim_{n\to\infty} \sum_{\sigma a \in err_t} (f(\sigma a) \cdot g_{t,f,n}) \\
&\{\text{Summation independent of limit}\} \\
&= \sum_{\sigma a \in err_t} \Big(f(\sigma a) \cdot \lim_{n\to\infty} g_{t,f,n}\Big) \\
&\{\text{Derivation below}\} \\
&= \sum_{\sigma a \in err_t} f(\sigma a) \\
&\{\text{Def. of } absPotCov,\ f(\sigma a) = 0 \text{ if } \sigma a \notin err_t\} \\
&= absPotCov(t, f)
\end{aligned}$$

since

$$\begin{aligned}
&\lim_{n\to\infty} g_{t,f,n} \\
&\{\text{Def. of } g\} \\
&= \lim_{n\to\infty} \Bigg((1 - (1 - p_t^{\text{to}}(\sigma a))^n) \cdot 1 + \sum_{k=0}^{n} \binom{n}{k} p_t^{\text{to}}(\sigma)^k (1 - p_t^{\text{to}}(\sigma))^{n-k} \cdot (1 - p_t^{\text{br}}(\sigma)(a))^k p_t^{\text{cov}}(\sigma a, k)\Bigg)
\end{aligned}$$

$$
\begin{aligned}
&\{\text{Limit of sum is sum of limits}\}\\
&= \lim_{n\to\infty}\left(1-(1-p_t^{\mathrm{to}}(\sigma a))^n\right)\\
&+ \lim_{n\to\infty}\sum_{k=0}^{n}\binom{n}{k}p_t^{\mathrm{to}}(\sigma)^k(1-p_t^{\mathrm{to}}(\sigma))^{n-k}\cdot\\
&\qquad (1-p_t^{\mathrm{br}}(\sigma)(a))^k p_t^{\mathrm{cov}}(\sigma a,k)\\
&\{\text{Def. of } p^{\mathrm{cov}}\}\\
&= \lim_{n\to\infty}\left(1-(1-p_t^{\mathrm{to}}(\sigma a))^n\right)\\
&+ \lim_{n\to\infty}\sum_{k=0}^{n}\binom{n}{k}p_t^{\mathrm{to}}(\sigma)^k(1-p_t^{\mathrm{to}}(\sigma))^{n-k}\cdot\\
&\qquad (1-p_t^{\mathrm{br}}(\sigma)(a))^k(1-(1-p^{\mathrm{cbr}}(a\mid\sigma))^k)\\
&\{p_t^{\mathrm{to}}(\sigma a)>0 \text{ (assumption)}\}\\
&= 1+\lim_{n\to\infty}\sum_{k=0}^{n}\binom{n}{k}p_t^{\mathrm{to}}(\sigma)^k(1-p_t^{\mathrm{to}}(\sigma))^{n-k}\cdot\\
&\qquad (1-p_t^{\mathrm{br}}(a\mid\sigma))^k(1-(1-p^{\mathrm{cbr}}(a\mid\sigma))^k)\\
&\{\text{Lem. B.2 } (p=p_t^{\mathrm{to}}(\sigma),\ q=1-p^{\mathrm{cbr}}(a\mid\sigma) \text{ and } r=1-p^{\mathrm{br}}(a\mid\sigma))\}\\
&= 1+0=1
\end{aligned}
$$

□

Note that Lemma B.2 can be found on page 115.

7.3 Variance of actual coverage

Assume we have some test case for which each execution has a relative actual coverage of fifty percent. Another test case has a relative actual coverage of zero percent for half of the executions, and of one hundred percent for the other half. These test cases have the same expected value of relative actual coverage, but are clearly not identical. Therefore, some way of describing the variance of the coverage is important.

Since each execution has its own probability and relative coverage percentage, it does not seem to be possible to describe the behaviour by any familiar statistical distribution. However, we can calculate the variance using its definition in a direct way.

Given a real-valued random variable X with valuations $x_1 \dots x_n$, each occurring with a probability $\mathbb{P}[X = x_i]$, the variance $\mathrm{Var}(X)$ is defined by

$$\mathrm{Var}(X) = \sum_{x\in S_X}((x-\mathbb{E}(X))^2\cdot\mathbb{P}[X=x])$$

This, combined with the familiar law $\mathrm{Var}(aX) = a^2X$ (for every constant a), immediately leads to the following equations.

$$\mathrm{Var}(A_{t,f}) = \sum_{\sigma\in exec_t}\left(absCov_t(\sigma,f)-\mathbb{E}(A_{t,f})\right)^2\cdot\mathbb{P}[X_t=\sigma] \tag{7.5}$$

$$\mathrm{Var}(A^n_{t,f}) = \sum_{E\in exec^n_t}\left(absCov_t(E,f)-\mathbb{E}(A^n_{t,f})\right)^2\cdot\mathbb{P}[X^n_t=E] \tag{7.6}$$

$$\mathrm{Var}(R_{t,f}) = \frac{\mathrm{Var}(A_{t,f})}{totcov(f)^2} \tag{7.7}$$

$$\mathrm{Var}(R^n_{t,f}) = \frac{\mathrm{Var}(A^n_{t,f})}{totcov(f)^2} \tag{7.8}$$

Unfortunately, the nonlinearity of variance does not seem to allow a more efficient calculation.

Example 7.8. To illustrate the concept just described, observe the test cases t_1 and t_2 shown in Figure 7.1 and Figure 7.2. Using Theorem 7.5, we see that t_1 has an expected absolute coverage of 5.3568, while t_2 has an expected absolute coverage of 10.7136. Since the total coverage for the second test case is twice as high, both have an equal relative actual coverage (assuming no other faults can be present). However, the test cases are obviously very different, since t_1 has the same actual coverage for all likely executions, while t_2 has much more variation.

To see how different they are, the variance of the relative actual coverage of a single execution is calculated using the formulae above. In order to use these, we need to know all the elements σ of $exec_t$, their absolute actual coverage and the probability of their occurrence. Table 7.1 shows precisely this information.

Now, the variances are calculated as follows.

$$\begin{aligned}
\mathrm{Var}(A_{t_1,f}) &= (5 - 5.3568)^2 \cdot 0.060025 + \cdots + \\
&\qquad (15 - 5.3568)^2 \cdot 0.02 = 3.127 \\
\mathrm{Var}(R_{t_1,f}) &= \frac{3.127}{55^2} = 0.001 \\
\mathrm{Var}(A_{t_2,f}) &= (14 - 10.7136)^2 \cdot 0.060025 + \cdots + \\
&\qquad (30 - 10.7136)^2 \cdot 0.02 = 35.715 \\
\mathrm{Var}(R_{t_2,f}) &= \frac{35.715}{110^2} = 0.003
\end{aligned}$$

The variance of the second test case is indeed larger than the variance of the first, giving a formal indication of the wider distribution of its actual coverage.

7.4 An approximation for coverage prediction

The formula derived in this chapter for the expected actual coverage of n executions of a test case is based on several probabilities. Since for every fault σa it included $p^{\mathrm{to}}(\sigma a)$ and $p^{\mathrm{br}}(a \mid \sigma)$, we need to estimate the probability that a certain fault occurs. As discussed in Chapter 5, this involves estimating the probability that a fault is present.

As the most difficult part of obtaining p^{br} and p^{cbr} seems to be the estimation of the presence probabilities, we propose an approximation that does not need this estimation. Instead, we assume that all presence probabilities are close to 0, by taking the limit. Assuming that the fault presence probabilities are indeed small in reality, this gives a good approximation.

First, we discuss several observations concerning systems with small presence probabilities.

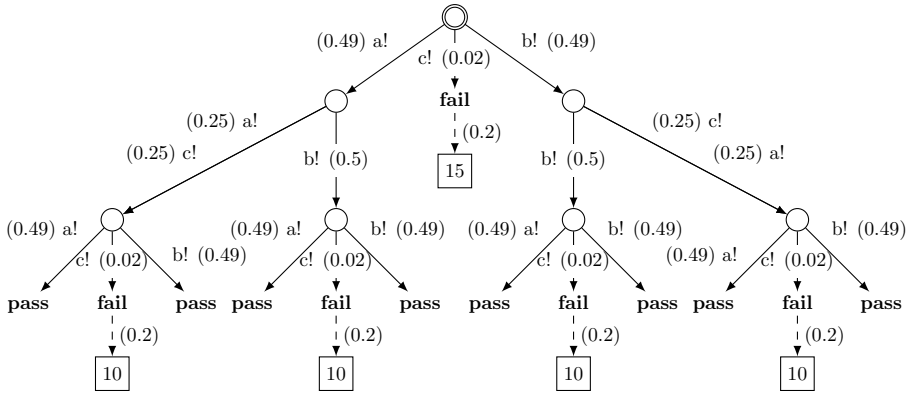


Figure 7.1: A test case t_1 with probabilities and error weights

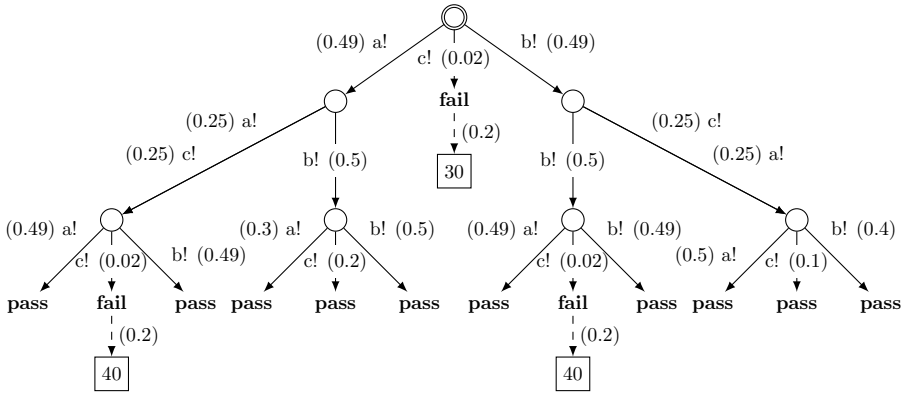


Figure 7.2: A test case t_2 with probabilities and error weights

(a) Test case t_1

σ	$p^{\text{to}}_{t_1}(\sigma)$	$absCov_{t_1}$
a! a! a!	0.060025	5
a! a! b!	0.060025	5
a! a! c!	0.002450	13
a! b! a!	0.120050	5
a! b! b!	0.120050	5
a! b! c!	0.004900	13
a! c! a!	0.060025	5
a! c! b!	0.060025	5
a! c! c!	0.002450	13
b! a! a!	0.060025	5
b! a! b!	0.060025	5
b! a! c!	0.002450	13
b! b! a!	0.120050	5
b! b! b!	0.120050	5
b! b! c!	0.004900	13
b! c! a!	0.060025	5
b! c! b!	0.060025	5
b! c! c!	0.002450	13
c!	0.020000	15

(b) Test case t_2

σ	$p^{\text{to}}_{t_2}(\sigma)$	$absCov_{t_2}$
a! a! a!	0.060025	14
a! a! b!	0.060025	14
a! a! c!	0.002450	46
a! b! a!	0.073500	6
a! b! b!	0.122500	6
a! b! c!	0.049000	6
a! c! a!	0.060025	14
a! c! b!	0.060025	14
a! c! c!	0.002450	46
b! a! a!	0.061250	6
b! a! b!	0.049000	6
b! a! c!	0.012250	6
b! b! a!	0.120050	14
b! b! b!	0.120050	14
b! b! c!	0.004900	46
b! c! a!	0.061250	6
b! c! b!	0.049000	6
b! c! c!	0.012250	6
c!	0.020000	30

Table 7.1: Executions with their probability and absolute actual coverage

Lemma 7.9. *Let σa be an erroneous trace, then*

$$\lim_{p^{\mathrm{pr}}(\sigma a)\to 0} p^{\mathrm{br}}(a \mid \sigma) = 0$$

Proof. Let σa be an erroneous trace, then

$$\begin{aligned}\lim_{p^{\mathrm{pr}}(\sigma a)\to 0} p^{\mathrm{br}}(a \mid \sigma) &= \lim_{p^{\mathrm{pr}}(\sigma a)\to 0} p^{\mathrm{cbr}}(a \mid \sigma)\cdot p^{\mathrm{pr}}(\sigma a) && \{\text{Prop. 5.36}\}\\ &= 0 && \{\text{Basic calculus}\}\end{aligned}$$ □

Lemma 7.10. *Let σa be an erroneous trace, then*

$$\lim_{p^{\mathrm{pr}}(\sigma a)\to 0} p^{\mathrm{to}}(\sigma a) = 0$$

Proof. Let σa_n be an erroneous trace, with $\sigma = a_0 \dots a_{n-1}$. Then

$$\begin{aligned}&\lim_{p^{\mathrm{pr}}(\sigma a_n)\to 0} p^{\mathrm{to}}(\sigma a_n)\\ &= \lim_{p^{\mathrm{pr}}(\sigma a_n)\to 0} \prod_{i=0}^{n} p^{\mathrm{br}}(a_i \mid a_0 \dots a_{i-1})\cdot && \{\text{Prop. 5.15}\}\\ &= \lim_{p^{\mathrm{pr}}(\sigma a_n)\to 0} \prod_{i=0}^{n-1} p^{\mathrm{br}}(a_i \mid a_0 \dots a_{i-1})\cdot p^{\mathrm{br}}(a_n \mid \sigma) && \{\text{Basic rewriting}\}\\ &= \lim_{p^{\mathrm{pr}}(\sigma a_n)\to 0} \prod_{i=0}^{n-1} p^{\mathrm{br}}(a_i \mid a_0 \dots a_{i-1})\cdot p^{\mathrm{cbr}}(a_n \mid \sigma)\cdot p^{\mathrm{pr}}(\sigma a_n) && \{\text{Prop. 5.36}\}\\ &= 0 && \{\text{Basic calculus}\}\end{aligned}$$ □

Lemma 7.11. *Let σa_n be an erroneous trace, with $\sigma = a_0 \dots a_{n-1}$. Then*

$$\lim_{\substack{\forall \sigma a \in err_t:\\ p^{\mathrm{pr}}(\sigma a)\to 0}} p^{\mathrm{to}}(\sigma) = \prod_{i=0}^{n} p^{\mathrm{fbr}}(a_i \mid a_0 \dots a_{i-1})$$

Proof. Let σa_n be an erroneous trace, with $\sigma = a_0 \dots a_{n-1}$. Then

$$\lim_{\substack{\forall \sigma a \in err_t:\\ p^{\mathrm{pr}}(\sigma a)\to 0}} p^{\mathrm{to}}(\sigma a_n)$$

{Prop. 5.15}

$$= \lim_{\substack{\forall \sigma a \in err_t:\\ p^{\mathrm{pr}}(\sigma a)\to 0}} \prod_{i=0}^{n} p^{\mathrm{br}}(a_i \mid a_0 \dots a_{i-1})$$

{Prop. 5.36}

$$= \lim_{\substack{\forall \sigma a \in err_t:\\ p^{\mathrm{pr}}(\sigma a)\to 0}} \prod_{i=0}^{n} p^{\mathrm{fbr}}(a_i \mid a_0 \dots a_{i-1}) \cdot \left(1 - \sum_{\substack{b! \in L^O\\ \sigma b! \notin traces_{\mathcal{A}}}} p^{\mathrm{br}}(b! \mid a_0 \dots a_{i-1})\right)$$

{Lemma 7.9}

$$= \lim_{p^{\mathrm{pr}}(\sigma a_n)\to 0} \prod_{i=0}^{n} p^{\mathrm{fbr}}(a_i \mid a_0 \dots a_{i-1})$$ □

We will use the shorthand $p^{\text{fto}}(\sigma)$ (in which fto is short for flawless trace occurrence) to denote $\lim_{\substack{\forall \sigma a \in err_t: \\ p^{\text{pr}}(\sigma a) \to 0}} p^{\text{to}}(\sigma)$.

Using these lemmas and Theorem 7.5, we directly obtain the following theorem.

Theorem 7.12. *Let t be a test case for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times t is executed. Then*

$$\lim_{\substack{\forall \sigma a \in err_t: \\ p^{\text{pr}}(\sigma a) \to 0}} \mathbb{E}(A^n_{t,f}) = \sum_{\sigma a \in err_t} f(\sigma a) \cdot \sum_{k=0}^{n} \binom{n}{k} p_t^{fto}(\sigma)^k (1 - p_t^{fto}(\sigma))^{n-k} p_t^{\text{cov}}(\sigma a, k)$$

Note that we now only need the flawless branching probabilities and the conditional branching probabilities. Therefore, we removed the need to estimate presence probabilities.

The next example shows the difference between using the original formula and the approximation for a small test case.

Example 7.13. Observe the test case in Figure 7.3. The presence probability of each erroneous trace has been estimated at 0.02, and all conditional branching probabilities have been estimated at 0.5. We will calculate the expected actual coverage for 5 executions using both the approximation and the original formula. We used a probability of 0.75 for starting with an *a!* and a probability of 0.25 for starting with a *b!* when calculating the flawless trace occurrences.

$$\mathbb{E}(A^5_{t,f}) = 32.2340$$
$$\lim_{\substack{\forall \sigma a \in err_t: \\ p^{\text{pr}}(\sigma a) \to 0}} \mathbb{E}(A^5_{t,f}) = 32.2425$$

This makes it very clear that even with presence probabilities as high as two percent, using the approximation only yields a very slight deviation. Interestingly, this deviation is much smaller than the presence probabilities we neglected. Changing the presence probabilities from 2 percent to 0 percent changed the expected actual coverage by not more than 0.03 percent.

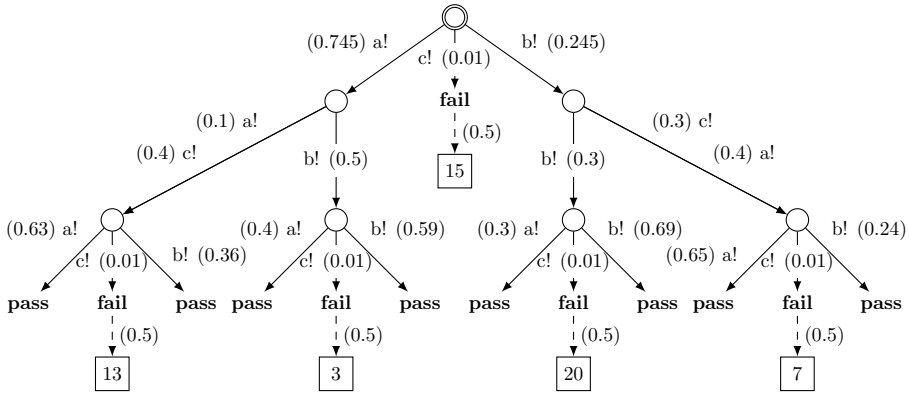


Figure 7.3: A test case t with probabilities and error weights

This was expected, since omitting the term $(1 - (1 - p_t^{\mathrm{to}}(\sigma\, a))^n)$ causes a slight *decrease* in coverage, whereas omitting $(1 - p_t^{\mathrm{br}}(a \mid \sigma))^k$ and replacing p^{to} by p^{fto} causes a slight *increase*. Apparently, in this case they quite nicely cancel each other out. It would be interesting and useful for a future research project to investigate more thoroughly whether this holds in general.

Chapter 8

Actual coverage of test suites

All definitions and propositions in the previous chapters have been given for just one test case t. However, in practice one often applies a *test suite*; a tuple of test cases. This chapter generalises our most important definitions and propositions to test suites.

We will first look at the probabilities associated with the execution of a test suite, by generalising trace occurrence probabilities and branching probabilities.

Then, the evaluation of actual coverage for test suites is discussed. Coverage probabilities are generalised as well, and this notion is used to generalise fault coverage and actual coverage.

Finally, the prediction of actual coverage is generalised. We show that the expected actual coverage can still be calculated efficiently.

Organisation of this chapter
First, Section 8.1 discusses all relevant probabilities. Then, the evaluation of actual coverage for test suites is covered in Section 8.2. We conclude with the prediction of actual coverage in Section 8.3.

8.1 Probabilities in test suite executions

Our framework expects that a probabilistic execution model is specified for every test case. Therefore, one could specify a test suite in which the conditional branching probability of a certain fault is not the same in different test cases. Since these probabilities are dependent on the implementation and not on the test case, this would obviously not make any sense.

To resolve this issue, we define *consistency* of test suites and assume all test suites are consistent. A test suite is consistent if all its test cases are consistent with *one single* probabilistic fault automaton conform Definition 5.27.

Definition 8.1. *Let $T = (t_1, \ldots, t_n)$ be a test suite. T is* consistent *if there exists a probabilistic fault automaton $\mathcal{P}$ such that every t_i is consistent with $\mathcal{P}$.*

Using consistency, trace occurrence probabilities for test suites can be introduced. The trace occurrence probability of a trace is defined as its trace occurrence probability in the test cases *in which it is contained.*

Definition 8.2. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite for an LTS $\mathcal{A}$. Then, the* trace occurrence function *of T is the function $p_T^{\mathrm{to}} : \bigcup t_i \to [0..1]$ given for every $\sigma \in \bigcup t_i$ by*

$$p_T^{\mathrm{to}}(\sigma) = \begin{cases} p_{t_i}^{\mathrm{to}}(\sigma) & \text{, if there exists a } t_i \in T \text{ such that } \sigma \in t_i \\ 0 & \text{, otherwise} \end{cases}$$

Note that p_T^{to} is not a probability distribution, for the same reason p_t^{to} was not.

For Definition 8.2 to be uniquely defined, it should hold that $p_{t_i}^{\mathrm{to}}(\sigma) = p_{t_j}^{\mathrm{to}}(\sigma)$ if $\sigma \in t_i \cap t_j$. This is proved by the next lemma.

Lemma 8.3. *Let $T = (t_1, \ldots, t_k)$ be consistent test suite for an LTS $\mathcal{A}$, and $\sigma \in L_{\mathcal{A}}^*$. Then, if $\sigma \in t_i \cap t_j$, it follows that $p_{t_i}^{\mathrm{to}}(\sigma) = p_{t_j}^{\mathrm{to}}(\sigma)$.*

Proof. Let $\sigma = a_1 a_2 \ldots a_n \in t_i \cap t_j$. Then, for all $0 \leq m \leq n-1$ we have $p_{t_i}^{\mathrm{br}}(a_{m+1} \mid a_1 \ldots a_m) = p_{t_j}^{\mathrm{br}}(a_{m+1} \mid a_1 \ldots a_m)$ by Definition 5.27 and the assumption that T is consistent. Using Proposition 5.15 we immediately obtain that $p_{t_i}^{\mathrm{to}}(\sigma) = p_{t_j}^{\mathrm{to}}(\sigma)$. □

Since p^{br} is based on p^{to}, it is easy to see that we can generalise this function to test suites in the same way.

Definition 8.4. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite for an LTS $\mathcal{A}$. Then, the* branching probability function *of T is the function $p_T^{\mathrm{br}} : L_{\mathcal{A}}^* \times L_{\mathcal{A}} \to [0..1]$ given for every $\sigma \in L_{\mathcal{A}}^*$, $a \in L$ by*

$$p_T^{\mathrm{br}}(\sigma, a) = \begin{cases} p_{t_i}^{\mathrm{br}}(a \mid \sigma) & \text{, if there exists a } t_i \in T \text{ such that } \sigma a \in t_i \\ 0 & \text{, otherwise} \end{cases}$$

As usual, $p_T^{\mathrm{br}}(\sigma, a)$ is denoted by $p_T^{\mathrm{br}}(a \mid \sigma)$. Note that although $p^{\mathrm{br}}(\sigma)$ was a distribution function for test cases, it is not for test suites.

8.2 Evaluating actual coverage for test suites

To generalise the definitions of fault coverage and actual coverage to test suites, we first define coverage probabilities for test suites. Since a single execution can be seen as a sequence of executions of length 1, we will not explicitly define our notions for a single execution anymore.

Similar to the definition of coverage probabilities for test cases, the definition for test suites is based on the number of executions c that reached the state from which a fault could occur. For test suites, however, not every execution that starts with σ reaches a state from which a fault σa can occur.

To understand the difficulties, assume we have a test suite consisting of some test cases $t_i \ldots t_k$ that observe after some trace σ, and some others that perform an input action after σ. Then, only the executions e_i for which $\sigma \sqsubseteq e_i$ of $t_1 \ldots t_k$ influence our confidence in the absent (or presence) of a fault $\sigma a!$, not those of the other test cases. After all, these always perform an input action, whether $\sigma a!$ is present or not.

This observation leads to the following definition.

Definition 8.5. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite and $\sigma a! \in err_{t_l}$ for some l. Let $E = (e_1^{t_1}, e_2^{t_1}, \ldots, e_n^{t_1}, e_1^{t_2}, e_2^{t_2}, \ldots, e_n^{t_2}, \ldots, e_1^{t_k}, e_2^{t_k}, \ldots, e_n^{t_k})$ be an $k \cdot n$-tuple of executions of T, such that $e_j^{t_i}$ is the j^{th} execution of test case t_i. Then, the* coverage probability p_T^{cov} of $\sigma a!$ after E is defined by

$$p_T^{\text{cov}}(\sigma a!, E) = \begin{cases} 1 & , \text{ if } \sigma a! \sqsubseteq_\exists E \\ 0 & , \text{ if } \sigma \sqsubseteq_\nexists E \\ \mathbb{P}[\sigma a! \sqsubseteq_\exists X_{t_l}^c \mid \sigma \sqsubseteq_\forall X_{t_l}^c \wedge SoP_T(\sigma a!) = present] & , \text{ otherwise} \end{cases}$$

with $c = |\{(i,j) \mid \sigma \sqsubseteq e_j^{t_i} \wedge \sigma a! \in t_i\}|$.

Recall that $X_{t_l}^c$ is the random variable of the execution of a test case t_l (chosen such that $\sigma a! \in t_l$) for c times. Because of Lemma 8.3, it does not matter which one we choose. Also note that this definition can easily be seen to degenerate to the definition of p^{cov} for test cases, in case a test suite only contains one test case.

The next corollary relates the coverage probability of a fault after a sequence of test suite executions to the function p_t^{cov} of Definition 6.4. Combined with Proposition 6.6 it provides an easy way to calculate p_T^{cov}.

Corollary 8.6. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite and $\sigma a! \in err_{t_l}$ for some l. Let $E = (e_1^{t_1}, e_2^{t_1}, \ldots, e_n^{t_1}, e_1^{t_2}, e_2^{t_2}, \ldots, e_n^{t_2}, \ldots, e_1^{t_k}, e_2^{t_k}, \ldots, e_n^{t_k})$ be* any *$k \cdot n$-tuple of executions of T, such that $e_j^{t_i}$ is the j^{th} execution of test case t_i, with $c = |\{(i,j) \mid \sigma \sqsubseteq e_j^{t_i} \wedge \sigma a! \in t_i\}|$ and $\sigma a! \sqsubseteq_\nexists E$. Then, the coverage probability p_T^{cov} of $\sigma a!$ after E is given by*

$$p_{t_l}^{\text{cov}}(\sigma a!, c)$$

Proof. Immediate from Definition 8.5 and Definition 6.4. □

We now define a shorthand, that will be useful in the next section.

Definition 8.7. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite for an LTS $\mathcal{A}$, $\sigma \in L_{\mathcal{A}}^*$ and $a \in L$, such that $\sigma a \notin traces_{\mathcal{A}}$. Let c be any natural number. Then we define*

$$p_T^{\text{cov}}(\sigma a!, c) = \begin{cases} p_{t_i}^{\text{cov}}(\sigma a!, c) & , \text{ if there exists a } t_i \in T \text{ such that } \sigma a! \in t_i \\ 0 & , \text{ otherwise} \end{cases}$$

Using p_T^{cov} we can now easily define the fault coverage and actual coverage for test suites. These are both very similar to the corresponding notions for test cases, and will therefore just be proposed without much explanation. For motivational details, we refer to Chapter 6.

Definition 8.8. *Let $\mathcal{A}$ be an LTS, $T = (t_1, \ldots, t_k)$ a consistent test suite for $\mathcal{A}$, and f a weighted fault model consistent with $\mathcal{A}$. Let E be a tuple of executions of T and $\sigma a! \in \bigcup err_{t_i}$. Then, the fault coverage of $\sigma a!$ by E, denoted by $faultCov_T(\sigma a!, E, f)$, is defined by*

$$faultCov_T(\sigma a!, E, f) = f(\sigma a!) \cdot p_T^{\text{cov}}(\sigma a!, E)$$

Definition 8.9. *Let $\mathcal{A}$ be an LTS, $T = (t_1, \ldots, t_k)$ a consistent test suite for $\mathcal{A}$, and f a weighted fault model consistent with $\mathcal{A}$. Let E be a tuple of executions of T. Then, the* absolute actual coverage *of E, denoted $absCov_T(E, f)$, is defined by*

$$absCov_T(E, f) = \sum_{\sigma\, a! \in \bigcup\, err_{t_i}} faultCov_T(\sigma\, a!, E, f)$$

The relative actual coverage *of E, denoting the fraction of the total error weight that is actually covered, is written $relCov_T(E, f)$. This is defined by*

$$relCov_T(E, f) = \frac{absCov_T(E, f)}{totCov(f)}$$

Example 8.10. As an example of the definitions proposed thus far, consider the test suite depicted in Figure 8.1. It consists of five test cases, two of which are equal. It is not difficult to see that the test suite is consistent. We will refer to the test cases by $t_1 \ldots t_5$, from left to right and from top to bottom.

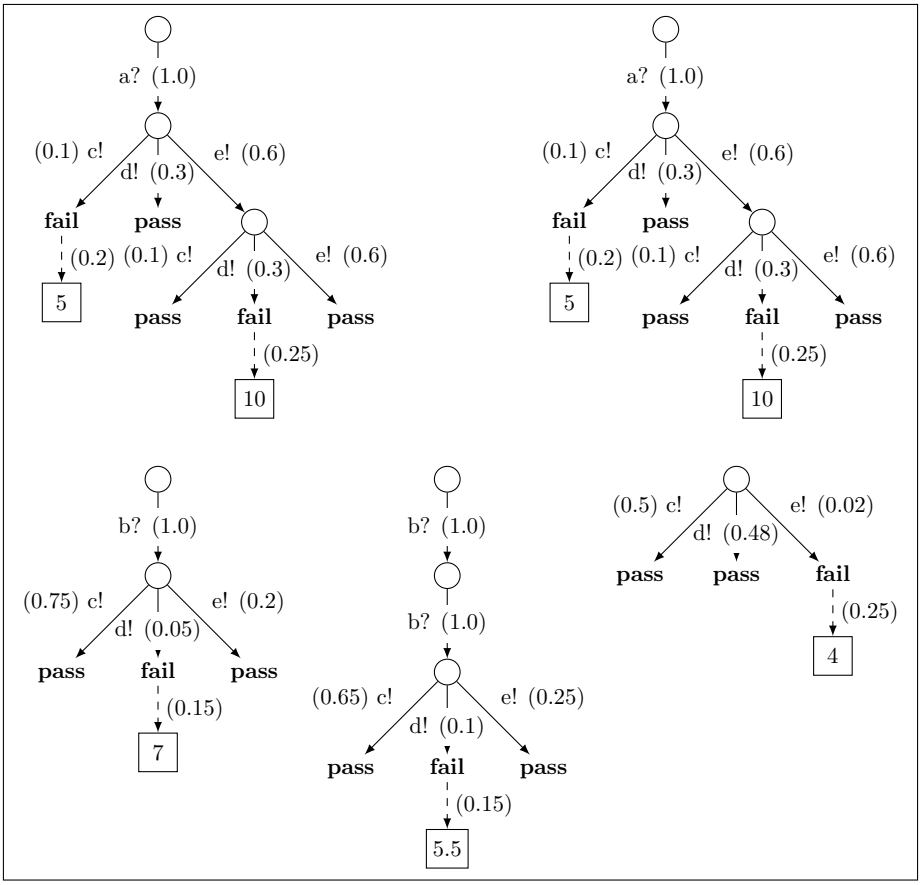


Figure 8.1: A test suite T consisting of five test cases

We first calculate some probabilities associated with the test suite.

$$p^{\mathrm{to}}_T(a?\ c!) = p^{\mathrm{to}}_{t_1}(a?\ c!) = p^{\mathrm{to}}_{t_2}(a?\ c!) = 1.0 \cdot 0.1 = 0.1$$
$$p^{\mathrm{br}}_T(c! \mid b?\ b?) = p^{\mathrm{br}}_{t_4}(c! \mid b?\ b?) = 0.65$$

Suppose the test suite has been executed three times, resulting in the following fifteen test case executions.

$$\begin{aligned} E = (&a?\ c!, a?\ d!, a?\ e?\ c!,\\ &a?\ d!, a?\ d!, a?\ e?\ e!,\\ &b?\ c!, b?\ c!, b?\ e?,\\ &b?\ b?\ c!, b?\ b?\ c!, b?\ b?\ e!\\ &c!, c!, e!) \end{aligned}$$

We can now calculate the coverage probability of each fault in the test suite, using Corollary 8.6 and Proposition 6.6.

$$p^{\mathrm{cov}}_T(a?\ c!, E) = 1$$
$$p^{\mathrm{cov}}_T(a?\ e!\ d!, E) = p^{\mathrm{cov}}_{t_1}(a?\ e!\ d!, 2) = 1 - \left(1 - p^{\mathrm{cbr}}_{t_1}(d! \mid a?\ e!)\right)^2 = 0.4375$$
$$p^{\mathrm{cov}}_T(b?\ d!, E) = p^{\mathrm{cov}}_{t_3}(b?\ d!, 3) = 1 - \left(1 - p^{\mathrm{cbr}}_{t_3}(d! \mid b?)\right)^3 = 0.386$$
$$p^{\mathrm{cov}}_T(b?\ b?\ d!, E) = p^{\mathrm{cov}}_{t_4}(b?\ b?\ d!, 3) = 1 - \left(1 - p^{\mathrm{cbr}}_{t_4}(d! \mid b?\ b?)\right)^3 = 0.386$$
$$p^{\mathrm{cov}}_T(e!, E) = 1$$

Notice that in the third calculation we use $p^{\mathrm{cov}}_{t_3}(b?\ d!, 3)$, even though there are 6 executions e_i such that $b? \sqsubseteq e_i$. However, only three of these resulted from a test case that observed after the $b?$.

Calculating the actual coverage of E has now become easy.

$$\begin{aligned} absCov_T(E, f) &= \sum_{\sigma a! \in \bigcup err_{t_i}} faultCov_T(\sigma a!, E, f)\\ &= \sum_{\sigma a! \in \bigcup err_{t_i}} f(\sigma a!) \cdot p^{\mathrm{cov}}_T(\sigma a!, E)\\ &= 5 \cdot 1 + 10 \cdot 0.4375 + 7 \cdot 0.386 + 5.5 \cdot 0.386 + 4 \cdot 1 = 18.2 \end{aligned}$$

8.3 Predicting actual coverage of test suites

The random variables for the actual coverage of test cases are generalised very easily as well.

Definition 8.11. *Let $T = (t_1, \ldots, t_k)$ be a consistent test suite for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times T is executed. Then we define*

$$A^n_{T,f} = absCov_T((X^n_{t_1}, X^n_{t_2}, \ldots, X^n_{t_k}), f)$$
$$R^n_{T,f} = relCov_T((X^n_{t_1}, X^n_{t_2}, \ldots, X^n_{t_k}), f)$$

where $(X^n_{t_1}, X^n_{t_2}, \ldots, X^n_{t_k})$ denotes the $k \cdot n$-tuple $(e^{t_1}_1, e^{t_1}_2, \ldots, e^{t_1}_n, e^{t_2}_1, e^{t_2}_2, \ldots, e^{t_2}_n, \ldots, e^{t_k}_1, e^{t_k}_2, \ldots, e^{t_k}_n)$ such that $(e^{t_i}_1, e^{t_i}_2, \ldots, e^{t_i}_n) = X^n_{t_i}$ for all $1 \leq i \leq k$.

Similar to the situation where we handled just test cases, we can also derive an efficient formula for the expected absolute actual coverage of a test suite. The following function will assist in this formula. It provides a shorthand for the number of test cases of a test suite that contain a certain trace.

Definition 8.12. *Let T be a test suite for an LTS $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ and $\sigma \in L^*$, then the function $c_T : L^* \to [0..1]$ denotes the number of test cases of T that contain σ. For all $\sigma \in L^*$*

$$c_T(\sigma) = |\{t \in T \mid \sigma \in t\}|$$

The following theorem uses this function to state the expected value of the actual coverage of a test suite.

Theorem 8.13. *Let $T = (t_1, \ldots, t_k)$ be a test suite for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times T is executed. Then*

$$\mathbb{E}(A^n_{T,f}) = \sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \cdot \Bigg((1 - (1 - p^{\text{to}}_T(\sigma a))^{c_T(\sigma a)n}) \cdot 1 + \sum_{k=0}^{c_T(\sigma a)n} \binom{c_T(\sigma a)n}{k} p^{\text{to}}_T(\sigma)^k (1 - p^{\text{to}}_T(\sigma))^{c_T(\sigma a)n-k} \cdot (1 - p^{\text{br}}_T(a \mid \sigma))^k p^{\text{cov}}_T(\sigma a, k) \Bigg)$$

Proof. Note that for test suites containing just one test case we have $c_T(\sigma a) = 1$. As expected, the formula that way immediately degenerates to the formula for the expected actual coverage of test cases. Therefore, the proof is also quite similar to the proof of Theorem 7.5.

First observe that

$$\sum_{\substack{E=(e^{t_1}_1,\ldots,e^{t_k}_n) \in exec^n_{t_1} \times \cdots \times exec^n_{t_k} \\ \sigma a \sqsubseteq_\exists E}} p^{\text{cov}}_T(\sigma a, E) \cdot \prod_{i=1}^{n} \mathbb{P}[X^n_{t_i} = (e^{t_i}_1, \ldots, e^{t_i}_n)]$$

{Def. of p^{cov}, $\sigma a \sqsubseteq_\exists E$}

$$= \sum_{\substack{E=(e^{t_1}_1,\ldots,e^{t_k}_n) \in exec^n_{t_1} \times \cdots \times exec^n_{t_k} \\ \sigma a \sqsubseteq_\exists E}} \prod_{i=1}^{n} \mathbb{P}[X^n_{t_i} = (e^{t_i}_1, \ldots, e^{t_i}_n)]$$

{Complementary probability}

$$= 1 - \sum_{\substack{E=(e^{t_1}_1,\ldots,e^{t_k}_n) \in exec^n_{t_1} \times \cdots \times exec^n_{t_k} \\ \sigma a \sqsubseteq_\nexists E}} \prod_{i=1}^{n} \mathbb{P}[X^n_{t_i} = (e^{t_i}_1, \ldots, e^{t_i}_n)]$$

{Basic probability theory}

$$= 1 - \mathbb{P}[\sigma a \sqsubseteq_\nexists X^n_{t_1} \wedge \cdots \wedge \sigma a \sqsubseteq_\nexists X^n_{t_k}]$$

{Independence of X^n_k}

$$= 1 - \prod_{i=1}^{k} \mathbb{P}[\sigma a \sqsubseteq_\nexists X^n_{t_i}]$$

{Prop. 5.7}

$$= 1 - \prod_{i=1}^{k} (\mathbb{P}[\sigma a \sqsubseteq_{\not\sharp} X_{t_i}])^n$$

{Splitting the product}

$$= 1 - \prod_{\substack{t_i \in T \\ \sigma a \in t_i}} (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_i}])^n \cdot \prod_{\substack{t_i \in T \\ \sigma a \notin t_i}} (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_i}])^n$$

{Def. of p^{to}}

$$= 1 - \prod_{\substack{t_i \in T \\ \sigma a \in t_i}} \left(1 - p^{\text{to}}_{t_i}(\sigma a)\right)^n \cdot \prod_{\substack{t_i \in T \\ \sigma a \notin t_i}} \left(1 - p^{\text{to}}_{t_i}(\sigma a)\right)^n$$

{$\sigma a \notin t_i$ implies $p^{\text{to}}_{t_i}(\sigma a) = 0$}

$$= 1 - \prod_{\substack{t_i \in T \\ \sigma a \in t_i}} \left(1 - p^{\text{to}}_{t_i}(\sigma a)\right)^n$$

{Def. 8.2}

$$= 1 - \prod_{\substack{t_i \in T \\ \sigma a \in t_i}} \left(1 - p^{\text{to}}_{T}(\sigma a)\right)^n$$

{Def. 8.12}

$$= 1 - \left(1 - p^{\text{to}}_{T}(\sigma a)\right)^{c_T(\sigma a) n}$$

Furthermore, observe the following equalities. In this proof, we assume without loss of generality that the $c_T(\sigma a)$ test cases covering σa are $t_1, \ldots, t_{c_T(\sigma a)}$. For brevity, $c_T(\sigma a)$ will be abbreviated as c. Moreover, we assume that $c_T(\sigma a) > 0$.

$$\sum_{\substack{E=(e_1^{t_1},\ldots,e_n^{t_k}) \in exec^n_{t_1} \times \cdots \times exec^n_{t_k} \\ \sigma a \sqsubseteq_{\not\sharp} E}} p^{\text{cov}}_T(\sigma a, E) \cdot \prod_{i=1}^{n} \mathbb{P}[X^n_{t_i} = (e_1^{t_i}, \ldots, e_n^{t_i})]$$

{The test cases $t_{c+1} \ldots t_k$ can be dropped, because (1) $\sigma a \sqsubseteq_{\not\sharp} E$ does not restrict them by definition of c_T and the ordering assumption, and (2) they do not change $p^{\text{cov}}_T(\sigma a, E)$ by definition of p^{cov}_T. The assumption that $c > 0$ is used as well.}

$$= \sum_{\substack{E=(e_1^{t_1},\ldots,e_n^{t_c}) \in exec^n_{t_1} \times \cdots \times exec^n_{t_c} \\ \sigma a \sqsubseteq_{\not\sharp} E}} p^{\text{cov}}_T(\sigma a, E) \cdot \prod_{i=1}^{c} \mathbb{P}[X^n_{t_i} = (e_1^{t_i}, \ldots, e_n^{t_i})]$$

{Independence and equivalence of $X^n_{t_i}$ (Lemma 8.3)}

$$= \sum_{\substack{E=(e_1,\ldots,e_{cn}) \in exec^{cn}_{t_1} \\ \sigma a \sqsubseteq_{\not\sharp} E}} p^{\text{cov}}_T(\sigma a, E) \cdot \mathbb{P}[X^{cn}_{t_1} = (e_1, \ldots, e_{cn})]$$

{Partitioning based on the number of e_i that cover σa}

$$= \sum_{k=0}^{cn} \left(\sum_{\substack{E=(e_1,\ldots,e_{cn})\in exec_{t_1}^{cn} \\ \sigma a \sqsubseteq\!\!\!\!\!\!\neq\ E \\ |\{i|\sigma\sqsubseteq e_i\}|=k}} p_T^{\text{cov}}(\sigma a, E) \cdot \mathbb{P}[X_{t_1}^{cn} = (e_1, \ldots, e_{cn})] \right)$$

{Cor. 8.6, Def. 8.7}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\sum_{\substack{E=(e_1,\ldots,e_{cn})\in exec_{t_1}^{cn} \\ \sigma a \sqsubseteq\!\!\!\!\!\!\neq\ E \\ |\{i|\sigma\sqsubseteq e_i\}|=k}} \mathbb{P}[X_{t_1}^{cn} = (e_1, \ldots, e_{cn})] \right)$$

{Independence of $e_1, \ldots, e_{cn}$}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} \sum_{\substack{E=(e_1,\ldots,e_{cn})\in exec_{t_1}^{cn} \\ \sigma a \sqsubseteq\!\!\!\!\!\!\neq\ E \\ \sigma\sqsubseteq e_1,\ldots,\sigma\sqsubseteq e_k \\ \sigma\not\sqsubseteq e_{k+1},\ldots,\sigma\not\sqsubseteq e_{cn}}} \mathbb{P}[X_{t_1}^{cn} = (e_1, \ldots, e_{cn})] \right)$$

{Basic probability theory}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} \mathbb{P}[X_{t_1}^{cn} = (e_1, \ldots, e_{cn}) \text{ such that } \forall e_i : \sigma a \not\sqsubseteq e_i \wedge \sigma \sqsubseteq e_1, \ldots, \sigma \sqsubseteq e_k \wedge \sigma \not\sqsubseteq e_{k+1}, \ldots, \sigma \not\sqsubseteq e_{cn}] \right)$$

{Prop. 5.7}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_1} \wedge \sigma \sqsubseteq X_{t_1}])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_1} \wedge \sigma \not\sqsubseteq X_{t_1}])^{cn-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} (\mathbb{P}[\sigma \sqsubseteq X_{t_1}] - \mathbb{P}[\sigma a \sqsubseteq X_{t_1} \wedge \sigma \sqsubseteq X_{t_1}])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_1} \wedge \sigma \not\sqsubseteq X_{t_1}])^{cn-k} \right)$$

{Def. of conditional probabilities}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} (\mathbb{P}[\sigma \sqsubseteq X_{t_1}] - \mathbb{P}[\sigma a \sqsubseteq X_{t_1} \mid \sigma \sqsubseteq X_{t_1}] \cdot \mathbb{P}[\sigma \sqsubseteq X_{t_1}])^k \cdot (\mathbb{P}[\sigma a \not\sqsubseteq X_{t_1} \wedge \sigma \not\sqsubseteq X_{t_1}])^{cn-k} \right)$$

{Basic rewriting, $\sigma \not\sqsubseteq X_{t_1}$ implies $\sigma a \not\sqsubseteq X_{t_1}$}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} (\mathbb{P}[\sigma \sqsubseteq X_{t_1}] - \mathbb{P}[\sigma a \sqsubseteq X_{t_1} \mid \sigma \sqsubseteq X_{t_1}] \cdot \mathbb{P}[\sigma \sqsubseteq X_{t_1}])^k \cdot (1 - \mathbb{P}[\sigma \sqsubseteq X_{t_1}])^{cn-k} \right)$$

{Definition of p^{to} and p^{br}}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} \left(p_{t_1}^{\text{to}}(\sigma) - p_{t_1}^{\text{br}}(a \mid \sigma) p_{t_1}^{\text{to}}(\sigma)\right)^k \left(1 - p_{t_1}^{\text{to}}(\sigma)\right)^{cn-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{cn} p_T^{\text{cov}}(\sigma a, k) \left(\binom{cn}{k} p_{t_1}^{\text{to}}(\sigma)^k (1 - p_{t_1}^{\text{br}}(a \mid \sigma))^k (1 - p_{t_1}^{\text{to}}(\sigma))^{cn-k} \right)$$

{Basic rewriting}

$$= \sum_{k=0}^{cn} \binom{cn}{k} p_{t_1}^{\text{to}}(\sigma)^k (1 - p_{t_1}^{\text{to}}(\sigma))^{cn-k} (1 - p_{t_1}^{\text{br}}(a \mid \sigma))^k p_T^{\text{cov}}(\sigma a, k)$$

{Def. 8.2, Def. 8.4}

$$= \sum_{k=0}^{cn} \binom{cn}{k} p_T^{\text{to}}(\sigma)^k (1 - p_T^{\text{to}}(\sigma))^{cn-k} (1 - p_T^{\text{br}}(a \mid \sigma))^k p_T^{\text{cov}}(\sigma a, k)$$

Now

$\mathbb{E}(A_{T,f}^n)$

{Def. 8.11}

$= \mathbb{E}(absCov_T((X_{t_1}^n, X_{t_2}^n, \ldots, X_{t_k}^n), f))$

{Def. of $\mathbb{E}$, independence of $X_{t_i}^n$}

$$\sum_{E=(e_1^{t_1},\ldots,e_n^{t_k}) \in exec_{t_1}^n \times \cdots \times exec_{t_k}^n} absCov_T(E, f) \cdot \prod_{i=1}^{n} \mathbb{P}[X_{t_i}^n = (e_1^{t_i}, \ldots, e_n^{t_i})]$$

{Def. of *absCov*, Def. of *faultCov*}

$$\sum_{E=(e_1^{t_1},\ldots,e_n^{t_k}) \in exec_{t_1}^n \times \cdots \times exec_{t_k}^n} \left(\sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \cdot p_T^{\text{cov}}(\sigma a, E) \right) \cdot \prod_{i=1}^{n} \mathbb{P}[X_{t_i}^n = (e_1^{t_i}, \ldots, e_n^{t_i})]$$

{Basic rewriting}

$$\sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \left(\sum_{E=(e_1^{t_1},\ldots,e_n^{t_k}) \in exec_{t_1}^n \times \cdots \times exec_{t_k}^n} p_T^{\text{cov}}(\sigma a, E) \cdot \prod_{i=1}^{n} \mathbb{P}[X_{t_i}^n = (e_1^{t_i}, \ldots, e_n^{t_i})] \right)$$

{Splitting the summation}

$$\sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \left(\sum_{\substack{E=(e_1^{t_1},\ldots,e_n^{t_k}) \in exec_{t_1}^n \times \cdots \times exec_{t_k}^n \\ \sigma a \sqsubseteq_\exists E}} p_T^{\text{cov}}(\sigma a, E) \cdot \prod_{i=1}^{n} \mathbb{P}[X_{t_i}^n = (e_1^{t_i}, \ldots, e_n^{t_i})] + \right.$$

$$\left. \sum_{\substack{E=(E_1,\ldots,E_k) \in exec_{t_1}^n \times \cdots \times exec_{t_k}^n \\ \sigma a \sqsubseteq_\nexists E}} p_T^{\text{cov}}(\sigma a, E) \cdot \prod_{i=1}^{n} \mathbb{P}[X_{t_i}^n = (e_1^{t_i}, \ldots, e_n^{t_i})] \right)$$

{The equalities derived above}

$$= \sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \cdot \Bigg((1 - (1 - p_T^{\text{to}}(\sigma a))^{c_T(\sigma a)n}) \cdot 1 + \sum_{k=0}^{c_T(\sigma a)n} \binom{c_T(\sigma a)n}{k} p_T^{\text{to}}(\sigma)^k (1 - p_T^{\text{to}}(\sigma))^{c_T(\sigma a)n-k} \cdot (1 - p_T^{\text{br}}(a \mid \sigma))^k p_T^{\text{cov}}(\sigma a, k) \Bigg)$$

Note that the earlier assumption $c_T(\sigma a) > 0$ indeed holds, since we only sum over traces σa that are in at least one test case. □

It is not difficult to see that the complexity is comparable to the complexity of the formula for a single test case. In this case it is in $O((sn)^2 + msn \log(sn))$, with $s = |T|$ and $m = |\bigcup_{t_i} err_{t_i}|$.

Example 8.14. To demonstrate the applicability of Theorem 8.13, we calculate the expected actual coverage of three executions of the test suite T of Figure 8.1. Using Theorem 8.13, we obtain

$$\mathbb{E}(A_{T,f}^3) = \sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \cdot \Bigg((1 - (1 - p_T^{\text{to}}(\sigma a))^{3c_T(\sigma a)}) \cdot 1 + \sum_{k=0}^{3c_T(\sigma a)} \binom{3c_T(\sigma a)}{k} p_T^{\text{to}}(\sigma)^k (1 - p_T^{\text{to}}(\sigma))^{3c_T(\sigma a)-k} \cdot (1 - p_T^{\text{br}}(a \mid \sigma))^k p_T^{\text{cov}}(\sigma a, k) \Bigg)$$

For layout purposes, we only show one term of this summation; the one corresponding to the fault *a? c!*. Obviously, all others are very similar.

$$
\begin{aligned}
f(a?\ c!) \cdot \Bigg(& (1-(1-p_T^{\mathrm{to}}(a?\ c!))^{3c_T(a?\ c!)}) + \\
& \sum_{k=0}^{3c_T(a?\ c!)} \binom{3c_T(a?\ c!)}{k} p_T^{\mathrm{to}}(a?)^k (1-p_T^{\mathrm{to}}(a?))^{3c_T(a?\ c!)-k} \cdot \\
& (1-p_T^{\mathrm{br}}(c! \mid a?))^k p_T^{\mathrm{cov}}(a?\ c!, k) \Bigg) \\
= 5 \cdot \Bigg(& 1 - 0.9^6 + \sum_{k=0}^{6} \binom{6}{k} 1^k \cdot 0^{6-k} \cdot 0.9^k \cdot (1-0.8^k) \Bigg) = 4.3034
\end{aligned}
$$

Calculating the other terms similarly, we obtain

$$\mathbb{E}(A^3_{T,f}) = 4.3034 + 8.6639 + 3.3143 + 3.0377 + 2.4117 = 21.731$$

The following theorem states that the absolute actual coverage of an infinite number of test suite executions is equal to its absolute potential coverage.

Theorem 8.15. *Let $T = (t_1, \ldots, t_k)$ be a test suite for an LTS $\mathcal{A}$ and f a weighted fault model consistent with $\mathcal{A}$. Let $n \in \mathbb{N}$ be the number of times T is executed. If for all erroneous traces $\sigma a!$ it holds that $p_T^{\mathrm{to}}(\sigma a!) > 0$, then*

$$\lim_{n\to\infty} \mathbb{E}(A^n_{T,f}) = absPotCov(T, f)$$

Proof. Since $c_T(\sigma a)$ is positive for each $\sigma a \in \bigcup err_{t_i}$, we have

$$
\begin{aligned}
\lim_{n\to\infty} \Bigg(& (1-(1-p_T^{\mathrm{to}}(\sigma a))^{c_T(\sigma a)n}) \cdot 1 + \\
& \sum_{k=0}^{c_T(\sigma a)n} \binom{c_T(\sigma a)n}{k} p_T^{\mathrm{to}}(\sigma)^k (1-p_T^{\mathrm{to}}(\sigma))^{c_T(\sigma a)n-k} \cdot \\
& (1-p_T^{\mathrm{br}}(\sigma)(a))^k p_T^{\mathrm{cov}}(\sigma a, k) \Bigg) \\
= \lim_{n\to\infty} \Bigg(& (1-(1-p_T^{\mathrm{to}}(\sigma a))^{n}) \cdot 1 + \\
& \sum_{k=0}^{n} \binom{n}{k} p_T^{\mathrm{to}}(\sigma)^k (1-p_T^{\mathrm{to}}(\sigma))^{n-k} \cdot \\
& (1-p_T^{\mathrm{br}}(\sigma)(a))^k p_T^{\mathrm{cov}}(\sigma a, k) \Bigg)
\end{aligned}
$$

Combining this observation with the proof of Theorem 7.7, we immediately obtain

$$
\begin{aligned}
\lim_{n\to\infty} \mathbb{E}(A^n_{T,f}) &= \sum_{\sigma a \in \bigcup err_{t_i}} f(\sigma a) \\
&= absPotCov(T, f)
\end{aligned}
$$

□

Chapter 9

A detailed example

To see the concepts of the previous chapters come to life, this chapter provides a detailed example. It is meant to demonstrate how to work with all the concepts developed in the previous chapters.

We start from a system specification, for which we provide error weights and estimate presence probabilities and conditional branching probabilities.

We consider two implementations, providing us with the flawless branching probabilities. A probabilistic execution model is derived using this information.

Then, two test cases and a test suite are derived, and their potential coverage as well as their expected actual coverage is calculated. We demonstrate how to calculate actual coverage for both a single execution and a sequence of executions. To get a feeling of the process of executing test cases and test suites, we simulate several executions using Matlab, and calculate the actual coverage of the resulting traces. Moreover, we perform many simulations to show how close our prediction of the expected actual coverage approaches the results obtained by (simulated) executions of implementations.

As the current example is still theoretical, an extensive future study applying actual coverage to real software projects would be necessary to fully show the usability, sensitivity and accuracy of our methods.

Organisation of this chapter
We first give a system specification and the corresponding probabilities in Section 9.1. Then, two implementations and their PFA are discussed in Section 9.2. Section 9.3 provides test cases and calculates their potential and expected actual coverage. Finally, Section 9.4 demonstrates the evaluation of actual coverage and simulates test case executions.

9.1 The specification of a simple system

Our example is aimed at a (fictional) chemical dispenser. It dispenses two chemical substances, for notational ease denoted by x and y.

The chemical x is produced by mixing ammonium (a) with barium (b) and adding some more ammonium afterwards. A prefabricated concentrated emulsion of ammonium and barium (e) might also be used, although it should directly

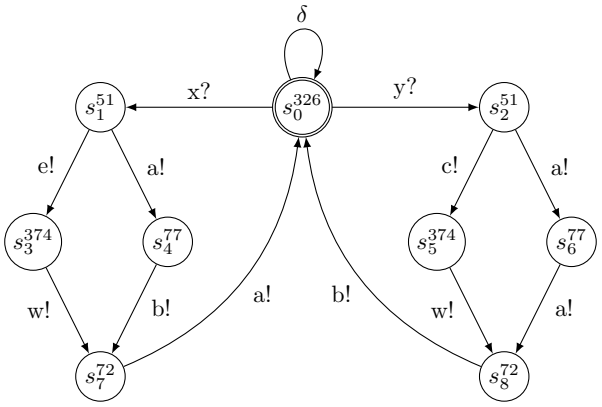


Figure 9.1: The system $\mathcal{A}$

be thinned out by adding water (w).

The chemical y is produced by first mixing two quantities of ammonium, and then adding some barium. A concentrated version of ammonium (c) might also be used, although it should directly be thinned out by adding water as well.

The system is represented by the LTS $\mathcal{A}$ given in Figure 9.1. Formally, $\mathcal{A} = \langle S, s_0, L, \Delta \rangle$ with $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ and $L = \{x?, y?, a!, b!, c!, e!, w!\}$. Δ contains among others the transitions $(s_0, x?, s_1)$ and $(s_1, e!, s_3)$.

We assume that the presence probability of each erroneous trace is 0.05. All faults are assumed to have a conditional branching probability of 0.25.

Since the chemical compounds e and c are concentrated and as such dangerous, it is very important that they are thinned out by adding water immediately. Therefore, from s_3 and s_5 all actions other than adding water have been given a high severity. Based on this information, the outputs e and c from s_0 are also considered severe. Every other error is of a significantly lower severity.

The error weights $r(s_i, a)$ for all states $s_i \in S$ and actions $a \in L^O \cup \{\delta\}$ are shown in Table 9.1. It also shows the accumulated error weights $\bar{r}$ (according to Definition 2.23). These accumulated error weights have been included in Figure 9.1 as superscripts of the state names. For simplicity, we ignore discounting by assuming a finite depth weighted fault model f based on these error weights, with k larger than the depth of the test cases.

r	$a!$	$b!$	$e!$	$c!$	$w!$	δ	$\bar{r}$
s_0	10	8	150	150	8	0	326
s_1	0	17	0	12	11	11	51
s_2	0	19	12	0	9	11	51
s_3	85	60	67	91	0	71	374
s_4	15	0	12	18	11	21	77
s_5	85	60	91	67	0	71	374
s_6	0	13	17	15	11	21	77
s_7	0	13	8	25	18	8	72
s_8	17	0	14	21	13	7	72

Table 9.1: Error weights for $\mathcal{A}$

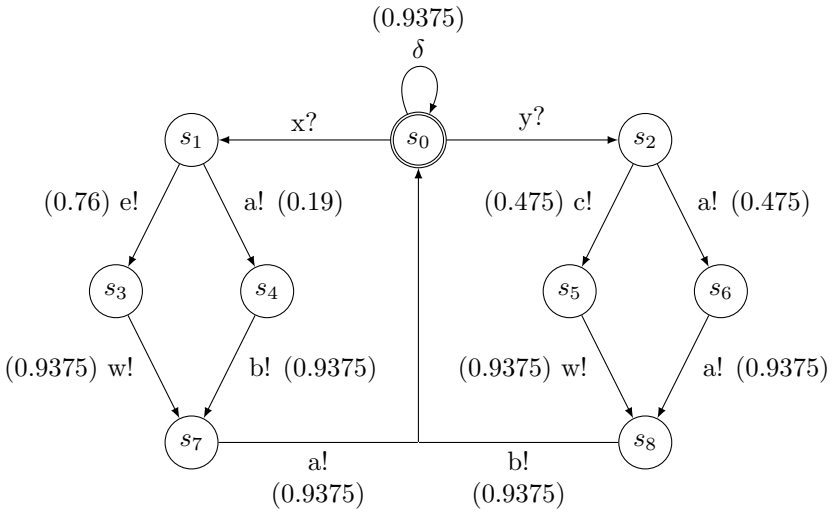


Figure 9.2: A PFA of $\mathcal{A}$

9.2 Implementations of the system

Given an implementation of $\mathcal{A}$, the flawless branching probabilities can be estimated or might even be known. We consider implementations that choose *e!* instead of *a! b!* with a probability of 0.8, because there is more of the concentrated ammonium–barium emulsion in stock than of the individual substances. For the production of y there is an identical probability of choosing two amounts of normal ammonium or the thinned out concentrated version.

Together with the estimation that all conditional branching probabilities are equal to 0.25 and all presence probabilities to 0.05, we can also derive the branching probabilities using Proposition 5.36. As a result, we obtain the PFA shown in Figure 9.2.

For layout purposes we only show the values of p^{br} for the correct outputs, and omit the error weights, conditional branching probabilities and the values of p^{br} for the erroneous outputs. The error weights and the conditional branching probabilities were already given textually, and the branching probabilities of the erroneous traces are all equal to $0.05 \cdot 0.25 = 0.0125$.

We consider two erroneous implementations: $\mathcal{I}_1$ and $\mathcal{I}_2$. For $\mathcal{I}_1$, there is a probability of 0.25 that the system provides ammonium after it has provided the ammonium–barium emulsion. Furthermore, there is also a probability of 0.25 that nothing happens after the first two steps of producing the chemical y. Figure 9.3 shows this behaviour as an LTS.

For $\mathcal{I}_2$, the first request always results in the correct chemical. From the second request on the behaviour is identical to $\mathcal{I}_1$. Figure 9.4 shows this behaviour as an LTS.

Note that the information assumed above would obviously not be known in practice, but is necessary here to simulate executions.

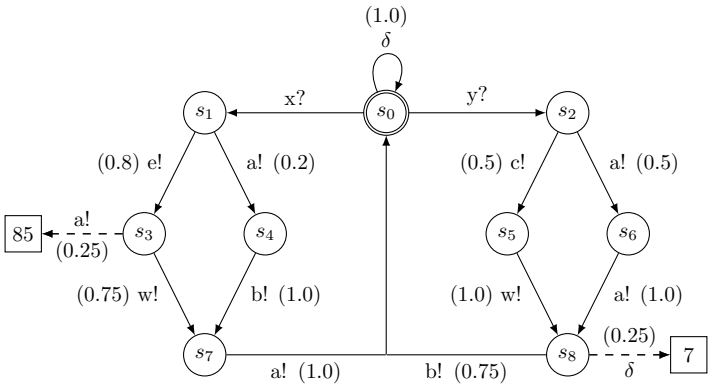


Figure 9.3: The actual behaviour of an implementation $\mathcal{I}_1$

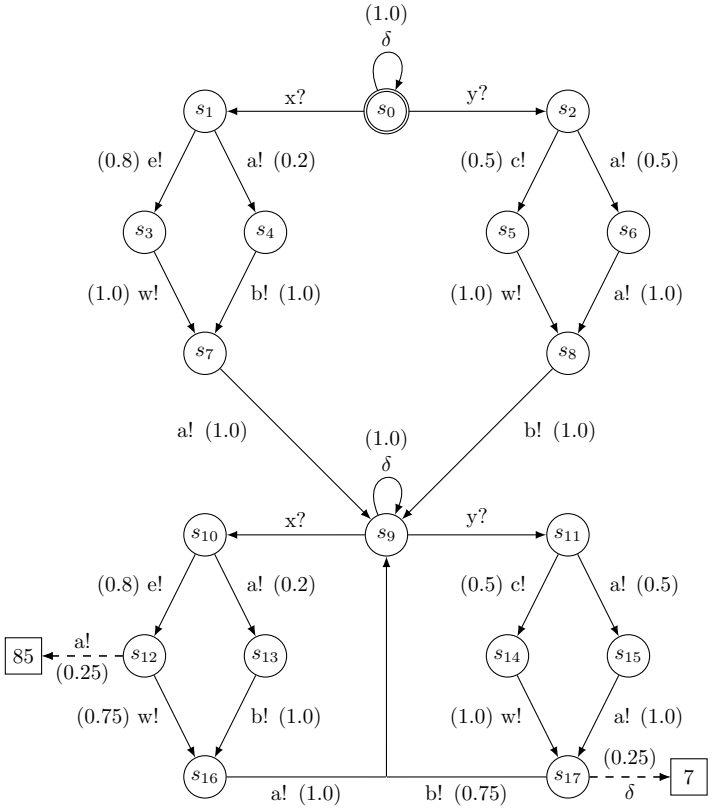
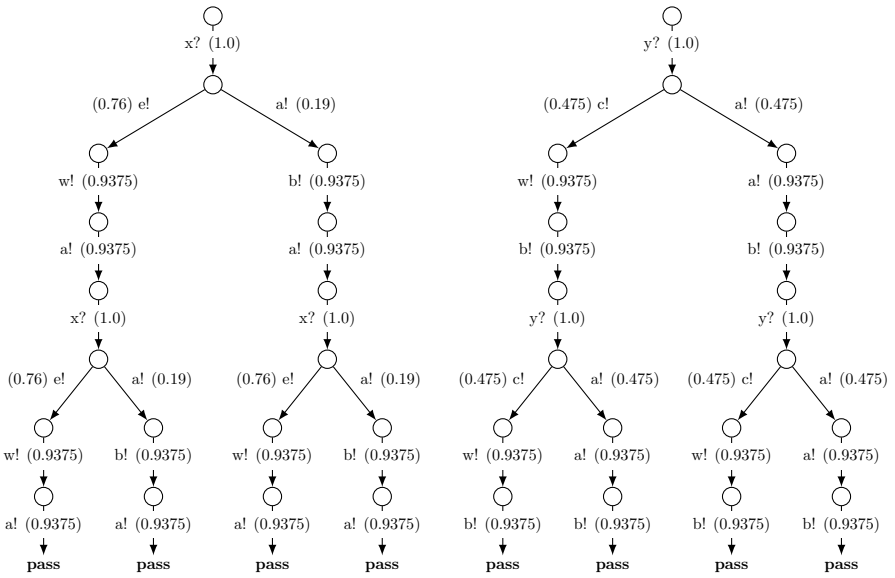


Figure 9.4: The actual behaviour of an implementation $\mathcal{I}_2$



(a) The test case t_1 (b) The test case t_2

Figure 9.5: Test cases for $\mathcal{A}$

9.3 Test cases for the implementations

To test the system $\mathcal{A}$, two test cases (t_1 and t_2) have been derived. They are shown in Figure 9.5, although for layout purposes only the correct traces have been included. The omitted erroneous traces can easily be obtained from the specification, and their error weights were already given in Table 9.1. We will also consider the test suite T, consisting of three test cases: t_1 twice, and t_2 once. Formally, $T = (t_1, t_1, t_2)$.

9.3.1 Calculating potential coverage

We start by calculating the potential coverage of both test cases and the test suite. Note that for every trace σ of the test case where the next step is observing, the potential coverage is increased by $\bar{r}(s)$ with $s = \mathit{final}(\sigma)$. Therefore

$$\begin{aligned}
\mathit{absPotCov}(t_1, f) &= \bar{r}(\mathit{final}(x?)) + \bar{r}(\mathit{final}(x?\ e!)) + \bar{r}(\mathit{final}(x?\ a!)) \\
&\quad + \bar{r}(\mathit{final}(x?\ e!\ w!)) + \bar{r}(\mathit{final}(x?\ e!\ w!\ a!\ x?)) \\
&\quad + \bar{r}(\mathit{final}(x?\ e!\ w!\ a!\ x?\ e!)) + \bar{r}(\mathit{final}(x?\ e!\ w!\ a!\ x?\ e!\ w!)) \\
&\quad + \bar{r}(\mathit{final}(x?\ e!\ w!\ a!\ x?\ a!)) + \bar{r}(\mathit{final}(x?\ e!\ w!\ a!\ x?\ a!\ b!)) \\
&\quad + \bar{r}(\mathit{final}(x?\ a!\ b!)) + \bar{r}(\mathit{final}(x?\ a!\ b!\ a!\ x?)) \\
&\quad + \bar{r}(\mathit{final}(x?\ a!\ b!\ a!\ x?\ e!)) + \bar{r}(\mathit{final}(x?\ a!\ b!\ a!\ x?\ e!\ w!)) \\
&\quad + \bar{r}(\mathit{final}(x?\ a!\ b!\ a!\ x?\ a!)) + \bar{r}(\mathit{final}(x?\ a!\ b!\ a!\ x?\ a!\ b!)) \\
&= \bar{r}(s_1) + \bar{r}(s_3) + \bar{r}(s_4) + \bar{r}(s_7) + \bar{r}(s_1) + \bar{r}(s_3) + \bar{r}(s_7) + \bar{r}(s_4) \\
&\quad + \bar{r}(s_7) + \bar{r}(s_7) + \bar{r}(s_1) + \bar{r}(s_3) + \bar{r}(s_7) + \bar{r}(s_4) + \bar{r}(s_7)
\end{aligned}$$

$$
\begin{aligned}
&= 51 + 374 + 77 + 72 + 51 + 374 + 72 + 77 \\
&+ 72 + 72 + 51 + 374 + 72 + 77 + 72 \\
&= 1938
\end{aligned}
$$

$$
\begin{aligned}
absPotCov(t_2, f) &= \bar{r}(final(y?)) + \bar{r}(final(y?\ c!)) + \bar{r}(final(y?\ a!)) \\
&+ \bar{r}(final(y?\ c!\ w!)) + \bar{r}(final(y?\ c!\ w!\ b!\ y?)) \\
&+ \bar{r}(final(y?\ c!\ w!\ b!\ y?\ c!)) + \bar{r}(final(y?\ c!\ w!\ b!\ y?\ c!\ w!)) \\
&+ \bar{r}(final(y?\ c!\ w!\ b!\ y?\ a!)) + \bar{r}(final(y?\ c!\ w!\ b!\ y?\ a!\ a!)) \\
&+ \bar{r}(final(y?\ a!\ a!)) + \bar{r}(final(y?\ a!\ a!\ b!\ y?)) \\
&+ \bar{r}(final(y?\ a!\ a!\ b!\ y?\ c!)) + \bar{r}(final(y?\ a!\ a!\ b!\ y?\ c!\ w!)) \\
&+ \bar{r}(final(y?\ a!\ a!\ b!\ y?\ a!)) + \bar{r}(final(y?\ a!\ a!\ b!\ y?\ a!\ a!)) \\
&= \bar{r}(s_2) + \bar{r}(s_5) + \bar{r}(s_6) + \bar{r}(s_8) + \bar{r}(s_2) + \bar{r}(s_5) + \bar{r}(s_8) + \bar{r}(s_6) \\
&+ \bar{r}(s_8) + \bar{r}(s_8) + \bar{r}(s_2) + \bar{r}(s_5) + \bar{r}(s_8) + \bar{r}(s_6) + \bar{r}(s_8) \\
&= 51 + 374 + 77 + 72 + 51 + 374 + 72 + 77 \\
&+ 72 + 72 + 51 + 374 + 72 + 77 + 72 \\
&= 1938
\end{aligned}
$$

Apparently, the potential coverage of both test cases is equal. Since the error weights were chosen symmetrically for traces starting with an $x?$ and traces starting with a $y?$, this was also expected.

For the test suite, we have to take the union of all erroneous traces in its test cases and calculate the potential coverage of the resulting trace set. Since t_1 and t_2 are disjoint this can easily be seen equal to $absPotCov(t_1, f) + absPotCov(t_2, f) = 1938 + 1938 = 3876$.

9.3.2 Calculating expected actual coverage

Before simulating test executions, we first calculate the expected actual coverage of both test cases and the test suite for $1, 5, 10, 50$ and 250 executions. Recall that the following formula was derived in Theorem 7.5.

$$
\begin{aligned}
\mathbb{E}(A^n_{t,f}) = \sum_{\sigma a \in err_t} f(\sigma a) \cdot \Bigg(& (1 - (1 - p^{\text{to}}_t(\sigma a))^n) \cdot 1 + \\
& \sum_{k=0}^{n} \binom{n}{k} p^{\text{to}}_t(\sigma)^k (1 - p^{\text{to}}_t(\sigma))^{n-k} \cdot \\
& (1 - p^{\text{br}}_t(a \mid \sigma))^k p^{\text{cov}}_t(\sigma a, k) \Bigg)
\end{aligned}
$$

Note that this calculation uses several probabilities, each of which can be obtained from the information provided thus far.

As an example we discuss the contribution of the erroneous trace $\sigma a = x?\ e!\ a!$ of t_1, for $n = 5$. For this trace, we have

$$
\begin{aligned}
f(x?\ e!\ a!) &= r(final(x?\ e!), a!) = r(s_3, a!) = 85 \\
p^{\text{to}}(x?\ e!\ a!) &= 1.0 \cdot 0.76 \cdot 0.0125 = 0.0095
\end{aligned}
$$

$$p^{\mathrm{to}}(x?\ e!) = 1.0 \cdot 0.76 = 0.76$$
$$p^{\mathrm{br}}(a! \mid x?\ e!) = 0.0125$$
$$p^{\mathrm{cov}}(x?\ e!\ a!, k) = 1 - (1 - p^{\mathrm{cbr}}(a! \mid x?\ e!))^k = 1 - (1 - 0.25)^k$$

Note that $f(x?\ e!\ a!)$ was calculated using Definition 2.25, p^{to} using Proposition 5.15 and p^{cov} using Proposition 6.6. We can now calculate the contribution of the erroneous trace $x?\ e!\ a!$:

$$\begin{aligned} & f(x?\ e!\ a!) \cdot \Bigg((1 - (1 - p_t^{\mathrm{to}}(x?\ e!\ a!))^n) \cdot 1 + \\ & \qquad \sum_{k=0}^{n} \binom{n}{k} p_t^{\mathrm{to}}(x?\ e!)^k (1 - p_t^{\mathrm{to}}(x?\ e!))^{n-k} \cdot \\ & \qquad\qquad (1 - p_t^{\mathrm{br}}(a! \mid x?\ e!))^k p_t^{\mathrm{cov}}(x?\ e!\ a!, k) \Bigg) \\ = 85 & \left(1 - (1 - 0.0095)^n + \sum_{k=0}^{5} \binom{5}{k} 0.76^k (1 - 0.76)^{5-k} (1 - 0.0125)^k (1 - 0.25^k) \right) \\ = 56.6 & \end{aligned}$$

Performing similar calculations for all other faults and summing the results, we obtain $\mathbb{E}(A^5_{t_1,f})$. Doing this also for different values of n and for t_2, we obtain the first two columns of Table 9.2.

For the test suite, the formula from Theorem 8.13 can be used. Note that it sums over faults potentially covered by the test suite, applying almost the same calculation as was the case for test cases. The only difference is that instead of using n, we use $c_T(\sigma a)n$.

Since t_1 and t_2 are disjoint, the contribution to $\mathbb{E}(A^n_{T,f})$ of the faults in t_2 is equal to their contribution to $\mathbb{E}(A^n_{t_2,f})$. Furthermore, the contribution of the faults in t_1 is just equal to their contribution to $\mathbb{E}(A^{2n}_{t_1,f})$, since $c_T(\sigma a) = 2$ for all $\sigma a \in t_1$. This results in the observation that $\mathbb{E}(^n_{T,f}) = \mathbb{E}(^{2n}_{t_1,f}) + \mathbb{E}(^n_{t_2,f})$. Using this formula, the expected actual coverage for the test suite executions is given in the last column of Table 9.2. Observe that indeed $\mathbb{E}(A^5_{T,f}) = \mathbb{E}(A^{10}_{t_1,f}) + \mathbb{E}(A^5_{t_2,f})$.

Figure 9.6 shows a graph of the expected actual coverage of both t_1, t_2, and T, for n = 1 ... 100. The potential coverage of the test cases (1938) and of the test suite (3876) have also been indicated in the graph. We can see that t_1, t_2,

$\mathbb{E}(A^1_{t_1,f})$	=	197.0	$\mathbb{E}(A^1_{t_2,f})$	=	156.8	$\mathbb{E}(A^1_{T,f})$	=	520.2
$\mathbb{E}(A^5_{t_1,f})$	=	729.1	$\mathbb{E}(A^5_{t_2,f})$	=	639.8	$\mathbb{E}(A^5_{T,f})$	=	1716.7
$\mathbb{E}(A^{10}_{t_1,f})$	=	1076.9	$\mathbb{E}(A^{10}_{t_2,f})$	=	1032.1	$\mathbb{E}(A^{10}_{T,f})$	=	2423.8
$\mathbb{E}(A^{50}_{t_1,f})$	=	1704.6	$\mathbb{E}(A^{50}_{t_2,f})$	=	1848.0	$\mathbb{E}(A^{50}_{T,f})$	=	3695.3
$\mathbb{E}(A^{250}_{t_1,f})$	=	1917.7	$\mathbb{E}(A^{250}_{t_2,f})$	=	1938.0	$\mathbb{E}(A^{250}_{T,f})$	=	3873.3

Table 9.2: Prediction of actual coverage for t_1, t_2 and T

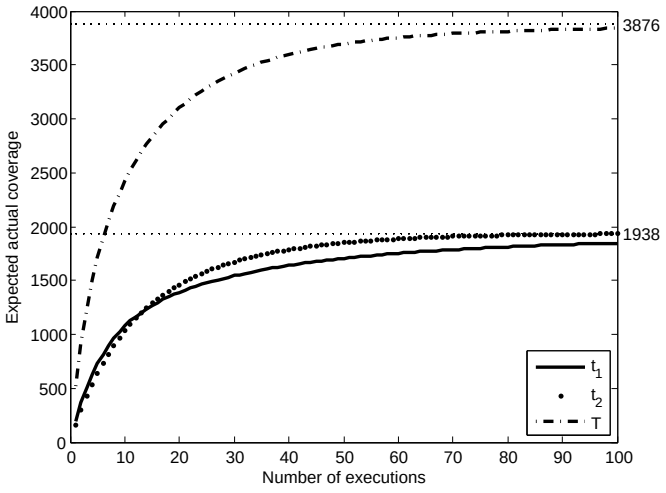


Figure 9.6: Expected actual coverage of t_1, t_2, and T

and T indeed approach their potential coverage for a large number of executions (as was also proved mathematically).

Interestingly, it turns out that t_1 performs better than t_2 if the test case is to be executed thirteen times or less, while t_2 is better if it is executed more often.

The fact that t_1 has a higher expected actual coverage than t_2 for a small number of executions can be understood by observing that the probability of reaching a state from where faults with high error weights are covered (s_3 and s_5) is much larger in t_1 than in t_2. For a large number of executions, however, t_2 will on average also cover these faults many times. Moreover, it covers the states with less severe faults many times, while t_1 covers these less often. Because of the nonlinearity of p^{cov} this results in a higher expected actual coverage for t_2.

9.4 Executing the test cases and test suite

We illustrate the calculation used to obtain the actual coverage of an execution, by performing it for $x?\ e!\ w!\ a!\ x?\ e!\ a!$. Note that by Definition 6.3 and Proposition 6.6 we have

$$p_t^{\text{cov}}(\sigma m, x?\ e!\ w!\ a!\ x?\ e!\ a!) = \begin{cases} 1 & , \text{ if } \sigma m = x?\ e!\ w!\ a!\ x?\ e!\ a! \\ 1-(1-0.25)^1 = 0.25 & , \text{ if } \sigma \sqsubseteq x?\ e!\ w!\ a!\ x?\ e!\ a! \\ 0 & , \text{ otherwise} \end{cases}$$

Using this observation, we obtain

$$\begin{aligned} & absCov(x?\ e!\ w!\ a!\ x?\ e!\ a!, f) \\ &= \sum_{\sigma m! \in err_t} faultCov(\sigma m!, x?\ e!\ w!\ a!\ x?\ e!\ a!, f) \\ &= \sum_{\sigma m! \in err_t} f(\sigma m!) \cdot p^{\text{cov}}(\sigma m!, x?\ e!\ w!\ a!\ x?\ e!\ a!) \end{aligned}$$

$$
\begin{aligned}
&= (f(x?\ b!) + f(x?\ c!) + f(x?\ w!) + f(x?\ \delta)) \cdot 0.25 \\
&+ (f(x?\ e!\ a!) + f(x?\ e!\ b!) + f(x?\ e!\ e!) + f(x?\ e!\ c!) + f(x?\ e!\ \delta)) \cdot 0.25 \\
&+ (f(x?\ e!\ w!\ b!) + f(x?\ e!\ w!\ e!) + f(x?\ e!\ w!\ c!) + f(x?\ e!\ w!\ w!) + f(x?\ e!\ w!\ \delta)) \cdot 0.25 \\
&+ (f(x?\ e!\ w!\ a!\ x?\ b!) + f(x?\ e!\ w!\ a!\ x?\ c!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ w!) + f(x?\ e!\ w!\ a!\ x?\ \delta)) \cdot 0.25 \\
&+ (f(x?\ e!\ w!\ a!\ x?\ e!\ b!) + f(x?\ e!\ w!\ a!\ x?\ e!\ e!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ e!\ c!) + f(x?\ e!\ w!\ a!\ x?\ e!\ \delta)) \cdot 0.25 + f(x?\ e!\ w!\ a!\ x?\ e!\ a!) \cdot 1.0 \\
&= (17 + 12 + 11 + 11 + 85 + 60 + 67 + 91 + 71 + 13 + 8 + 25 + 18 + 8 \\
&\quad 17 + 12 + 11 + 11 + 60 + 67 + 91 + 71) \cdot 0.25 + 85 = 294.3
\end{aligned}
$$

Besides evaluating the actual coverage of individual test executions, our framework also provides methods to evaluate the actual coverage of sequences of executions.

To demonstrate the calculations used to obtain the simulation results, we evaluate the absolute actual coverage of the following sequence of executions:

$$
\begin{aligned}
E = &\ x?\ e!\ a! \\
&\ x?\ e!\ w!\ a!\ x?\ e!\ a! \\
&\ x?\ e!\ w!\ a!\ x?\ e!\ a! \\
&\ x?\ e!\ w!\ a!\ x?\ a!\ b!\ a! \\
&\ x?\ a!\ b!\ a!\ x?\ a!\ b!\ a!
\end{aligned}
$$

We obtain

$$
\begin{aligned}
&absCov(E, f) \\
&= \sum_{\sigma m! \in err_t} faultCov(\sigma m!, E, f) \\
&= \sum_{\sigma m! \in err_t} f(\sigma m!) \cdot p^{\text{cov}}(\sigma m!, E) \\
&= (f(x?\ b!) + f(x?\ c!) + f(x?\ w!) + f(x?\ \delta)) \cdot (1 - (1 - 0.25)^5) \\
&+ (f(x?\ e!\ b!) + f(x?\ e!\ e!) + f(x?\ e!\ c!) + f(x?\ e!\ \delta)) \cdot (1 - (1 - 0.25)^4) \\
&+ (f(x?\ a!\ a!) + f(x?\ a!\ e!) + f(x?\ a!\ c!) + f(x?\ a!\ w!) + f(x?\ a!\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ e!\ w!\ b!) + f(x?\ e!\ w!\ e!) + f(x?\ e!\ w!\ c!) + f(x?\ e!\ w!\ w!) + \\
&\quad f(x?\ e!\ w!\ \delta)) \cdot (1 - (1 - 0.25)^3) \\
&+ (f(x?\ a!\ b!\ b!) + f(x?\ a!\ b!\ e!) + f(x?\ a!\ b!\ c!) + f(x?\ a!\ b!\ w!) + \\
&\quad f(x?\ a!\ b!\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ e!\ w!\ a!\ x?\ b!) + f(x?\ e!\ w!\ a!\ x?\ c!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ w!) + f(x?\ e!\ w!\ a!\ x?\ \delta)) \cdot (1 - (1 - 0.25)^3) \\
&+ (f(x?\ a!\ b!\ a!\ x?\ b!) + f(x?\ a!\ b!\ a!\ x?\ c!) + \\
&\quad f(x?\ a!\ b!\ a!\ x?\ w!) + f(x?\ a!\ b!\ a!\ x?\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ e!\ w!\ a!\ x?\ a!\ a!) + f(x?\ e!\ w!\ a!\ x?\ a!\ e!) + f(x?\ e!\ w!\ a!\ x?\ a!\ c!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ a!\ w!) + f(x?\ e!\ w!\ a!\ x?\ a!\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ a!\ b!\ a!\ x?\ a!\ a!) + f(x?\ a!\ b!\ a!\ x?\ a!\ e!) + f(x?\ a!\ b!\ a!\ x?\ a!\ c!) + \\
&\quad f(x?\ a!\ b!\ a!\ x?\ a!\ w!) + f(x?\ a!\ b!\ a!\ x?\ a!\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ e!\ w!\ a!\ x?\ a!\ b!\ b!) + f(x?\ e!\ w!\ a!\ x?\ a!\ b!\ e!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ a!\ b!\ c!) + f(x?\ e!\ w!\ a!\ x?\ a!\ b!\ w!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ a!\ b!\ \delta)) \cdot (1 - (1 - 0.25)^1)
\end{aligned}
$$

$$
\begin{aligned}
&+ (f(x?\ a!\ b!\ a!\ x?\ a!\ b!\ b!) + f(x?\ a!\ b!\ a!\ x?\ a!\ b!\ e!) + \\
&\quad f(x?\ a!\ b!\ a!\ x?\ a!\ b!\ c!) + f(x?\ a!\ b!\ a!\ x?\ a!\ b!\ w!) + \\
&\quad f(x?\ a!\ b!\ a!\ x?\ a!\ b!\ \delta)) \cdot (1 - (1 - 0.25)^1) \\
&+ (f(x?\ e!\ w!\ a!\ x?\ e!\ b!) + f(x?\ e!\ w!\ a!\ x?\ e!\ e!) + \\
&\quad f(x?\ e!\ w!\ a!\ x?\ e!\ c!) + f(x?\ e!\ w!\ a!\ x?\ e!\ \delta)) \cdot (1 - (1 - 0.25)^2) \\
&+ f(x?\ e!\ a!) \cdot 1.0 + f(x?\ e!\ w!\ a!\ x?\ e!\ a!) \cdot 1.0 \\
&= 51 \cdot (1 - (1 - 0.25)^5) + 289 \cdot (1 - (1 - 0.25)^4) + 77 \cdot (1 - (1 - 0.25)^1) \\
&+ 72 \cdot (1 - (1 - 0.25)^3) + 72 \cdot (1 - (1 - 0.25)^1) + 51 \cdot (1 - (1 - 0.25)^3) \\
&+ 51 \cdot (1 - (1 - 0.25)^1) + 77 \cdot (1 - (1 - 0.25)^1) + 77 \cdot (1 - (1 - 0.25)^1) \\
&+ 72 \cdot (1 - (1 - 0.25)^1) + 72 \cdot (1 - (1 - 0.25)^1) + 289 \cdot (1 - (1 - 0.25)^2) + 85 + 85 = 728.5
\end{aligned}
$$

9.4.1 Simulating test case executions

For both $\mathcal{I}_1$ and $\mathcal{I}_2$ we simulated 20 executions for t_1 and 20 executions for t_2. A random number generator has been used to determine which outputs the system provides, based on the probabilistic models shown in Figure 9.3 and Figure 9.4.

Table 9.3 and Table 9.4 show which executions occurred, how often they occurred and each execution's individual absolute actual coverage. We discuss the results separately for $\mathcal{I}_1$ and $\mathcal{I}_2$. Since executions of the test suite are just tuples of executions of t_1 and t_2, these are omitted from the discussion.

Executing $\mathcal{I}_1$

Dealing with $\mathcal{I}_1$, the average actual coverage of the sample for t_1 is 216.6. The fact that we obtained an average higher than $\mathbb{E}(A^1_{t_1,f}) = 197.0$ can be explained by the observation that one of the faults that were chosen to be present is coincidentally of higher error weight than the average fault. This severe fault can be detected by t_1, resulting in more actual coverage.

The average actual coverage of the sample for t_2 is 156.4; a satisfactory result given that we calculated $\mathbb{E}(A^1_{t_2,f}) = 156.8$ in advance.

Executing $\mathcal{I}_2$

Dealing with $\mathcal{I}_2$, the average actual coverage of the sample for t_1 is 228.0. For t_2, we have an average of 183.5.

The fact that we obtained averages that are even higher can be explained by the fact that we assumed for each fault a presence probability of 0.05. Therefore, on average one out of every twenty erroneous traces is assumed to be present.

Executions of t_1	#	*absCov*	Executions of t_2	#	*absCov*
$x?\ e!\ a!$	5	170.0	$y?\ c!\ w!\ \delta$	3	129.5
$x?\ e!\ w!\ a!\ x?\ e!\ a!$	6	294.3	$y?\ a!\ a!\ b!\ y?\ c!\ w!\ \delta$	1	179.5
$x?\ e!\ w!\ a!\ x?\ e!\ w!\ a!$	3	248.5	$y?\ a!\ a!\ b!\ y?\ a!\ a!\ \delta$	1	105.3
$x?\ e!\ w!\ a!\ x?\ a!\ b!\ a!$	3	174.3	$y?\ c!\ w!\ b!\ y?\ c!\ w!\ \delta$	1	253.8
$x?\ a!\ b!\ a!\ x?\ e!\ w!\ a!$	2	174.3	$y?\ c!\ w!\ b!\ y?\ a!\ a!\ b!$	5	174.3
$x?\ a!\ b!\ a!\ x?\ a!\ b!\ a!$	1	100.0	$y?\ a!\ a!\ b!\ y?\ a!\ a!\ b!$	2	100.0
			$y?\ a!\ a!\ \delta$	2	55.3
			$y?\ a!\ a!\ b!\ y?\ c!\ w!\ b!$	3	174.3
			$y?\ c!\ w!\ b!\ y?\ c!\ w!\ b!$	2	248.5

Table 9.3: Simulating 20 executions of t_1 and t_2 using $\mathcal{I}_1$

Executions of t_1	#	*absCov*	Executions of t_2	#	*absCov*
x? e! w! a! x? e! w! a!	8	248.5	*y? c! w! b! y? c! w! δ*	4	253.8
x? e! w! a! x? a! b! a!	5	174.3	*y? c! w! b! y? a! a! b!*	2	174.3
x? e! w! a! x? e! a!	4	294.3	*y? a! a! b! y? a! a! b!*	3	100.0
x? a! b! a! x? e! w! a!	3	174.3	*y? a! a! b! y? c! w! b!*	6	174.3
			y? c! w! b! y? a! a! δ	1	179.5
			y? c! w! b! y? c! w! b!	2	248.5
			y? a! a! b! y? a! a! δ	1	105.3
			y? a! a! b! y? c! w! δ	1	179.5

Table 9.4: Simulating 20 executions of t_1 and t_2 using $\mathcal{I}_2$

Although this is indeed approximately the case in $\mathcal{I}_2$, during the first part of our test cases no faults are present. Therefore, we obtain executions that are longer than expected, showing the absence of more faults and therefore obtaining more actual coverage.

We can also directly observe that the executions that are present in $\mathcal{I}_1$ but not in $\mathcal{I}_2$ (*x? e! a!*, *y? c! w! δ*, and *y? a! a! δ*), are the traces with the lowest absolute actual coverage. This immediately explains that testing $\mathcal{I}_2$ yields a higher actual coverage than testing $\mathcal{I}_1$.

Although the simulation of 20 executions gives insight in the procedure of test executions, it is actually not quite representative to make a comparison between the predicted and simulated actual coverage values. After all, some executions might coincidentally occur often in these small samples, while in the long run they would only occur sporadically.

Moreover, it is interesting to know how much actual coverage *sequences* of executions yield when executing the test cases or test suite several times.

Therefore, we performed several simulations for all combinations of t_1, t_2, and T with $\mathcal{I}_1$ and $\mathcal{I}_2$. For each combination we simulated 10.000 single executions, 10.000 sequences of 5 executions, 10.000 sequences of 10 executions, 10.000 sequences of 50 executions, and 10.000 sequences of 250 executions. We calculated the actual coverage of each of these sequences, and took the averages. Moreover, we calculated the standard deviation for each of the samples, indicating the width of the distribution.

Table 9.5 shows the expected values obtained earlier and the results of the simulations for $\mathcal{I}_1$. Table 9.6 shows this information for $\mathcal{I}_2$. For layout purposes, the subscript f has been omitted from $\mathbb{E}(A^n_{t_i}, f)$. The notation $\overline{\text{Sim.}}$ is used to denote the average of the simulation results.

	t_1			t_2			T		
n	$\mathbb{E}(A^n_{t_1})$	$\overline{\text{Sim.}}$	std.	$\mathbb{E}(A^n_{t_2})$	$\overline{\text{Sim.}}$	std.	$\mathbb{E}(A^n_T)$	$\overline{\text{Sim.}}$	std.
1	197.0	213.3	50.1	156.8	155.1	60.8	520.2	543.0	88.0
5	729.1	762.1	84.0	639.8	629.7	107.0	1716.7	1742.5	150.0
10	1076.9	1112.6	104.8	1032.1	1013.3	114.8	2423.8	2443.7	161.4
50	1704.6	1743.3	62.4	1848.0	1831.2	39.5	3695.3	3697.9	50.0
250	1917.7	1925.8	11.2	1938.0	1938.0	0.0	3873.3	3875.0	1.4

Table 9.5: Simulation results for $\mathcal{I}_1$

	t_1			t_2			T		
n	$\mathbb{E}(A^n_{t_1})$	$\overline{\text{Sim.}}$	std.	$\mathbb{E}(A^n_{t_2})$	$\overline{\text{Sim.}}$	std.	$\mathbb{E}(A^n_T)$	$\overline{\text{Sim.}}$	std.
1	197.0	229.1	48.4	156.8	174.7	52.0	520.2	591.5	73.5
5	729.1	813.9	56.5	639.8	711.6	89.4	1716.7	1878.9	130.6
10	1076.9	1167.9	94.8	1032.1	1133.6	92.9	2423.8	2603.9	146.0
50	1704.6	1757.3	62.9	1848.0	1890.7	19.4	3695.3	3762.0	35.4
250	1917.7	1926.0	11.1	1938.0	1938.0	0.0	3873.3	3875.0	1.5

Table 9.6: Simulation results for $\mathcal{I}_2$

Interpretation
As explained earlier, the simulations of $\mathcal{I}_1$ yield a slightly higher actual coverage, since one of the faults that were chosen to be present is coincidentally of higher error weight than the average fault.

Furthermore, the simulations of $\mathcal{I}_2$ result in an even higher actual coverage, because of the absence of faults during the first request. This results in longer executions, showing the absence of more faults and that way achieving more actual coverage.

Nevertheless, the expected actual coverage values are all quite close to their corresponding simulation results. For the combination of $\mathcal{I}_1$ and t_2, which most accurately corresponds to the estimated probabilistic behaviour of the system, the errors in our predictions are even less than two percent of the simulation results.

As stated before, this example is solely meant as a demonstration of the framework. An extensive future study applying actual coverage to real software projects would be necessary to fully show the usability, sensitivity and accuracy of our methods. However, the current results provide good hope for the applicability of our measures.

Chapter 10

Conclusions and Future Work

In this thesis, we developed a framework for actual coverage. This chapter first summarises the main results, listing the most important concepts that were developed. Then, we evaluate our framework by discussing the main advantages and disadvantages. Also, we recall the requirements for actual coverage discussed in Section 4.2, explaining how our methods meet these requirements. Finally, directions for future work are proposed.

Organisation of this chapter
First, the summary is given in Section 10.1. Then, the evaluation is discussed in Section 10.2. Finally, future work is covered in Section 10.3.

10.1 Summary of the results

- We first extended the existing framework for potential coverage from [BBS06], explaining in detail how to deal with *nondeterministic systems*. The notion of *quiescence* was updated to support its preservation under determinisation.

- We developed a new notion of coverage: *actual coverage*. It deals not only with test cases or test suites, but also with the *number of executions* planned and the *probabilistic behaviour* of the system. Using this notion, we defined the *actual coverage distribution* of test cases.

- Actual coverage resembles the notion of potential coverage of [BBS06] in the sense that it accumulates the error weights of faults in a test case or test suite. However, we take only a fraction of each error weight, equal to the *coverage probability* of the corresponding fault. This coverage probability indicates how certain we know whether or not the fault is present. For an infinite amount of executions, our notion coincides with potential coverage.

- We introduced the *probabilistic execution model* (PEM), containing the probabilities necessary to evaluate the actual coverage of a given execution and to predict the actual coverage a test case or test suite will yield. We introduced *probabilistic fault automata* (PFAs) to syntactically specify

PEMs. Moreover, we provided an *efficient method* to predict the actual coverage of both test cases and test suites.

10.2 Evaluation

We obtained a framework that supports describing the expected behaviour of a system. Based on this information it calculates the actual coverage of a given execution or sequence of executions, and predict the actual coverage a test case or test suite will yield. The framework is therefore useful in test evaluation, but also in test selection. Since the most important properties can be calculated in polynomial time, it seems feasible to implement the theory in a tool.

Although an extensive case study would be necessary to fully show the usability, sensitivity and accuracy of our methods, our detailed example demonstrated that our methods indeed can be applied to small examples. Moreover, another research project our group is currently working on showed that the notion of potential coverage indeed gives an indication for how well a test suite will be able to detect erroneous behaviour [Men08]. Since our framework is based on the same philosophies, this is quite promising.

The necessity of many estimations might limit the usability of our framework. However, the approximations we proposed reduce this limitation substantially. Still, these approximations have to be thoroughly verified by future research. Another approach would be to estimate probabilities *during* the test process, as indicated in Section 10.3.

We list the requirements of actual coverage discussed in Section 4.2 once more, and discuss for each requirements to what extend it has been met.

1. When the number of executions of a test case approaches infinity, its actual coverage should approach its potential coverage.

   Fulfillment:

   This requirement has been fulfilled completely, as proved formally in Theorem 7.5 and Theorem 8.13. The only assumption for the prove to remain valid is that all errors defined in a test case are actually reachable. It is obvious that this assumption was necessary, since unreachable errors *are* accounted for in the potential coverage measure, while of course one would *not* want the actual coverage measure to ever include them.

2. The actual coverage of a sequence of test case or test suite executions E should be larger than or equal to the actual coverage of a sequence of executions $E' \subseteq E$.

   Fulfillment:

   Looking at the definition of coverage probabilities and actual coverage in Section 6.2 and Section 6.3, observe that the coverage of any number of executions is defined as a sum over all erroneous traces in the test case, multiplying the error weight of each trace by its coverage probability. From the definition of coverage probabilities it is immediate that increasing the number of executions cannot decrease the probability, therefore remaining at the same or obtaining an increased actual coverage.

3. Correct executions might have a nonzero actual coverage value.

   Fulfillment:

   Since coverage probabilities have been defined to be positive if a state has been reached from where an error *could* occur, even correct executions can have a nonzero value for actual coverage. Example 6.12 provided the calculation of the actual coverage of a correct execution, showing it is nonzero.

4. Observing the same correct execution more often should increase actual coverage, with an amount depending on the probability with which potential failures on its path occur. It should depend on the error weights of the corresponding faults as well.

   Fulfillment:

   The definition of coverage probabilities in Section 6.2 shows that it depends on both the conditional branching probability of a fault *and* the number of executions covering it. The more correct executions covering it, the higher the coverage probability. After observing an erroneous execution the coverage fraction cannot increase anymore, since it is already 1.

5. After an execution terminated by failure, later executions observing either the presence or absence of the corresponding fault should not influence its actual coverage anymore.

   Fulfillment:

   Immediately from the definition of coverage probabilities.

10.3 Future work

Our work motivates several interesting directions for future research.

First, it is crucial to *validate* our framework by means of a series of *case studies*. Obviously, tool support is an essential prerequisite. The case studies should at least investigate how well software engineers can estimate the probabilities necessary for our calculations. They might reveal the accuracy of the estimations, but also the effect of inaccuracies on the results (sensitivity).

Second, it could be useful to take into account the *dependencies between erroneous traces*. As indicated in Section 5.7, at the moment we still assume that observing a fault from some state s after a trace σ does not necessarily mean that the same fault is also present from s after a trace $\sigma' \neq \sigma$. In practice it might often be the case that a fault is present after any trace ending in some state, or maybe after any trace with at least a certain length.

Third, more work should be done to investigate the *accuracy of the approximations* we proposed at the end of Section 5.8.3 and in Section 7.4. Although they appeared to be accurate for small examples, this has yet to be proved for industrial applications.

Fourth, it would be very interesting to investigate the possibilities of *on-the-fly test derivations* using actual coverage. This could result in a tool, calculating probabilities while testing and deciding which branches to take during the process. This might lift a large burden, since several estimations might be left to the tool.

Appendix A

The subset construction algorithm

Algorithm A.1 gives the subset construction in pseudo-code, with a few extra lines of functionality for incorporating the error weight function.

The algorithm starts by an initialization, during which the alphabet and the initial state of the deterministic LTS $\mathcal{A}$ are set equal to the alphabet and the initial state of the non-deterministic LTS $\mathcal{N}$. Furthermore, that initial state is added to the set of states of $\mathcal{A}$ and it is added to an auxiliary variable F containing all states of $\mathcal{A}$ for which not all transitions have been included yet. Notice that we do not just add s^0, we add $\{s^0\}$. The reason for this is that we identify states of $\mathcal{A}$ as sets of states of $\mathcal{N}$. If for example $\mathcal{N}$ can be in either state s_1, s_2 or s_5, we say that $\mathcal{A}$ is in state $\{s_1, s_2, s_5\}$.

After the initialization, we take a state S' from the auxiliary variable F and add the appropriate transitions from it to other states. This involves iterating over all possible actions in the alphabet, and calculating for each action the set of states of $\mathcal{N}$ (called Y) that can be reached from one of the states contained in S'. Suppose Y is the empty set for some action a, then apparently no a transition can occur from any of the states in S'. Therefore, we do not add anything to $\mathcal{A}$ and we just continue. If Y is not empty and it is not yet contained in S_2, we add it to S_2 and we also add it to F. We then add a transition labeled a from S' to Y, indicating that if the system is in one of the states of S', it can after that go to one of the states of Y by an a transition. After this has been done for all actions, we are done with S' and it is accordingly removed from F. This process continues, until all states have been processed (i.e., F is empty).

Thus far, we described the regular subset construction algorithm. However, we updated the classical version such that it defines the error weight function for the resulting deterministic FA. It is not hard to see how this works. When a transition is added from a state S' to a state Y by an action a, we set $r_2(S', a) = 0$. After all, this transition is apparently possible and therefore correct. However, in case we find that for some output action there are no states reachable from any of the states contained in S', we know that this output action is erroneous in S'. Therefore, we give $r_2(S', a)$ a positive value, obtained by taking the minimum over all error weights defined for outputting a in the states contained in S'.

Algorithm A.1: The subset algorithm incorporating error weights

Input: a non-deterministic LTS $\mathcal{N} = \langle S, s^0, L^O \cup L^I, \Delta \rangle$ and a function $r : S \times L^O \to \mathbb{R}^{\geq 0}$

Output: the equivalent deterministic LTS $\mathcal{A} = \langle S_2, s_2^0, L_2^O \cup L_2^I, \Delta_2 \rangle$ and its function $r_2 : S_2 \times L_2^O \to \mathbb{R}^{\geq 0}$

Initialization
$S_2 \leftarrow \{\{s^0\}\}$
$F \leftarrow \{\{s^0\}\}$
$s_2^0 \leftarrow s^0$
$L_2 \leftarrow L$

repeat
  Let $S' \in F$
  forall $a \in L \cup \{\delta\}$ **do**
    $Y \leftarrow \bigcup_{s_i \in S'} \{s' \in S \mid (s_i, a, s') \in \Delta\}$
    if $a \in L^O \wedge Y = \varnothing$ **then**
      let $r_2(S', a) = \min_{s_i \in S'} r(s_i, a)$
    else if $Y \neq \varnothing$ **then**
      if $Y \notin S_2$ **then**
        add Y to S_2
        add Y to F
      end
      add the transition (S', a, Y) to Δ_2
      let $r_2(S', a) = 0$
    end
  end
  remove S' from F
until $F = \varnothing$

A.1 An example

Now that the algorithm is given, we clarify the procedure by working through a detailed example, based on the fault automaton of Figure A.1. We will refer to that FA by $\mathcal{F} = \langle \langle S, s^0, L, \Delta \rangle, r \rangle$, and to the resulting deterministic FA by $\mathcal{F}_2 = \langle \langle S_2, s_2^0, L_2, \Delta_2 \rangle, r_2 \rangle$.

Example A.1. As explained, we start with an initialization. We add the state $\{s_0\}$ to S_2, and give it the correct alphabet and initial state. Furthermore, F is initialized to $\{\{s_0\}\}$. The intermediate $\mathcal{F}_2$ is shown in Figure A.2(a) on page 112.

Then, we choose an element of F for the first iteration. Since F only contains $\{s_0\}$, this state is chosen. We check for each action in the alphabet what can happen in state s_0 of $\mathcal{F}$. Since an *a!* transition to either s_0, s_1 or s_2 can take place, we add the state $\{s_0, s_1, s_2\}$ to S_2, and we add a transition labeled *a!* from $\{s_0\}$ to $\{s_0, s_1, s_2\}$. No *c!* transition can take place from s_0, but since this is an output action nothing has to be added to $\mathcal{F}_2$ to denote that. Also no *b!* transition can take place, but since this is an output action now we *should*

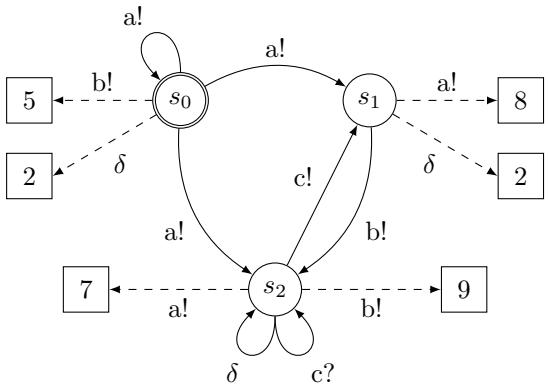


Figure A.1: A nondeterministic fault automaton

add an error weight. There is only one applicable error weight defined in $\mathcal{F}$, namely 5, so this is added to $\mathcal{F}_2$. Likewise, an error weight of 2 is assigned to the quiescence action δ. The intermediate $\mathcal{F}_2$ is shown in Figure A.2(b).

Now, we will add the transitions for the $\{s_0, s_1, s_2\}$ state. No *a!* transition can occur from s_1 or s_2, but from s_0 there *are* *a!* transitions to s_0, s_1 and s_2. Therefore, we add a transition from $\{s_0, s_1, s_2\}$ labeled *a!* to itself. A *b!* transition can take place from s_1 to s_2, but from no state in the set $\{s_0, s_1, s_2\}$ we can go by a *b!* transition to any other state. Therefore, we add the state $\{s_2\}$ to S_2 and add a transition from $\{s_0, s_1, s_2\}$ to it, labeled *b!*. Likewise, a transition from $\{s_0, s_1, s_2\}$ to $\{s_2\}$ labeled δ is added. Looking at *c?* transitions, we observe that it is possible to go to state s_2 or to s_1 in case we were in s_2, so we add the state $\{s_1, s_2\}$ to S_2 and we add a transition from $\{s_0, s_1, s_2\}$ to it, labeled *c?*. The intermediate $\mathcal{F}_2$ is shown in Figure A.2(c).

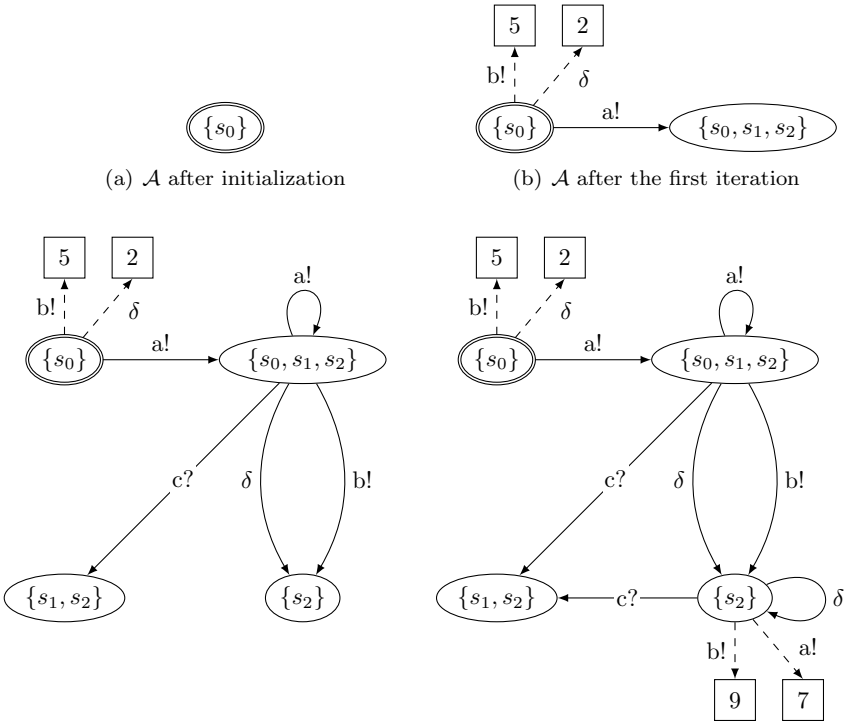
During the previous iteration, both $\{s_2\}$ and $\{s_1, s_2\}$ were added to S_2, and therefore to F. Hence, for the next iteration we can choose which state to process. This choice does not influence the output, only the intermediate fault automata [Sud97]. Let's say we choose $\{s_2\}$. From there, a δ brings us back to the same state. No *a!* should occur, and this error is given the weight 7 ($r(s_2, a!)$ in the original fault automaton). Similarly, an error weight of 9 is added for *b!*. By a *c?* transition we can go to either s_1 or s_2, so a transition from $\{s_2\}$ to $\{s_1, s_2\}$ labeled *c?* is added. The intermediate $\mathcal{F}_2$ is shown in Figure A.2(d).

Finally, in the last iteration $\{s_1, s_2\}$ is processed. It is interesting to see what happens to the *a!* transition here. In neither s_1 nor s_2 an *a!* transition can occur, so it is erroneous. We have $r(s_1, a!) = 8$ and $r(s_2, a!) = 7$, so since the algorithm states that we take the minimum, we have $r_2(\{s_1, s_2\}, a!) = 7$.

The final deterministic automaton $\mathcal{F}_2$ is shown in Figure A.2(e).

A.2 Complexity

If a non-deterministic automaton has a state set S with n elements, then all combinations of these states can appear as a state in the deterministic equivalent automaton. Since these combinations are elements of the powerset of S, the algorithm is sometimes refered to as the *powerset construction*. From basic

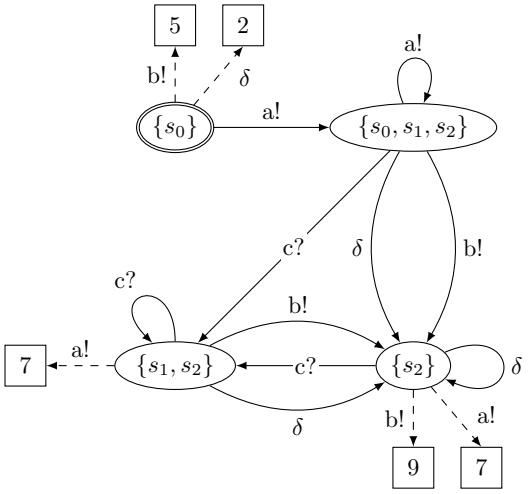


(a) $\mathcal{A}$ after initialization

(b) $\mathcal{A}$ after the first iteration

(c) $\mathcal{A}$ after the second iteration

(d) $\mathcal{A}$ after the third iteration



(e) $\mathcal{A}$ after the fourth iteration

Figure A.2: Removing non-determinism

combinatorics it follows that the deterministic automaton can have at most 2^n states.

Looking at the algorithm, it is not difficult to see that the time in which it operates is therefore in worst-case exponentially related to the number of states of its input automaton. The forall loop is even executed at most $2^n m$ times, where m is the number of actions in the alphabet.

The space complexity can be quite bad as well. The number of transitions in the resulting deterministic automaton is at most $2^{2n}m$; an exponential growth as compared to the maximum of $n^2 m$ transitions in the non-deterministic input automaton.

However, despite these disappointing worst-case scenarios, the subset construction method is still used a lot. According to [Kla98], it behaves usually very well in practice, often resulting in a deterministic automaton with even less states than the original non-deterministic automaton.

For more information on efficient implementations of and optimization for the subset construction, we refer to [Les95].

Appendix B

Lemmas for Theorem 7.7

Lemma B.1. *Let p and q be probabilities, $p > 0$, $q < 1$ and $n \in \mathbb{N}$. Then*

$$\sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} \cdot (1-q^i) \leq \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} = 1$$

Proof. The term $\binom{n}{i} p^i (1-p)^{n-i}$ corresponds to the probability of having i successes in n Bernoulli experiments with parameter p. It follows directly that the sum of this term over all i from 0 to n is equal to 1, as stated by the equality, since it sums over the probabilities of all possible number of successes.

Furthermore, from the requirement $0 \leq q < 1$ we obtain $\forall i \geq 0 : 0 \leq q^i \leq 1$, and therefore $\forall i \geq 0 : 0 \leq (1-q^i) \leq 1$, proving the inequality. □

Lemma B.2. *Let p, q and r be probabilities, $p > 0$, $q, r < 1$ and $n \in \mathbb{N}$. Then*

$$\lim_{n \to \infty} \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} \cdot (1-q^i) \cdot r^i = 0$$

Proof. By the definition of limits it needs to be shown that for all $\epsilon > 0$ there exists a $\delta > 0$, such that for all $n > \delta$

$$\left| \sum_{i=0}^{n} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) \cdot r^i \right| < \epsilon \tag{B.1}$$

Without loss of generality, assume $0 < \epsilon < 1$. Now find a value η such that

$$(1-q^\eta) r^\eta < \frac{\epsilon}{2}$$

Such an η exists, because by the assumptions on q and r we know that $\lim_{\eta \to \infty} (1-q^\eta) r^\eta = \lim_{\eta \to \infty} (1-q^\eta) \cdot \lim_{\eta \to \infty} r^\eta = 1 \cdot 0 = 0$.

It follows directly that

$$\forall i \geq \eta : (1-q^i) r^i < \frac{\epsilon}{2} \tag{B.2}$$

Now find a value δ such that

$$\sum_{i=0}^{\eta} \binom{\delta}{i} p^i \cdot (1-p)^{\delta-i} \cdot (1-q^i) \cdot r^i < \frac{\epsilon}{2}$$

Such a δ exists, because $(1-q^i)r^i$ is obviously always smaller than 1 and $\sum_{i=0}^{\eta} \binom{\delta}{i} p^i \cdot (1-p)^{\delta-i}$ denotes the probability of $0 \dots \eta$ successes when performing δ Bernoulli experiments with parameter p. It is trivial that the probability of $0 \dots \eta$ successes approaches 0 when δ approaches ∞ (using the assumption that $p > 0$), so a δ for which it is smaller than $\frac{\epsilon}{2}$ exists.

Then, for $n > \delta$

$$\begin{aligned}\sum_{i=0}^{n} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) r^i &= \sum_{i=0}^{\eta} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) r^i \\ &\quad + \sum_{i=\eta+1}^{n} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) r^i \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.\end{aligned}$$

The fact that $\sum_{i=0}^{\eta} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) r^i < \frac{\epsilon}{2}$ follows from the choice of δ, while $\sum_{i=\eta+1}^{n} \binom{n}{i} p^i \cdot (1-p)^{n-i} \cdot (1-q^i) r^i < \frac{\epsilon}{2}$ follows from Equation B.2 and Lemma B.1.

It can easily be seen that no summand can be negative, so this proves the correctness of Equation B.1. Because δ was derived for an arbitrary ϵ, this proves Lemma B.2. □

References

[ADD99] R.B. Ash and C.A. Doleans-Dade. *Probability & Measure Theory, Second Edition.* Academic Press, December 1999.

[AL06] K.B. Athreya and S.N. Lahiri. *Measure Theory and Probability Theory (Springer Texts in Statistics).* Springer-Verlag, Secaucus, NJ, USA, 2006.

[Aml00] S. Amland. Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software, Volume 53, Issue 3*, pages 287–295, 2000.

[Bal05] T. Ball. A Theory of Predicate-Complete Test Coverage and Generation. In *Formal Methods for Components and Objects*, pages 1 – 22, 2005.

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[BBS06] E. Brinksma, L. Brandán Briones, and M.I.A. Stoelinga. A semantic framework for test coverage. In *Proceedings of the fourth international symposium on Automated Technology for Verification and Analysis (ATVA'06)*, LNCS. Springer-Verlag, 2006.

[BFS05] A. Belinfante, L. Frantzen, and C. Schallhart. Tools for test case generation. In *Model-Based Testing of Reactive Systems*, pages 391 – 438, 2005.

[Bri07] L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage.* PhD thesis, University of Twente, Enschede, March 2007.

[CGN+05] C. Campbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. Technical report, Microsoft Research, Redmond, May 2005.

[Dij70] E.W. Dijkstra. Structured programming. In *Software Engineering Techniques*. NATO Science Committee, August 1970.

[Gog03] N. Goga. A probabilistic coverage for on-the-fly test generation algorithms. In *Proceedings of Automated Verification of Critical Systems (AVoCS '03), Southampton, UK*, 2003.

[Hal50] P.R. Halmos. *Measure Theory*. Van Nostrand Company Inc, 1950.

[HT96] L. Heerink and J. Tretmans. Formal Methods in Conformance Testing: A Probabilistic Refinement. In *Proceedings of the 9th International Workshop on Testing of Communicating Systems*, pages 261–276. Chapman & Hall, 1996.

[Kla98] N. Klarlund. Mona & fido: The logic-automaton connection in practice. In *CSL '97: Selected Papers from the 11th International Workshop on Computer Science Logic*, pages 311–326, London, UK, 1998. Springer-Verlag.

[Les95] T. Leslie. Efficient approaches to subset construction. Master's thesis, Computer Science, University of Waterloo, 1995.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[McC76] T.J. McCabe. A complexity measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.

[Men08] J. Mengerink. SeCo: a tool for semantic test coverage. Master's thesis, Computer Science, University of Twente, forthcoming in 2008.

[MSBT04] G.J. Myers, C. Sandler, T. Badgett, and T.M. Thomas. *The Art of Software Testing, Second Edition*. Wiley, June 2004.

[Mye79] G.J. Myers. *The Art of Software Testing*. Wiley, New York, NY, USA, 1979.

[New02] M. Newman. Software errors cost u.s. economy 59.5 billion annually, NIST assesses technical needs of industry to improve software-testing. *Press Release, http://www.nist.gov/public_affairs/releases/n02-10.htm*, 2002.

[NVS+04] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. *SIGSOFT Software Engineering Notes*, 29(4):55–64, 2004.

[PM92] R.L. Probert and O. Monkewich. TTCN: The International Notation for Specifying Tests of Communications Systems. *Computer Networks and ISDN Systems*, 23(5):417–438, 1992.

[RRS05] G. Rodrigues, D. Rosenblum, and S.Uchitel. Using Scenarios to Predict Reliability for Concurrent Component-Based Systems. In *Fundamental Approaches to Software Engineering (FASE) at the European Joint Conferences on Theory and Practice of Software (ETAPS)*, 2005.

[Seg95] R. Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, jun 1995. Available as Technical Report MIT/LCS/TR-676.

[SLK01] T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Commununications of the ACM*, 44(6):103–108, 2001.

[Sud97] T.A. Sudkamp. *Languages and machines: an introduction to the theory of computer science*. Addison-Wesley, Boston, MA, USA, 1997.

[TPB95] Q. Tan, A. Petrenko, and G. Bochmann. Testing trace equivalence for labeled transition systems. Technical report, University of Montreal, 1995.

[Tre96] G.J. Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software—Concepts and Tools*, 3:103–120, 1996.

[Ura92] H. Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311–325, 1992.

[WS00] T.W. Williams and S. Sunter. How Should Fault Coverage Be Defined? In *Proceedings of the 18th IEEE VLSI Test Symposium (VTS'00)*, page 325, Washington, DC, USA, 2000. IEEE Computer Society.

[Wun85] H.-J. Wunderlich. PROTEST: A Tool for Probabilistic Testability Analysis. In *DAC '85: Proceedings of the 22nd ACM/IEEE conference on Design automation*, pages 204–211, New York, NY, USA, 1985.

[Yan91] M. Yannakakis. Testing Finite State Machines. In *Proceedings of the twenty-third annual ACM Symposium on Theory of Computing*, pages 476–485, New York, NY, USA, 1991.

[ZE00] L. Zhao and S. Elbaum. A survey on quality related activities in open source. *SIGSOFT Software Engineering Notes*, 25(3):54–57, 2000.

List of Notations, Functions and Abbreviations

Operators

$\sigma \sqsubseteq_{\forall}$ X	For all members e_i of the tuple X we have $\sigma \sqsubseteq e_i$, page 38
$\sigma \sqsubseteq_{\exists}$ X	There exists a member e_i of the tuple X such that $\sigma \sqsubseteq e_i$, page 38
$\sigma \sqsubseteq_{\nexists}$ X	There does not exist a member e_i of the tuple X such that $\sigma \sqsubseteq e_i$, page 38
$\sqsubset$	Proper prefix, page 10
$\sqsubseteq$	Prefix, page 10
$\mid \sigma \mid$	The length of a trace σ, page 9

Symbols

δ	The quiescence action, page 12
$\mathbb{E}$	The expected value of a random variable, page 67
$f_{\mathcal{F}}^{\alpha}$	The discounted WFM of an FA, page 19
$f_{\mathcal{F}}^{k}$	The finite depth WFM of an FA, page 18
$f_{p_t^{\text{to}}}$	A risk-based weighted fault model, page 52
$\mathcal{N}$	A nondeterministic LTS, page 22
Ω	A sample space, page 36
$\mathbb{P}$	A probability distribution function for a random variable, page 36
P	A probability distribution function for a probability space, page 36

S_X	The codomain of the random variable X, page 36

Random variables

$A^n_{t,f}$	The random variable denoting the absolute actual coverage of X^n_t given the WFM f, page 66
$A_{t,f}$	The random variable denoting the absolute actual coverage of X_t given the WFM f, page 65
$R^n_{t,f}$	The random variable denoting the relative actual coverage of X^n_t given the WFM f, page 66
$R_{t,f}$	The random variable denoting the relative actual coverage of X_t given the WFM f, page 65
X^n_t	The random vector denoting the sequence of executions that n trials of X_t yield, page 38
X_t	The random variable denoting the trace obtained when executing the test case t once, page 37
$SoP_t(\sigma)$	The random variable denoting the state of presence of the trace σ, page 42

Sets

$Distr(S)$	The set of all distribution functions over S, page 10
err_t	The set of all erroneous executions of t, page 13
$exec_t$	The set of all executions of t (maximal traces), page 13
$inner_t$	The set of all non-maximal traces it t, page 13
$traces_{\mathcal{A}}$	The set of all traces of the LTS $\mathcal{A}$, page 12
L^*	The set of all traces over L, page 9

Functions

absCov	The absolute actual coverage function, page 59
absPotCov	The absolute potential coverage function, page 15
faultCov	The fault coverage function, page 59
$final_{\mathcal{A}}$	The final state function, page 12
observedSoP	The observed state of presence function, page 61
p^{br}	The branching probability function, page 39
p^{cbr}	The conditional branching probability function, page 42
p^{fbr}	The flawless branching probability function, page 45

p^{cov}	The coverage probability function, page 56
p^{pr}	The presence probability function, page 46
p^{scbr}	The specialised conditional branching probability function, page 47
p^{to}	The trace occurrence function, page 39
r	The error weight function, page 17
$\bar{r}$	The accumulated error weight function, page 17
relCov	The relative actual coverage function, page 59
relPotCov	The relative potential coverage function, page 15
totCov	The total coverage function, page 15
v	The verdict function, page 14

Abbreviations

FA	Fault automaton, page 17
LTS	Input-output labeled transition system, page 10
PFA	Probabilistic fault automaton, page 44
WFM	Weighted fault model, page 15