Theories for Model-based Testing: Real-time and Coverage

Laura Brandán Briones

Graduation committee:

Prof. Dr. H. Brinksma (promotor)	University of Twente/Embedded Systems Institute, The Netherlands
Prof. Dr. J. W. Fokkink	Vrije Universiteit Amsterdam, The Netherlands
Prof. Dr. P. H. Hartel	University of Twente, The Netherlands
Prof. Dr. K. G. Larsen	Aalborg University, Denmark
Dr. Ir. A. Rensink	University of Twente, The Netherlands
Dr. M. I. A. Stoelinga	University of Twente, The Netherlands
Dr. Ir. J. Tretmans	Radboud University Nijmegen, The Netherlands
Dr. W. Visser	NASA Ames, United States

The work in this thesis has been carried out under the auspices of the Institute for Programming Research and Algorithmics (IPA) research school and within the context of the Center for Telematics and Information Technology (CTIT).

IPA Dissertation Series No. 2007–05
Series title: CTIT Ph.D.-thesis Series
Series number: 1381-3617
CTIT number: 07-97

Typeset by LaTeX, edited with Emacs, and printed by Gildeprint b.v.
Cover: The Bugis designed by Laura Brandán Briones.

ISBN: 978-90-365-2476-6

THEORIES FOR MODEL-BASED TESTING: REAL-TIME AND COVERAGE

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, March 21, 2007 at 15:00

by

Laura Brandán Briones

born on 23 September 1975

in Mendoza, Argentina

This dissertation is approved by:

Prof. Dr. H. Brinksma (promotor)

A mis padres, Elizabeth y Eugenio.

Acknowledgments

I thank Ed, not only for being a great advisor, director and friend, but mainly because he believed in me. He gave me this great opportunity, he allowed me to try to do a PhD and be here. I know that I am not an easy person and I am sure that only a great being could see beyond to give me this kind of opportunity. As one day I said: “I will remember you”.

In the technical aspects, firstly, I thank our nice secretary Joke, who did a lot of useful things for me that are not supposed to be done by a secretary. I am very grateful to her for all her help. Secondly, I thank some friends that helped me to solve problems and understand better things in research: Ed, Pedro, Axel, Conrado, Mariëlle, Jan, and Riky. Thirdly, I thank everyone who helped me to have a more understandable thesis: Riky, Mariëlle, Henrik, Conrado, Wan, Pieter, and Ed. I also warmly thank the members of my thesis committee: Ed Brinksma, Wan Fokkink, Pieter Hartel, Kim Larsen, Arend Rensink, Mariëlle Stoelinga, Jan Tretmans, and Willem Visser.

I thank Axel who translated the Summary into Dutch, and also Andre and Theo for their useful comments. Moreover, I thank Isabel and Riky for their appropriate corrections in the translation of the Summary into Spanish.

I thank the FMT group for allowing me to do several trips in these years: summer schools, conferences, research visits, where I made new friends and enjoyed beautiful places. Also, I thank Willem and Corina that made my stay in NASA so pleasant.

My years in Holland were not difficult, the truth is that I enjoyed them very much, although I did not learn Dutch I felt very comfortable. Many people made my days here so nice, like my BOCOM friends (Arend, Joost, Georgios, etc.), the 4^{th} floor friends (Vasu, Nikolay, Jordan, Supriyo, etc.), and some old friends that left (Pedro, Joost-Pieter, Holger, Tomas, Ruj, Yaroslav, etc.). Last but not least I thank my closest friends: Gabriele, Conrado, Isabel, Marcos, Lorena, Axel, and Mariëlle.

I thank some friends that even though I was far away these years they were always with me: Mariana, Ileana, Natalia, Tamara, Victor, Eli, Andrea, Flavio, Ester, Hubert, and Valeria.

Muchísimas gracias a mi familia, a mi madre Elizabeth por nuestras largas charlas por teléfono que me hicieron sentirlos cerca, a mi padre Eugenio por sus sabios consejos que me sirvieron y servirán durante toda mi vida, a mi hermana Virginia por enseñarme la importancia de ser feliz, a mi hermano Eugenio y su familia por darnos la alegría de sus hijos. Y por supuesto a todos mis demás familiares, tíos, primos, sobrinos, cuñados, suegros, etc.

Finally, I thank my friend, my partner, my lover.

My confidant, my dream, my happiness, my sadness.

My quiet morning, my warm end of the day.

My song, my silence, my peace, my desire.

I thank my love, Ricardo.

Enschede, February 2007.

Table of contents

CHAPTER 1

Introduction

1.1 Software testing

Nowadays, software is ubiquitous; it is used not only to drive computers, but also home devices, cars, airplanes, and even flood control barriers (e.g., the Beslis en Ondersteunend Systeem [36] which controls the barrier built in the Nieuwe Waterweg, a canal connecting the harbour of Rotterdam to the North Sea in The Netherlands).

At a rapidly increasing rate, more and more sophisticated tasks are being left to be handled by computerized systems. This requires, in turn, accordingly complex software. Given the significance of these tasks, it is of utmost importance to attain a high degree of the software correctness, quality, and reliability.

In order to prevent as much as possible faulty system behaviours which may cause severe damage, there is a need to check whether systems behave as expected. Usually, the importance of correctness of such a system is best measured by the consequences of their failure (e.g., how much money an error could cost, or how many human lives are endangered by an error).

One particularly successful method for achieving these properties is based on systematically testing the software. Testing is the process of trying to find errors in a system, by means of experimenting with it. Thus, software testing is an operational way to check the correct behaviour of a system implementation, by carefully interacting and experimenting with it. This is achieved by applying a series of tests on the implementation, and studying its reactions in a controlled environment.

Testing is in essence an instrument for measuring quality; it increases the value of a product by establishing confidence in its quality, and it helps in assessing the risk of putting a product into operation.

Despite its importance, the thorough testing of software is under pressure when products have to be delivered on time. There are practical reasons for this: testing is both expensive and difficult. Primitive, “hands-on” testing methods lack a solid foundation, and their testing strategies are driven by heuristics that may not be always successful. Moreover, software testing has not been studied as much as one would expect. On the other hand, it has been mainly applied in the industry so far; there, testing is one of the most important techniques for software validation (whenever there is time and money to do any kind of software validation). In contrast, in academia, software testing has only become a topic of serious research in the past decade or so. Fortunately, after some years of limited attention, the theory of testing

has now become a widely studied, academically respectable subject of research. This is evidenced by the increasing number of papers related to testing being presented on international workshops and conferences (e.g. FATES, TESTCOM).

In order to test a system, the desired behaviour of the system must be known in advance. A description of the desired behaviour describes what a system must do, not how this is done. A system that is supposed to implement the desired behaviour is called an implementation; possibly, an implementation can be a real-life object, consisting of a combination of both hardware and software components.

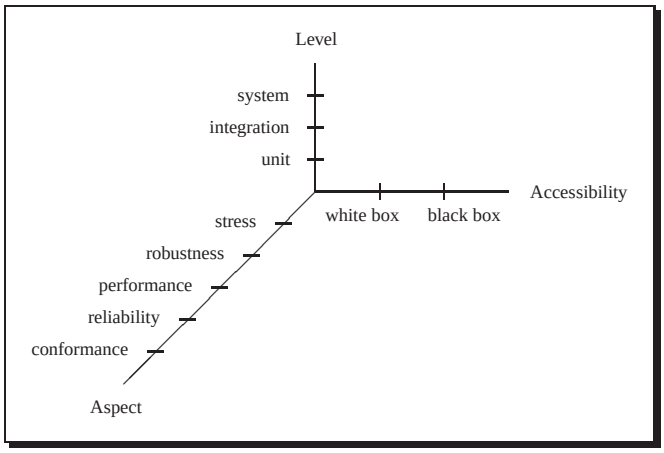


Figure 1.1: Types of testing

Figure 1.1 shows different types of testing. As it occurs usually in other methods for analyzing software, one can test a system at different levels of abstraction. If we wish to test a system at its most fine-grained level, thus testing the smallest "testable" pieces, then we are performing the so-called unit testing. If, on the other hand, we are interested in testing the cooperation of a number of units whose composition forms a system component, then we are performing integration testing. Finally, we perform system testing when we aim at testing the whole, complete system.

Orthogonally to the abstraction layers, one must decide which aspect of a system needs to be tested. For instance, we may want to perform stress testing, which focuses on the performance of the implementation under heavy workload. Similarly, we could perform robustness testing, to explore how an implementation reacts to unspecified environments, or performance testing, to investigate how fast the implementation can perform its tasks. This list can grow indefinitely; related to our original aim of characterizing model based correctness of a system, is conformance testing, where we are interested in testing whether the behaviour of the implementation conforms to the specified behaviour.

Yet another testing distinction lies in the degree of visibility we assume in the system implementation to have. One possibility, called black-box testing, arises when we have only the interface information regarding the implementation; its inner workings are completely opaque to us. In the other extreme we find white-box testing, where the full internal details

of the system implementation are exposed. This case has certainly more appeal, as the testing procedure can exploit this information; however, a system tester may not be always so fortunate as to get access to this information. Naturally, the degree of visibility between black-box and white-box testing may vary, leading to a scale of gray-box testing. For instance, it could happen that in a system divided into modules one knows the overall module structure, but however lacks information regarding each module implementation.

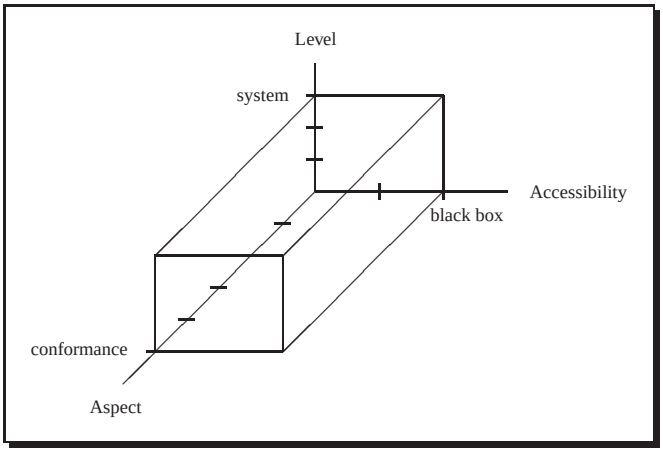


Figure 1.2: Types of testing considered in this thesis

In this thesis we focus on conformance testing at the system level with black box implementations, as presented in Figure 1.2.

Model based testing When the intended behaviour of a system is described with a model, and in addition the test cases are derived from that model, we are performing model-based testing. In this case we can use the model (or specification), which is (ideally) provided by the designer, to define the notion of correctness for a given system implementation.

Hence, the model allows us to give a correctness verdict, based on observations made during the test execution: a positive result gives us confidence in the correct functioning of the system implementation – a negative result, in turn, indicates the presence of an error.

It must be noted that the specification is a fundamental ingredient for model based testing: without it, verdicts cannot be reached, as we cannot tell whether an implementation behaviour is correct or not. Still, the necessity of having a model as a description is not restrictive, as we can simply test on the basis of crude models, or even try to develop models as we test.

Formal methods in conformance testing In practice, system designers usually provide behaviour specifications which are simply written in natural language, like English, Dutch, or Spanish. This causes several complications. Firstly, it is difficult to describe a system fully just with words, and so the specifications are typically incomplete. Even worse is the fact that natural language is inherently ambiguous, which can lead to different interpretations or inconsistencies.

As discussed, specifications are crucial for testing a system implementation. To obtain good testing results, it is also vital that specifications are precise and unambiguous. Otherwise, if we do not understand precisely what a system is supposed to do, surely we will not be able to see whether the system implementation performs appropriately.

Formal methods help us to solve this problem; they provide us with mathematical and logical techniques to specify and model systems rigorously [51, 50, 30, 27]. Formal specifications, being precisely defined mathematical objects with a clear semantical meaning, allow us to reason soundly about them. Moreover, specifications written in formal languages can be manipulated by specialized tools, allowing us to automate (part of) the analysis.

Conformance testing means that we are checking the functional correctness of a black-box system under test with respect to a formal specification. In particular, we are going to focus on specifications whose formal semantics are expressed in terms of labelled transition systems (e.g., LOTOS [35], PROMELA [34]). Transition systems are well-studied and used to give semantics to process algebraic languages like CCS or CSP [46, 33]. Having a formal specification allows us to automate the test generation phase, since test cases can be derived algorithmically in an efficient, effective, and systematic fashion.

Testing and verification Unfortunately, there exists an old myth stating that the practical and operational (or "dirty hands") approach of testing cannot be combined with the clean mathematical theories associated with formal verification. However, this wrong belief is fortunately changing. The activities of formal testing and verification are in fact complementary. Verification aims at proving properties about systems by formal manipulation of the mathematical model of the system. Thus, verification can give certainty about the satisfaction of a required property based only on the model of the system, not on its real physical implementation. Testing, on the other hand, is performed by exercising the real, executable implementation. This is of course a highly desirable property; the price to pay for this advantage is that testing cannot be, in general, complete: its result is based on observations of only a small subset of all (usually infinitely many) possible instances of the system behaviour. However, people from industry usually consider verification as impracticable and not applicable to realistically-sized systems, preferring testing as a validation method.

In this thesis we exploit the fact that verification and testing are complementary in the following way. Our specifications are assumed to be good models of the desired behaviour (this could be established, for instance, using verification techniques like model checking where system properties, represented by logical formulas, are checked upon the specification transition system). Critical systems, like airplanes or nuclear power plants, are some of the cases where this assumption applies.

Test selection As mentioned above, complete test suites usually cannot be covered in finite time for most interesting cases. Hence, the process of testing is inherently incomplete, meaning that it cannot guarantee the absence of all and every error. This implies that having a good strategy for test selection is of vital importance.

Technically, test coverage is a measure of the proportion of the implementation exercised by a test suite. These coverage measures are used to evaluate the quality of a test suite and help the tester to select test cases with maximal impact or minimum cost. Typical black-box coverage metrics are state and transition coverage of the specification [61, 42, 49]. Typical white-box testing considers statement, condition and path coverage [47, 48, 8].

A related notion (in the context of software testing) to coverage and the issue of choosing tests with maximum impact can be found in the work done on risk analysis [54, 55, 15], where the metrics consider the risk of an error to occur. In this setting, in order to be able to perform risk analysis, two ingredients are necessary. Firstly, we need to consider all different errors that could possible appear in an implementation. Secondly, we need to assign, to each error, an assessment (possibly stated as a numerical value) representing the cost of the error occurrence.

In this thesis we extend an existing model-based testing theory and we introduce a semantic based coverage. To this end we first present a formal approach for formal testing.

1.2 A formal framework for software testing

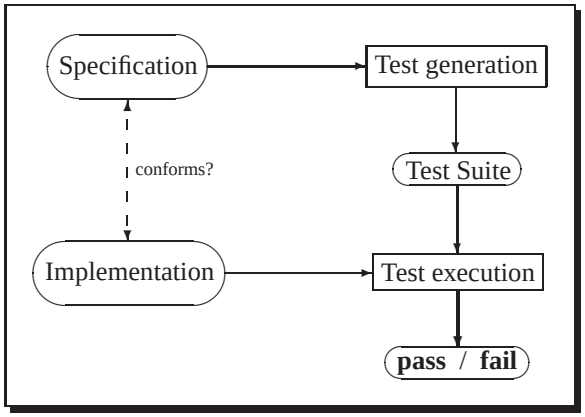


Figure 1.3: Process of formal conformance testing

In this section we illustrate a framework that helps to understand the process of formal conformance testing, independently of any specific formal method (for a more elaborate treatment of the contents of this section we refer the reader to [59]). This process is described schematically in Figure 1.3. There, the specification, the test generation, and the test suite are formal objects. Meanwhile, the implementation and the test execution (therefore also the verdict, pass or fail) are not formal. The implementation can be a real object (or part of), the test execution is an activity of feeding and reading values from the implementation. We wish to know whether the implementation is correct (does not have errors) with respect to the specification (indicated by the dashed arrow in the figure). The criterion of correctness is given by the so-called conformance relation. A decision of whether the implementation "conforms" the specification is reached by a test generation procedure that generates a test suite from the specification. So, the (generated) test suite is executed with the implementation and a verdict from the test execution is reached, telling whether the implementation does or does not conform with the specification.

To formally deal with the implementation and test execution we make the assumptions described next.

The conformance relation & the test hypothesis In order to define the conformance relation, we need three objects: the specification (object to be conformed to), the implementation (object to be checked for conformance), and the conformance relation (the criterion of conformance). We assume that there is a universe of formal specifications, called $\mathcal{SPECS}$, and a universe of implementations, called $\mathcal{IMPS}$. Then, the conformance is a relation *conforms-to* $\subseteq \mathcal{IMPS} \times \mathcal{SPECS}$. An implementation under test (called IUT) conforms to a specification (S), i.e. IUT *conforms-to* S, expresses that IUT is a correct implementation of the specification S.

However, since we are restricting ourselves to black-box testing and thus cannot see inside the implementation, we can only interact with the implementation through their external behaviour. Moreover, implementations can be physical objects (as pieces of hardware or software), therefore they are not always suitable for formal reasoning. Thus, as a way to give a formal definition of conformance, we make the assumption that any real implementation ($IUT \in \mathcal{IMPS}$) can be modelled as a formal object, i.e. $i_{IUT} \in \mathcal{MODS}$, where $\mathcal{MODS}$ is the universe of formal models. This assumption is known as the test hypothesis [11]. An important detail of this assumption is that it is supposed that the model exists but not that the model is actually known a priori.

The test hypothesis allows one to reason with real implementations just as if they are formal objects. Moreover, it allows to express conformance as a formal relation [14, 64], called the implementation relation: $\mathbf{imp} \subseteq \mathcal{MODS} \times \mathcal{SPECS}$. In this way, an implementation IUT conforms to a specification S if and only if the model of the implementation, $i_{IUT} \in \mathcal{MODS}$, is $\mathbf{imp}$-related to S, i.e. $i_{IUT}\ \mathbf{imp}\ S$.

The observations As we say before, testing studies the behaviour of an implementation by experimenting with it and observing its reaction. These experiments are called test cases, and the process of applying them to the implementation is called test execution.

Once we have generated a set of test cases, their execution is not a formal activity, as it is a matter of mechanically feeding to and reading values from the implementation. Nevertheless, the analysis of the results and the final verdict are still influenced and simplified by the formal framework. (Note that the test generation and execution do not need to be always sequential. One technique, called on-the-fly testing, allows to combine the test generation and execution. Here, the tests are executed while they are generated.)

The universe of formal test cases is called $TESTS$. Thus, a test execution is a process called $\mathcal{EXEC}(t, IUT)$, built from a test $t \in TESTS$ and an implementation $IUT \in \mathcal{IMPS}$. During the execution, process observations can be made or recorded; we call $\mathcal{OBS}$ the universe of observations (interpreting them as formal objects). Thus, an execution $\mathcal{EXEC}(t, IUT)$ returns a subset of $\mathcal{OBS}$.

Now, $\mathcal{EXEC}(t, IUT)$ corresponds to a physical execution of a test case in a real implementation, and therefore is not a formal concept. To formalize it we define the observation function $\mathsf{obs} : TESTS \times \mathcal{MODS} \to \mathcal{P}(\mathcal{OBS})$. In this way, $\mathsf{obs}(t, i_{IUT})$ models formally the real test execution of $\mathcal{EXEC}(t, IUT)$. Moreover, using the concepts introduced earlier we can reformulate the test hypothesis as for all IUT in $\mathcal{IMPS}$, there exists i_{IUT} in $\mathcal{MODS}$ such that for all t in $TESTS$, then $\mathcal{EXEC}(t, IUT) = \mathsf{obs}(t, i_{IUT})$. Meaning that for each real implementation IUT, it is assumed that there exists a model i_{IUT} such that executing t against IUT yields exactly the some observations as executing t against i_{IUT}. Hence, it is not possible to distinguish them in a black-box performing tests in $TESTS$.

The verdict Using the observations, we can determine whether an implementation is correct or wrong. To this end we introduce the verdict function for each t a test, $v_t : \mathcal{P}(\mathcal{OBS}) \rightarrow \{\mathbf{fail}, \mathbf{pass}\}$, and using the previous definition we can define IUT **passes** $t \triangleq v_t(\mathcal{EXEC}(t, IUT)) = \mathbf{pass}$, and its extension to test suites IUT **passes** $\mathbf{T} \triangleq \forall\ t \in \mathbf{T} : IUT$ **passes** t. On the other hand, we say that an implementation fails a test suite $\mathbf{T}$ if it does not pass it, i.e. IUT **fails** $\mathbf{T} \triangleq \neg(IUT$ **passes** $\mathbf{T})$.

Properties of conformance testing Because we want the conformance relation to be assessed through test cases, ideally, we would like to have a test suite $\mathbf{T}_S$, for a given specification S, such that

$$IUT \textit{ conforms-to } S \quad \text{if and only if} \quad i \textbf{ passes } \mathbf{T}_S$$

When a test suite has this property we say it is *complete*. Thus, a complete test suite is able to distinguish between implementations that do conform to their specifications and implementations that do not.

Unfortunately, this requirement is very strong since complete tests suites are usually infinite. Nevertheless, it is possible to divide this requirement in two

$$\text{if} \quad IUT \textit{ conforms-to } S \quad \text{then} \quad i \textbf{ passes } \mathbf{T}_S$$

this is known as *soundness*, meaning that all correct implementations pass the test suite; and

$$\text{if} \quad i \textbf{ passes } \mathbf{T}_S \quad \text{then} \quad IUT \textit{ conforms-to } S$$

which is known as *exhaustiveness*, meaning that all non-correct implementations are detected by the test suite. These properties can also be shown through formal models as $\forall\ i \in \mathcal{MODS} : i$ **imp** S if and only if for all tests t, $v_t(\mathsf{obs}(t, i)) = \mathbf{pass}$. Then, IUT **passes** $\mathbf{T}$ if and only if IUT *conforms-to* S. So, if the completeness property is proved at the level of models, assuming the test hypothesis, the conformance of an implementation with respect to its specification can be decided by the testing procedure.

The derivation procedure The algorithm that produces sound and/or exhaustive test suites from a specification given an implementation relation; this is called the *test derivation*. We define this procedure as a function $\mathsf{der}_{\mathbf{imp}} : \mathcal{SPECS} \rightarrow \mathcal{P}(TESTS)$. Following the requirement of soundness of test suites, such a function should produce sound test suites for any specification S in $\mathcal{SPECS}$.

1.2.1 The ioco testing relation

In this thesis we focus on a particular conformance relation, the **ioco** testing relation (which is an instantiation of the presented framework). The **ioco** testing relation relates systems described as labelled input-output transition systems [58]. It includes nondeterminism and quiescence (the absence of outputs). Also it requires implementations to accept always all inputs. Briefly, **ioco** allows implementations to have only outputs that are predicted by the

specification, with a special consideration of no output (quiescence) as a kind of output. Thus, implementations can be more deterministic than specifications. In Chapter 2 we formally describe **ioco**.

1.3 Research questions

As we mentioned in the previous section, the process of formal conformance testing involves mainly three ingredients: physical aspects (like black-box implementation and real execution of test cases); formal aspects (like specifications, implementations, verdicts, and test derivations), and assumptions (like the test hypothesis). Focusing on the formal ingredients, one can see that each of those elements can be improved independently, or specifically optimized for a particular task; this, in turn, benefits the whole testing framework, resulting in more accurate, effective and efficient testing.

Time in testing There are many systems where the consideration of time is crucial. Even, in same cases, small delays can cause huge problems. The necessity of consider time in some theories has long been recognized and many formal method have been extended with time. In this thesis we consider extensions of the **ioco** testing theory. Our first extensions address the following question:

> **Research question 1:** In which useful ways can we extend **ioco** testing theory to be able to test real-time systems in an accurate manner?

Coverage in testing Existing coverage criteria for test suites are usually defined in terms of syntactic characteristics of the implementation under test or its specification [61, 42, 49, 47, 48, 8]. A disadvantage of this syntactic approach is that different coverage figures are assigned to systems that are behaviorally equivalent, but syntactically different.

Moreover, those coverage metrics do not take into account risks, meaning that they do bear in mind the fact that certain failures are more severe than others. Therefore, it should be devoted more testing effort to cover more important bugs, while less critical system parts can be tested less thoroughly. Thus, our second extension address the following question:

> **Research question 2:** In a black-box testing setting with failure risks, how can we measure a given test suite independently of the specification syntactic aspects?

These two questions are in fact complementary; whereas Research question 1 extends the scope of testable systems, Research question 2 studies how to measure a given test suite allowing to find the optimal test suite (the one with more important bugs).

1.4 Structure of the thesis

Our thesis contributions, developed in the context described by the **ioco** testing theory, are depicted in Figure 1.4. After introducing formal testing and more specifically the **ioco** theory,

we present our extensions: a testing theory considering both real-time (**tioco**) and real-time plus channels (**mtioco**$_{\mathcal{M}}$) in Chapters 3 and 4 respectively, and a semantic coverage criterion in Chapter 5.

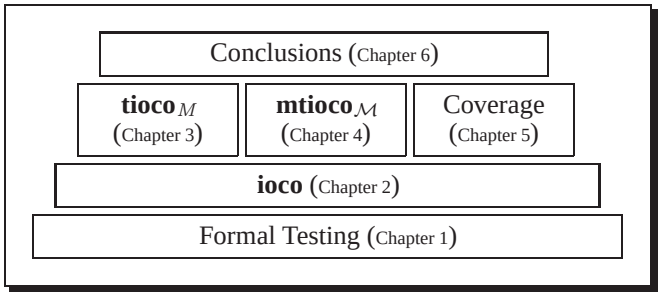


Figure 1.4: Thesis skeleton

Testing labelled input-output transition systems (Chapter 2) This chapter is an introduction to the theory of formal methods applied to testing labelled input-output transition systems. In particular, we introduce the **ioco** testing theory [58], where labelled input-output transition systems is the formalism used. **ioco** is a formal conformance testing relation that includes nondeterminism and quiescence (briefly, a system is in a quiescent state when it does not produce outputs). In addition, quiescence is considered an observable action. Implementations are required to be input-enabled (meaning that all input actions should be enabled at any time).
This chapter is the basic theory that the subsequent chapters build upon.

Testing timed labelled input-output transition systems (Chapter 3) We propose an extension of the **ioco** testing theory with real-time that enables test generation for timed input-output labelled transition systems. Our treatment is based on an operational interpretation of the notion of quiescence in the context of real-time behaviour. This gives rise to a family of implementation relations parameterized by observation durations for quiescence. We define a nondeterministic (parameterized) test generation procedure that generates test cases that are sound with respect to the corresponding implementation relation. Also, the test generation is shown to be exhaustive in the sense that for each non-conforming implementation a test case can be generated that detects the non-conformance.
We conclude this chapter showing a result that relates our proposed timed extension with the non-timed approach. Part of this work appears in [1] (joint work with Ed Brinksma) and in [4] (joint work with Mathias Röhl).

Testing timed labelled multi input-output transition systems (Chapter 4) Our starting point is the formal conformance timed testing relation defined in Chapter 3. We relax the input-enableness assumption (required in the previous chapter) by asking the input and output sets to be partitioned (we called these partitions channels), also we allow

some input sets to be enabled while others remain disabled. Moreover, we relax the general bound M (used in timed systems to detect quiescence), and allow different bounds for different sets of outputs. We propose a new testing relation for timed input-output labelled transitions systems that have the input and output set partitioned in subsets. A test derivation procedure which is nondeterministic and parameterized (by the set of bounds) is further developed, and shown to be sound and exhaustive with respect to our new testing relation.
We end this chapter by showing a result that relates the new proposed channel timed extension with the timed approach presented in Chapter 3. Part of this work appears in [2] (joint work with Ed Brinksma).

Semantic coverage in testing (Chapter 5) In this chapter we introduce a semantic approach to test coverage. Our starting point is a weighted fault model, which assigns a weight, in the specification, to each potential error in an implementation. We define a framework to express coverage measures that express how well a test suite covers such a specification, taking into account the error weights. Since our notions are semantic, they are insensitive to replacing a specification by one with equivalent behaviour. We present several algorithms that, given a certain minimality criterion, compute a minimal test suite with maximal coverage. These algorithms work on a syntactic representation of weighted fault models as fault automata.
We end this chapter with an illustrating our approach by analyzing and comparing a number of test suites for a chat protocol. Part of this work appears in [3] (joint works with Ed Brinksma and Mariëlle Stoelinga).

CHAPTER 2

Testing labelled input-output transition systems

2.1 Introduction

This chapter introduces the relevant, preliminary theories and relations describing the **ioco** testing theory developed originally by Tretmans [58]. As the rest of this thesis builds upon the **ioco** testing theory, this chapter provides the basic setting necessary for the rest of this thesis development.

The **ioco** testing theory has several desirable properties that make it attractive as an initial setting to which one can start building on, as in this thesis. Among these properties we find theoretical and practical advantages. As theoretical aspects, **ioco** has a clean and precise theory. Particularly, it allows non-determinism, as Hoare [33] says "There is nothing mysterious about non-determinism, it arises from the deliberated decision to ignore the factors which influence the selection". Also **ioco** considers quiescence as observable, allowing to distinguish systems that can not be distinguish with out quiescence. As practical aspects, **ioco** serves as a base theory for several successful testing tools, e.g. TORX [10] and TGV [26].

Organization of the chapter In Section 2.2 we survey labeled input-output transition systems (LTS): we formally define LTS by describing their notation and properties. Section 2.3 presents some implementation relations that can be applied to LTS; in particular, we describe the **ioco** testing relation [58]. Section 2.5 then shows how these models and implementation relations can be put into practice; by focusing on the **ioco** test implementation relation, a test derivation and a test execution procedure are presented. In Section 2.6 we show that the test generation procedure presented is sound and exhaustive with respect to **ioco**. We conclude this chapter resalting some of the useful characteristics of the **ioco** testing theory in Section 2.7.

2.2 Labelled input-output transition systems

A labelled input-output transition system (LTS) is a system that interacts with its environment through inputs and outputs. Input actions come from (or are driven by) the environment, while output actions are actions to the environment. The internal actions or silent actions are not observable by the environment. We use L to denote the action set and the special label

$\tau \notin L$ to represent internal actions. For arbitrary $L' \subseteq L$, L'_τ denotes $L' \cup \{\tau\}$.

Definition 2.2.1. *A* labeled input-output transition system (LTS) A *is a 4-tuple* $\langle Q, q^0, L, T\rangle$*, where*

- Q *is a countable, non-empty set of states*
- $q^0 \in Q$ *is the initial state*
- L *is a countable set of labels. We assume that* L *has a disjoint partition into* input labels, I*, and* output labels, O
- $T \subseteq Q \times L_\tau \times Q$ *is the transition relation*

We denote the components of A *by* Q_A, q^0_A, L_A, *and* T_A*, respectively. We omit the subscript* A *if its meaning is clear from the context. The class of all labeled input-output transition systems over* L *is denoted by* $\mathrm{LTS}(L)$*. When we need to make the input set and the output set explicit we write* $\mathrm{LTS}(I, O)$.

To specify that a certain label (l) is either an input action or an output action, we simply suffix a special symbol. Thus, for input actions we add an interrogation mark, and write $l?$. Similarly, for output actions we add an exclamation mark, writing $l!$.

A transition $(q, l, q') \in T$ is denoted as $q \xrightarrow{l} q'$. A path π is a finite or infinite sequence of transitions

$$\pi = q_0 \xrightarrow{l_1} q_1 \xrightarrow{l_2} q_2 \xrightarrow{l_3} \cdots \xrightarrow{l_{n-1}} q_{n-1} \xrightarrow{l_n} q_n (\rightarrow \dots)$$

with $q_z \xrightarrow{l_{z+1}} q_{z+1} \in T$ and $z = 0, 1, \dots, n, \dots$.

We denote by $\mathsf{paths}(A)$ the set of all paths in A. The set of all finite sequences of actions over L is denoted by L^*, while ε denotes the empty sequence. Moreover, if $\sigma_1, \sigma_2 \in L^*$ then with $\sigma_1 \cdot \sigma_2$ we denote the concatenation of σ_1 with σ_2.

An LTS is called *strongly convergent* if it does not have any infinite paths of internal actions. We require all LTSs that we work with to be strongly convergent systems.

Definition 2.2.2. *Let* $A = \langle Q, q^0, L, T\rangle$ *be an LTS with* $q, q', q_k \in Q$; $l_k \in L_\tau$; $\beta, \beta_k \in L, 1 \leq k \leq n$ *and* $\sigma \in L^*$*. Then*

$$\begin{array}{lcl}
q \xrightarrow{l_1 \cdots l_n} q' & \triangleq & \exists\ q_0, \dots, q_n : q = q_0 \xrightarrow{l_1} q_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} q_n = q' \\
q \xrightarrow{l_1 \cdots l_n} & \triangleq & \exists\ q' : q \xrightarrow{l_1 \cdots l_n} q' \\
q \not\xrightarrow{l_1 \cdots l_n} & \triangleq & \nexists\ q' : q \xrightarrow{l_1 \cdots l_n} q' \\
q \stackrel{\varepsilon}{\Rightarrow} q' & \triangleq & q = q' \text{ or } q \xrightarrow{\tau \cdots \tau} q' \\
q \stackrel{\beta}{\Rightarrow} q' & \triangleq & \exists\ q_1, q_2 : q \stackrel{\varepsilon}{\Rightarrow} q_1 \xrightarrow{\beta} q_2 \stackrel{\varepsilon}{\Rightarrow} q' \\
q \stackrel{\beta_1 \cdots \beta_n}{\Longrightarrow} q' & \triangleq & \exists\ q_0, \dots, q_n : q = q_0 \stackrel{\beta_1}{\Rightarrow} q_1 \stackrel{\beta_2}{\Rightarrow} \cdots \stackrel{\beta_n}{\Rightarrow} q_n = q' \\
q \stackrel{\sigma}{\Rightarrow} & \triangleq & \exists\ q' : q \stackrel{\sigma}{\Rightarrow} q' \\
q \not\stackrel{\sigma}{\Rightarrow} & \triangleq & \nexists\ q' : q \stackrel{\sigma}{\Rightarrow} q'
\end{array}$$

We say that a state q accepts the action l if and only if $q \xrightarrow{l}$. When an LTS is able to accept all input actions at any state, it is called *input-enabled*. This notion can be slightly relaxed into a weak form, considering an LTS to be able to accept all input actions at any state or any reachable state through τ's transitions. Then, a state q weakly accepts the action l if and only if $q \stackrel{l}{\Rightarrow}$. We say an LTS A is *deterministic* if does not exists a transitions labeled with a τ action, and all transitions with the same source states and actions, have the same target state.

Definition 2.2.3. *Let $A = \langle Q, q^0, L, T \rangle$ be an LTS(I, O), then*

$$
\begin{array}{lcl}
A \textit{ is} \text{ input-enabled} & \triangleq & \forall\ q \in Q : \forall\ a \in I : q \xrightarrow{a} \\
A \textit{ is} \text{ weak input-enabled} & \triangleq & \forall\ q \in Q : \forall\ a \in I : q \stackrel{a}{\Rightarrow} \\
A \textit{ is} \text{ deterministic} & \triangleq & \forall\ q \xrightarrow{l} q' \in T : l \neq \tau \textit{ and} \\
 & & \textit{if } q \xrightarrow{l} q', q \xrightarrow{l} q'' \in T, \textit{ then } q' = q''
\end{array}
$$

In order to be able to keep track precisely of the occurrence of input and output actions, we require LTS implementations to be input-enabled. Symmetrically, the environment of an LTS implementation is assumed to be able always to accept any output action.

Example 2.2.4. *Consider the LTS $A = \langle Q, q^0, L, T \rangle$ where: $Q = \{q_0, q_1, q_2\}$; $q^0 = q_0$; $L = I \cup O$ with $I = \{$card, ask_money$\}$ and $O = \{$give_money$\}$; and $T = \{(q_0,$ card$, q_1), (q_1,$ ask_money$, q_2), (q_2,$ give_money$, q_0)\}$.*

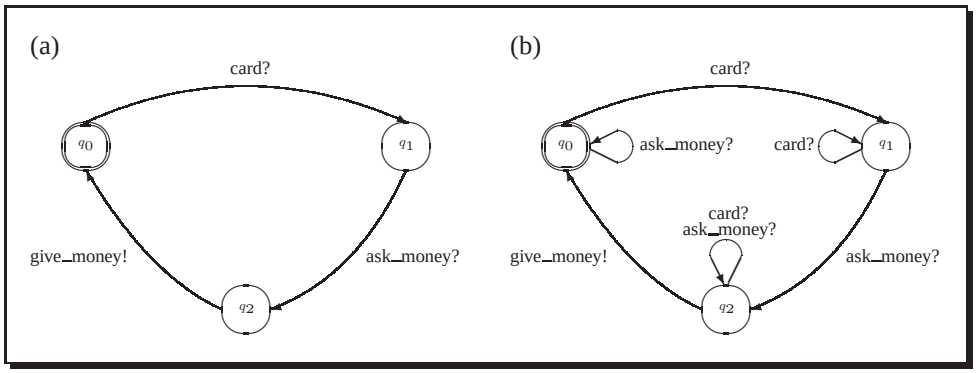


Figure 2.1: A cash machine and one of its input-enabled versions

This deterministic LTS is depicture in Figure 2.1 (a). The system A has an initial state q_0 denoted with double circles. Putting a card we arrive to state q_1. From q_1 it is possible to ask for money and go to q_2 where we leave receiving money, and again we are in the initial state. We can observe in A the path$\pi = q_0 \xrightarrow{card} q_1 \xrightarrow{ask_money} q_2 \xrightarrow{give_money} q_0$.

There are several ways to make this LTS input-enabled, in Figure 2.1 (b), we present one

option. The self-loop with ask_money in state q_0, the one with card in state q_1 and the one with card and ask_money in state q_2 makes A input enabled in every state.

Definition 2.2.5. *Let $A = \langle Q, q^0, L, T\rangle$ be an LTS, $q \in Q$ and $Q' \subseteq Q$, then*

$$\begin{array}{lcl} \mathsf{traces}(A) & \triangleq & \{\sigma \in L^* \mid q^0 \stackrel{\sigma}{\Rightarrow}\} \\ \mathsf{init}(q) & \triangleq & \{l \in L_\tau \mid q \stackrel{l}{\rightarrow}\} \\ \mathsf{der}(q) & \triangleq & \{q' \mid \exists\ \sigma \in L^* : q \stackrel{\sigma}{\Rightarrow} q'\} \\ q\ \mathbf{after}\ \sigma & \triangleq & \{q' \mid q \stackrel{\sigma}{\Rightarrow} q'\} \\ Q'\ \mathbf{after}\ \sigma & \triangleq & \bigcup_{q \in Q'} (q\ \mathbf{after}\ \sigma) \end{array}$$

Given an LTS A, $\mathsf{traces}(A)$ captures all possible observable behaviours of A. In other words, a trace is the sequence of input and output actions present in a finite path in $\mathsf{paths}(A)$. Moreover, given a trace σ, by $|\sigma|$ we denote the length of that trace. The init and der sets collect the allowed actions and the reachable states, respectively, from a given state. The **after** set is composed of all the states that can be reached after a certain trace is performed.

Example 2.2.6. *As an example of Definition 2.2.5, in Figure 2.1(a), we see that the trace σ = card · ask_money · give_money is in $\mathsf{traces}(A)$, $\mathsf{init}(q_2) = \{$give_money$\}$, $\mathsf{der}(q_0) = \{q_0, q_1, q_2\}$ and $(q_0\ \mathbf{after}\ \sigma) = \{q_0\}$.*

In fact, in this setting the determinism of an LTS can be clearly characterized, as shown in the following proposition.

Proposition 2.2.7. *Let $A = \langle Q, q^0, L, T\rangle$ be an LTS, then*

$$A \text{ is deterministic} \quad \text{if and only if} \quad \forall\ \sigma \in L^* : \forall\ q \in Q : |\ (q\ \mathbf{after}\ \sigma)\ | \leq 1$$

If $\sigma \in \mathsf{traces}(A)$ and $q \in Q$, then in case $(q\ \mathbf{after}\ \sigma)$ is a singleton set, $\{q'\}$, we abuse the notation to denote this element, q'.

Proof.

[$\Leftarrow$ If A is deterministic then $\forall\ \sigma \in L^* : \forall\ q \in Q : |\ (q\ \mathbf{after}\ \sigma)\ | \leq 1$]
By induction over the length ($|\sigma|$) of σ. Let q be any state in Q.
Let $|\sigma| = 1$ (the case $\sigma = \epsilon$ is trivial), then by Definition 2.2.3 of determinism we have $|\ q\ \mathbf{after}\ \sigma\ | \leq 1$.
Suppose for all $|\sigma| < n$ then $|\ q\ \mathbf{after}\ \sigma\ | \leq 1$.
Let $|\sigma| = n$ then $\sigma = \sigma' \cdot l$. By Definition 2.2.5 of after : $|\ q\ \mathbf{after}\ \sigma' \cdot l\ | = |\ (q\ \mathbf{after}\ \sigma')\ \mathbf{after}\ l\ |$. Now, using base case $(\exists\ q' : \{q'\} = q\ \mathbf{after}\ \sigma') \vee (q\ \mathbf{after}\ \sigma' = \emptyset)$ then by Definition 2.2.5 of after and Definition 2.2.3 of determinism we have $|\ q\ \mathbf{after}\ \sigma\ | \leq 1$.

[$\Rightarrow$ If $\forall\ \sigma \in L^* : \forall\ q \in Q : |\ (q\ \mathbf{after}\ \sigma)\ | \leq 1$ then A is deterministic]
Assume that for all trace in L^* and for all state q in $Q : |\ q\ \mathbf{after}\ \sigma\ | \leq 1$.

Then for all action l in L and for all state q in Q : $\mid q \ \mathbf{after}\ l \mid \leq 1$. This means that for all action l in L and for all state q in Q : if $q \xrightarrow{l} q' \wedge q \xrightarrow{l} q''$ then $q' = q''$. Using that LTS are strongly convergent we have that for all action l in L and for all state q in Q : if $q \xrightarrow{l} q' \wedge q \xrightarrow{l} q''$ then $q' = q'' \wedge q \overset{\tau}{\not\rightarrow}$. Now, by Definition 2.2.3 of determinism we have that A is deterministic.

□

When a state is unable to perform an action, we say that the state refuses that action. In case a state q has a set of actions that are not accepted, i.e. the set has an empty intersection with $\mathsf{init}(q)$, we say that the state q has this set as refusal. This latter concept is precisely defined in the following definition. Note that the case with only one action can also be described with a set with only one element.

In the particular case that a state refuses all actions, i.e. it does not have any outgoing transition, we say that the state is in *deadlock*.

Definition 2.2.8. *Let $A = \langle Q, q^0, L, T\rangle$ be an LTS with $l \in L$, $q \in Q$ and $L' \subseteq L$, then*

$$\begin{array}{lcl} q \ \mathbf{refuses}\ l & \triangleq & q \overset{l}{\not\rightarrow} \wedge\ q \overset{\tau}{\not\rightarrow} \\ q \ \mathbf{refuses}\ L' & \triangleq & \forall\ l \in L'_\tau : q \overset{l}{\not\rightarrow} \\ q \ \mathbf{deadlocks} & \triangleq & q \ \mathbf{refuses}\ L \end{array}$$

To be able to represent refusal sets in traces, we extend the transition relation in the LTS as follows. Given $A = \langle Q, q^0, L, T\rangle$ an LTS, its transition relation (T) is extended to denote the set of refused actions explicitly. Then, a transition $q \xrightarrow{L'} q$ is added for each state q that refuses L':

$$q \xrightarrow{L'} q \quad \text{if and only if} \quad q \ \mathbf{refuses}\ L'$$

In this way the transition relation that was $T \subseteq Q \times L_\tau \times Q$ now becomes $T \subseteq Q \times (L_\tau + \mathcal{P}(L)) \times Q$, where $\mathcal{P}(L)$ denotes the power set of L.

In the special case where a state cannot produce output actions ($L' = O$), and can only be activated by (further) supply of input actions, we say that the state is *quiescent*:

$$q \ \mathbf{quiescent} \quad \triangleq \quad \forall\ l \in O_\tau : q \overset{l}{\not\rightarrow}$$

We write $\delta(q)$ to denote that state q is quiescent.

Hence, the extension of the transition relation can be also applied to quiescence, with the aim of treating quiescence as an observable event (i.e., the absence of outputs). Using the new action δ ($\delta \notin L_\tau$), a transition $q \xrightarrow{\delta} q$ is added for each quiescent state q:

$$q \xrightarrow{\delta} q \quad \text{if and only if} \quad \delta(q)$$

Example 2.2.9. *Recall Figure 2.1 (a). If we consider $L' = \{ask\_money\}$ then q_0 and q_2* **refuses** *L'. Moreover, we can recognize that q_0 and q_1 do not have any outgoing transitions labelled with output actions, then both states are quiescent. In Figure 2.2 we show the cash machine with its extension on the quiescent states q_0 and q_1.*

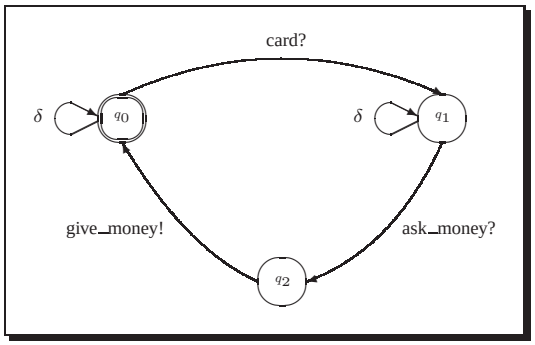


Figure 2.2: A cash machine with quiescent states explicitly denoted by self-loop quiescent transitions

We can now introduce some new notions of traces, viz. *failure traces* (i.e. a sequence of actions and refusals) and *suspension traces* (i.e. a sequence of actions and quiescence).

Definition 2.2.10. *Let $A = \langle Q, q^0, L, T \rangle$ be an LTS with the transition relation extended with refused transitions ($q \xrightarrow{L'} q$ in case q refuses L'). Then, the* failure traces *of A are*

$$\mathsf{Ftraces}(A) \quad \triangleq \quad \{\sigma \in (L + \mathcal{P}(L))^* \mid q^0 \stackrel{\sigma}{\Rightarrow}\}$$

Definition 2.2.11. *Let $A = \langle Q, q^0, L, T \rangle$ be an LTS with the transition relation extended with quiescent transitions ($q \xrightarrow{\delta} q$ in case $\delta(q)$). Then, the* suspension traces *of A are*

$$\mathsf{Straces}(A) \quad \triangleq \quad \{\sigma \in L_\delta^* \mid q^0 \stackrel{\sigma}{\Rightarrow}\}$$

usually we write L_δ to denote $L \cup \{\delta\}$.

There is a tight relationship between Straces and Ftraces:

Proposition 2.2.12. *Let $A = \langle Q, q^0, L, T \rangle$ be an LTS with the transition relation extended with refused transitions, then*

$$\mathsf{Straces}(A) \quad = \quad \mathsf{Ftraces}(A) \cap (L + \{O\})^*$$

where O is written as δ.

Proof.
Direct from Definition 2.2.11.

□

Considering that implementations are required to be input-enabled, it makes sense to believe that a system can be characterized by its output actions (considering quiescence as one of them). With this idea in mind, we introduce the notion of output set. For labeled input-output transition systems, all output actions that are enabled in state q (including the quiescent action δ) are collected into the output set $\mathbf{out}(q)$.

Definition 2.2.13. *Let $A = \langle Q, q^0, L, T\rangle$ be an LTS with $q \in Q$ and $Q' \subseteq Q$, then*

$$\begin{array}{lcl} \mathbf{out}(q) & \triangleq & \{l \in O \mid q \xrightarrow{l}\} \cup \{\delta \mid \delta(q)\} \\ \mathbf{out}(Q') & \triangleq & \bigcup_{q \in Q'} \mathbf{out}(q) \end{array}$$

Intuitively, the output set collects all possible observable outputs of the system.

Example 2.2.14. *The trace $\sigma =$ card·{give_money}·ask_money·{ask_money}·give_money is a trace in* $\mathsf{Ftraces}(A)$ *(from Figure 2.2). The trace $\sigma = \delta \cdot$ card·ask_money·give_money·δ is a trace in* $\mathsf{Straces}(A)$. *As example of an output set we can see that $\delta \in \mathbf{out}(q_0)$ and give_money $\in \mathbf{out}(q_2)$.*

2.3 Conformance relations

An implementation relation R (or conformance relation) is a relation that defines a notion of correctness between an implementation i, and a specification S. If the implementation relation holds (i.e. $(i, S) \in R$) we say that the implementation conforms to the specification. There are several conformance relations that have been applied to LTSs, and in this section we present some of them. For more details, the reader may consult surveys such as [22, 58].

As we already mentioned in Section 2.2, we require implementations to be input-enabled. For specifications we allow more freedom and do not require them to be input-enabled. In the following we present the *trace equivalence*, *failure trace equivalence* and the *suspension trace equivalence* and the corresponding preorders.

Definition 2.3.1. *Let $A_1 = \langle Q_1, q_1^0, L_1, T_1\rangle$ be an input-enabled implementation in LTS and $A_2 = \langle Q_2, q_2^0, L_2, T_2\rangle$ be a specification in LTS with $L_1 = L_2$, then*

- trace equivalence and preorder
  $A_1 \approx_{tr} A_2 \triangleq \mathsf{traces}(A_1) = \mathsf{traces}(A_2)$
  $A_1 \leq_{tr} A_2 \triangleq \mathsf{traces}(A_1) \subseteq \mathsf{traces}(A_2)$
- failure trace equivalence and preorder
  $A_1 \approx_{ft} A_2 \triangleq \mathsf{Ftraces}(A_1) = \mathsf{Ftraces}(A_2)$
  $A_1 \leq_{ft} A_2 \triangleq \mathsf{Ftraces}(A_1) \subseteq \mathsf{Ftraces}(A_2)$

- suspension trace equivalence and preorder
  $A_1 \approx_{st} A_2 \quad \triangleq \quad \mathsf{Straces}(A_1) = \mathsf{Straces}(A_2)$
  $A_1 \leq_{st} A_2 \quad \triangleq \quad \mathsf{Straces}(A_1) \subseteq \mathsf{Straces}(A_2)$

The intuition behind trace equivalence is that an implementation conforms to a specification if an external observer can not notice the difference in traces from both systems. In the presence of non-determinism there are systems that can not be distinguished by this equivalence, i.e. Example 2.4.2. The failure trace equivalence is a stronger relation than trace equivalence because it rejects implementations using trace enreached with information about the actions that can not be performed (i.e. refused sets). The suspension trace equivalence is also stronger than trace equivalence, but weaker than failure trace equivalence. Suspension trace equivalence compare traces enreached only with information about whether a system is or is not in a quiescent state.

Trace and failure trace relations can be applied to systems without a clear distinction of input and outputs. On the other hand, suspension trace relations require an explicit distinction between inputs and output, because they are needed to recognize quiescence.

There are two properties that we already required that can be used to make the relation more accurate. Firstly, we required an implementation to be input-enabled. This implies that if an implementation fails, it fails with an output action. Using this knowledge we can use the output set, from Definition 2.2.13, to check for inclusion of output actions. Secondly, when implementations are considered black boxes, then it make sense to consider traces that are not specified by the specification (in contrast with previous definitions, where we use any trace to test, even if its behaviour is not specified by the specification).

With these two new ideas we present the trace conformance relation (also known as ioconf), which is expressed as:

$$A_1 \; trconf \; A_2 \quad \triangleq \quad \forall \; \sigma \in \mathsf{traces}(A_2) : \mathbf{out}(A_1 \; \mathbf{after} \; \sigma) \subseteq \mathbf{out}(A_2 \; \mathbf{after} \; \sigma)$$

This definition expresses that unspecified behaviours are not tested (i.e. we only consider traces from A_2). As a (convenient) consequence, we can test using incomplete specifications; and complete specifications can of course be considered also.

Similarly, it is possible to define (ftconf) with inclusion of failure traces:

$$A_1 \; ftconf \; A_2 \quad \triangleq \quad \forall \; \sigma \in \mathsf{Ftraces}(A_2) : \mathbf{out}(A_1 \; \mathbf{after} \; \sigma) \subseteq \mathbf{out}(A_2 \; \mathbf{after} \; \sigma)$$

2.4 The ioco implementation relation

Finally using inclusion of suspension traces, we define the input-output conformance relation (**ioco**), presented in the next definition.

Definition 2.4.1. *Let i be input-enabled implementation in LTS and S be a specification in LTS, then*

$$i \; \mathbf{ioco} \; S \quad \triangleq \quad \forall \; \sigma \in \mathsf{Straces}(S) : \mathbf{out}(i \; \mathbf{after} \; \sigma) \subseteq \mathbf{out}(S \; \mathbf{after} \; \sigma)$$

Informally, this means that an input-enabled implementation $i \in LTS$ is **ioco** correct with respect to a specification $S \in LTS$, if and only if, after all possible behaviours of the specification ($\forall\ \sigma \in \mathsf{Straces}(S)$), any output action b produced by the implementation ($b \in \mathbf{out}(i\ \mathbf{after}\ \sigma)$) can also occur as an output of the specification ($b \in \mathbf{out}(S\ \mathbf{after}\ \sigma)$). In particular, this should also hold for the special action quiescent (δ), modeling the absence of outputs. In this way **ioco** requires an implementation to react correctly to the traces that are explicitly mentioned in the specification. Moreover, the implementation has freedom to react in any manner to traces not explicitly specified.

Example 2.4.2. *Figure 2.3 shows two different input-enabled nondeterministic specifications for the cash machine and the different relations between them. On the left hand side we see an implementation that prescribes that after we introduce a card we can receive* 7 *Euros, and in case we introduce two cards we can receive* 2 *or* 7 *Euros. On the right hand side we see another specification, but this time, after the introduction of two card if the machine internally chooses to go to its right size through* q_2 *then it does not produce* 7 *Euros. In both cases the machine can internally go to a state (*q_1*) where it is possible to receive* 7 *Euros or to a state (*q_2*) that is quiescent.*

If we try to distinguish these two machines using the trconf conformance relations we can not, because these machines have the same set of traces. But, we can distinguish them using **ioco** *give that:* $\mathbf{out}(\mathbf{impl}\ \mathbf{after}\ \textit{card?}\delta\textit{card?}) = \{\textit{give\_7€!}, \textit{give\_2€!}\}$ *is not included in* $\mathbf{out}(\mathbf{spec}\ \mathbf{after}\ \textit{card?}\cdot\delta\cdot\textit{card?}) = \{\textit{give\_2€!}\}$.

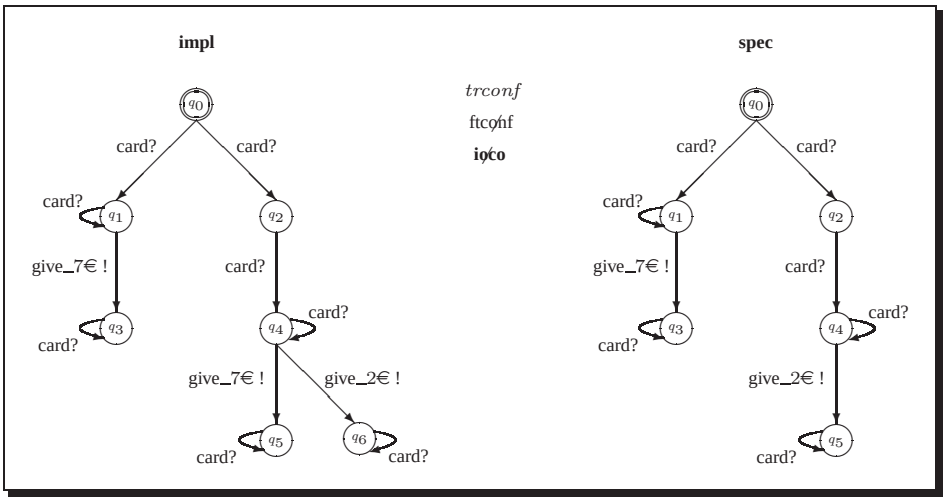


Figure 2.3: Specification of two cash machines and their relations

2.5 Test generation framework

Focusing on the **ioco** testing theory from [58], this section defines the concept of test cases, the nature of their execution, and the evaluation of their success or failure.

Definition 2.5.1.

- *A* test case $t = \langle Q, q^0, L_\delta, T \rangle$ *is an LTS such that*
  - t *is deterministic and has bounded behaviour (i.e.* $\exists\ N > 0 : \forall\ \sigma \in\ \mathsf{traces}(t) : |\sigma| \leq N$*)*
  - Q *contains the terminal states* $\mathbf{pass}$ *and* $\mathbf{fail}$*, with* $\mathsf{init}(\mathbf{pass}) = \mathsf{init}(\mathbf{fail}) = \emptyset$
  - *for any state* $q \in Q$ *of the test case with* $q \neq \mathbf{pass}, \mathbf{fail}$
    $\mathsf{init}(q) \quad = \quad \{a\}$ *for some* $a \in I$*, or*
    $\mathsf{init}(q) \quad = \quad O \cup \{\delta\}$
  - t *does not have* τ*-transitions*

  The class of test cases over I *and* O *is denoted as* $TESTS(I, O)$
- *A test suite* $\mathbf{T}$ *is a set of test cases:* $\mathbf{T} \subseteq TESTS(I, O)$

For the description of test cases, we use a process-algebraic notation with a syntax inspired by LOTOS [35]

$$B \quad \triangleq \quad l;\ B \ \mid\ B + B \ \mid\ \Sigma\, \mathcal{B}$$

where $l \in L$, $\mathcal{B}$ is a countable set of behaviour expressions, and the axioms and inference rules are:

$$\begin{array}{lcl} l \in L & \vdash & l;\ B \xrightarrow{l} B \\ B_1 \xrightarrow{l} B_1', l \in L & \vdash & B_1 + B_2 \xrightarrow{l} B_1' \\ B_2 \xrightarrow{l} B_2', l \in L & \vdash & B_1 + B_2 \xrightarrow{l} B_2' \\ B \xrightarrow{l} B', B \in \mathcal{B}, l \in L & \vdash & \Sigma\, \mathcal{B} \xrightarrow{l} B' \end{array}$$

A test run of an implementation with a test case is modeled by the synchronous parallel execution of the test case with the implementation under test. This run continues until no more interactions are possible, i.e. until a deadlock occurs.

Definition 2.5.2. *Let* t *be a test in* $TESTS(I, O)$ *and* i *be an input-enabled implementation in LTS*(I, O)*, then*

- *Running* t *with* i *is modeled by the parallel operator*
  $$||\quad :\quad TESTS(I, O) \times \mathit{LTS}(I, O) \rightarrow \mathit{LTS}(I, O)$$
  which is defined by the following inference rules

$$
\begin{array}{lll}
i \xrightarrow{\tau} i' & \vdash & t \,||\, i \xrightarrow{\tau} t \,||\, i' \\
t \xrightarrow{\delta} t', \forall\ l \in O : i \overset{l}{\not\rightarrow} & \vdash & t \,||\, i \xrightarrow{\delta} t' \,||\, i \\
t \xrightarrow{l} t', i \xrightarrow{l} i', l \in L & \vdash & t \,||\, i \xrightarrow{l} t' \,||\, i'
\end{array}
$$

- *A test run of t with i, is a trace $\sigma \in L_\delta^*$ of $t \,||\, i$ leading to a terminal state of t; σ is a test run of t and $i =$*

$$\exists\ i' : (t \,||\, i \overset{\sigma}{\Rightarrow} \mathbf{pass} \,||\, i') \text{ or } (t \,||\, i \overset{\sigma}{\Rightarrow} \mathbf{fail} \,||\, i')$$

- i **passes** t*, if all test runs do not lead to a* **fail** *state of* t

$$i \ \mathbf{passes}\ t \quad \triangleq \quad \forall\ \sigma \in L_\delta^* : \forall\ i' : t \,||\, i \overset{\sigma}{\not\Rightarrow} \mathbf{fail} \,||\, i'$$

- i **passes** $\mathbf{T}$*, if* i **passes** *all test cases in* $\mathbf{T}$

$$i \ \mathbf{passes}\ \mathbf{T} \quad \triangleq \quad \forall\ t \in \mathbf{T} : i \ \mathbf{passes}\ t$$

If i does not pass the test suite, it **fails**

$$i \ \mathbf{fails}\ \mathbf{T} \quad \triangleq \quad \exists\ t \in \mathbf{T} : i \ \mathbf{pa\not{s}ses}\ t$$

We define the verdict of a test run as the label of the reached terminal state (i.e. **pass** or **fail**). Since an implementation can behave nondeterministically, different test runs of the same test case with the same implementation may lead to different terminal states and hence to different verdicts. An implementation passes a test case if an only if all possible test runs lead to the verdict pass.

Systems are also considered to be fair in their executions. This means that an infinite trace of input actions can not prevent an output or internal action to occur. Intuitively, fair execution means that locally controlled actions cannot be blocked by input actions forever.

2.5.1 Test generation procedure

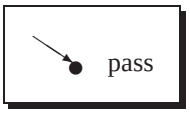
Using the **ioco** theory, the following algorithm for test derivation is presented in [58]. Here, a test case is understood as the specification of the behaviour of a deterministic and finite testing system that can be carried out against an implementation under test. The behaviour of test cases can be described by an LTS, where the occurrence of a δ-action[1] in a test case corresponds to the detection of quiescence in an implementation, i.e. the observation that no output is produced. In practice, the observation of δ is implemented using a time-out of sufficiently long duration. For a formal definition of the time-out, we refer the reader to Chapter 3, where the **ioco** theory is extended with time.

Let $S = \langle Q, q^0, L, T \rangle$ be a specification in LTS with $Q' \subseteq Q$ as a non-empty subset of states, and initially $Q' = \{q^0\}$. With Q' we represent the set of all possible states in which the specification can be at the current stage of the test case execution [58]; i.e. states where the specification could possibly be after the observations made so far. For notational convenience we abbreviate this test generation procedure as TGP.

The algorithm for the generation of test cases $t \in TESTS$ in Q' consists of a finite number of recursive applications of a nondeterministic selection between one of the following three alternatives

[1] In [58] the action symbol θ is used for the observation of quiescence. We prefer to use δ for both quiescent and its observation, in line with the philosophy that identical actions synchronize.

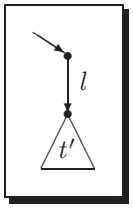
1. *termination*



The single state test case **pass**. It is possible to stop the recursion at any time using this step.

$$t \quad := \quad \mathbf{pass}$$

2. *inputs*

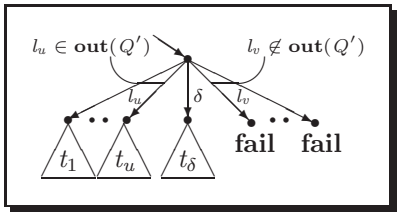


Test case t supplies the input l and behaves as test case t':

$$t \quad := \quad l;\ t'$$

where $l \in I, (Q' \textbf{ after } l) \neq \emptyset$, and t' is obtained by recursively applying the algorithm to $(Q' \textbf{ after } l)$.

3. *outputs*



Test case t checks the next output from the implementation; if it is a valid response the test case continues recursively; if it is an invalid response, i.e. $l \notin \mathbf{out}(Q')$, then the test case terminates in **fail**. The observation of quiescent δ is treated separately:

$$\begin{array}{rlll} t := & \sigma\,\{l_u;\ t_u & | & l_u \in O \wedge l_u \in \mathbf{out}(Q')\} \\ + & \sigma\,\{\delta;\ t_\delta & | & \delta \in \mathbf{out}(Q')\} \\ + & \sigma\,\{\delta;\ \mathbf{fail} & | & \delta \notin \mathbf{out}(Q')\} \\ + & \sigma\,\{l_v;\ \mathbf{fail} & | & l_v \in O \wedge l_v \notin \mathbf{out}(Q')\} \end{array}$$

where t_u and t_δ are obtained by recursively applying the algorithm for $(Q' \textbf{ after } l_u)$ and $(Q' \textbf{ after } \delta)$, respectively.

Example 2.5.3. *Figure 2.4 shows two examples of test cases derived for the cash machine from Figure 2.2. With these tests cases we can test the cash machine from Figure 2.1 (b), which passes both tests.*

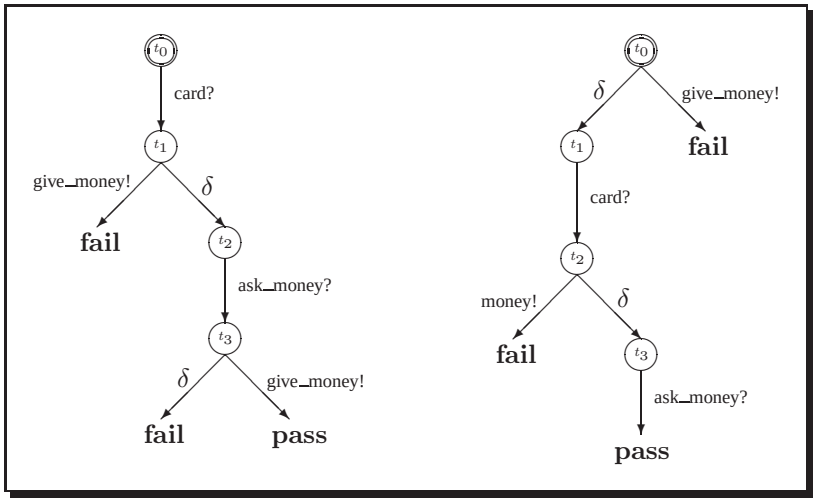


Figure 2.4: Two tests for the cash machine from Figure 2.2

2.6 Completeness

An important property of a test suite is to be sound, i.e. if an implementation fails any test case of the test suite, then it should be the case that there is an error according to the specification. If possible, a test suite also should be exhaustive, i.e. if an implementation has an error the test suite will detect it. Below we define these properties formally for **ioco** and the TGP.

Definition 2.6.1. *Let S be a specification in LTS(I, O) and $\mathbf{T}$ a test suite composed by all test cases obtained from S by the TGP. Then for all i an input-enabled implementation in LTS(I, O)*

$$\begin{array}{lcl} \mathbf{T} \textit{ is sound w.r.t. } \textbf{ioco} & \triangleq & \textit{if } i \textbf{ ioco } S \textit{ then } i \textbf{ passes } \mathbf{T} \\ \mathbf{T} \textit{ is exhaustive w.r.t. } \textbf{ioco} & \triangleq & \textit{if } i \textbf{ passes } \mathbf{T} \textit{ then } i \textbf{ ioco } S \end{array}$$

The test generation algorithm presented is sound in the sense that all test suite cases generated are sound. The test generation algorithm is also exhaustive in the sense that for every incorrect implementation, a test that exposes the error can be generated by the TGP. This important properties are shown in the next two theorems. Here we give a sketch of the proofs for more formal proves we refer the reader to [58].

Theorem 2.6.2. *Let S be a specification in LTS(I, O). Then for all i an input-enabled implementation in LTS(I, O) and all t a test case obtained from S by the TGP*

$$\textit{if } i \textbf{ ioco } S \quad \textit{then} \quad i \textbf{ passes } t$$

Proof.
We give a sketch of the proof. To prove soundness it is sufficient to prove that for all t obtained from S by the TGP

$$\text{if} \quad \forall\ \sigma \in L^*_\delta : t \xrightarrow{\sigma} \textbf{fail} \quad \text{then} \quad \exists\ \sigma' : \exists\ l \in O_\delta : \sigma = \sigma' \cdot l \text{ and } l \notin \textbf{out}(S \textbf{ after } \sigma')$$

This property is proved by contradiction: let t be not sound, then $\exists\ i\ :\ i$ **ioco** S, and $t\ ||\ i \stackrel{\sigma}{\Rightarrow} \textbf{fail}\ ||\ i'$. It follows that $t \xrightarrow{\sigma} \textbf{fail}$ and $i \stackrel{\sigma}{\Rightarrow} i'$, so from the premise: $\exists\ \sigma' : \exists\ l \in O_\delta : \sigma = \sigma' \cdot l$ and $l \notin \textbf{out}(S \textbf{ after } \sigma')$. But since $i \stackrel{\sigma' \cdot l}{\Longrightarrow} i'$ and i **ioco** S, we have $l \in \textbf{out}(S \textbf{ after } \sigma')$, so a contradiction.

By straightforward induction over the structure of t, it is then proved that each t generated with the TGP from S satisfies the previous property.

□

Theorem 2.6.3. *Let S be a specification in LTS(I, O). Then for all i an input-enabled implementation in LTS(I, O) with i **ioco** S, there exists t a test case generated from S by the TGP such that*

$$i \quad \textbf{passes} \quad t$$

Proof.
We give a sketch of the proof. To prove exhaustiveness we have to show that the set of all test cases $\mathbf{T}$ generated with the TGP satisfies

$$\forall\ i : \quad i \textbf{ ioco } S \quad \text{implies} \quad \exists\ t \in \mathbf{T} : t \textbf{ fails } i$$

So let σ be a trace such that $\textbf{out}(i \textbf{ after } \sigma) \not\subseteq \textbf{out}(S \textbf{ after } \sigma)$, so $\exists\ l \in \textbf{out}(i \textbf{ after } \sigma)$ with $l \notin \textbf{out}(S \textbf{ after } \sigma)$. A test case $t_{[\sigma]}$ can be constructed from the TGP as follows:

- $t_{[\epsilon]}$ is obtained with the third choice, followed by the first choice for each t_b with $b \in O$
- $t_{[a\sigma]}$ ($a \in I$) is obtained with the second choice, choosing $l = a$, and followed by recursive applications to obtain $t' = t_{[\sigma]}$
- $t_{[b\sigma]}$ ($b \in O_\delta$) is obtained with the third choice, followed by the first choice for each t'_b with $b' \in O$ and $b' \neq b$, and recursive application to obtain $t_b = t_{[\sigma]}$

Now it can be shown that $t_{[\sigma]}\ ||\ i \stackrel{\sigma}{\Rightarrow} t_{[\epsilon]}\ ||\ i' \stackrel{l}{\Rightarrow} \textbf{fail}\ ||\ i''$, so i **fails** $t_{[\sigma]}$.

□

Example 2.6.4. *We can see that the test cases from Figure 2.4 are sound with respect to the cash machine specification from Figure 2.2. However, considering the test suite composed by these two tests, this test suite is not exhaustive. Because a machine with gives money after the trace σ = card·ask_money·give_money·card, is a wrong implementation, but the tests form Figure 2.4 will not expose the error.*

We conclude this section remarking that the **ioco** testing theory has been practically applied. Different test tools have been built which implement, more or less strictly, the algorithm presented here (i.e. TVEDA [53, 19], TGV [26], TestComposer [37], TestGen [29], TORX [10]). Indeed, the developed theory plus its TORX support tool have proven to be quite a useful and successful approach to the functional testing of reactive systems [60].

2.7 Conclusion

We choose to build our testing framework on **ioco** for the following reasons. The **ioco** testing relation has several desirable properties that make it an attractive testing relation, suitable for building extensions on it. Hence, we choose this theory as our starting point. These desirable properties range from theoretical to practical, and clearly establish the success of the **ioco** as a testing theory and relation; we now highlight some of them:

- **ioco** *has a well defined formal testing theory* We present (a significant part of) this formal testing theory on this chapter.
- **ioco** *works cleanly with reactive systems* Reactive systems are systems that interact with the environment operating through stimuli and returning reactions. Typically, black boxes approaches to testing are done through reactive systems in which the set of possible actions to be performed are divided into inputs and outputs.
- **ioco** *allows non-determinism* Although it is a good engineering practice to refrain from the introduction of unnecessary non-determinism, in the context of black box testing non-determinism is often unavoidable, and hence it must be part of a sensible testing theory. Some further reasons are:
  - Although the implementation under test may be deterministic, often it can only be tested through a testing environment. Environments like operating system features or communication media typically introduce non-determinism into the observed behaviour.
  - An implementation under test often consists of concurrent components in asynchronous parallel composition. The lack of information about the relative progress of components results in nondeterministic properties of their integrated behaviour.
  - Non-determinism allows implementation freedom. Having nondeterministic specifications leaves room for alternatives in implementations.
- **ioco** *considers quiescence as observable* Quiescence characterizes systems that do not produce output actions without a prior stimulation arising from an input action. Making the quiescence concept an observable output action allows to distinguish systems that without quiescence would be indistinguishable (i.e. Example 2.4.2).
- **ioco** *has been successfully implemented in tools* The TORX [10] tool and TGV [26] implement **ioco** . These tools have been used extensively; for instance, TORX has been applied successfully in the several industrial cases studies [9, 17, 63]. Similar algorithms have been also implemented in TVEDA [53, 19], TestComposer [37], TestGen [29].

CHAPTER 3

Testing timed labelled input-output transition systems

3.1 Introduction

Most of the typical formal testing approaches (like the one presented in the previous chapter) are limited to qualitative behaviour of systems, and exclude quantitative aspects such as real-time properties. However, the explosive advance of embedded software has caused a growing need to extend existing testing theories to deal with real-time features. To this end, in this chapter we present an extension of Tretmans' **ioco** testing theory [58] considering real-time.

As explained in Chapter 2 the notion of quiescence plays a central role, in the (untimed) test generation framework for labelled transition systems (LTS) on **ioco** testing theory. Quiescence characterizes systems that do not, and never will, produce an output without prior stimulation with an input. By treating quiescence as a special kind of system output, the notion of behavioural traces can be generalized to include quiescent observations.

A big difference between the untimed case and the time case is the recognition of quiescence. Since we focus on black box testing to recognize quiescence in an implementation we have to wait for outputs. But, it is clear that we can not wait forever. Therefore, we make a realistic assumption in the sense that we assume that there exists a maximal duration M for the waiting period. Then, M is a bound that precisely tells the time is needed to recognize that the black box implementation is in a quiescent state.

Treating quiescence as an observable action allows us to formulate an implementation relation that establishes unambiguously whether an implemented behaviour conforms to a given specification model or not. More precisely, the conformance relation demands that after all specified traces, every possible generalized output from the implementation is allowed according to the specification. In other words, every output and quiescence from the implementation has to be correctly predicted by the specification.

In practice, the above implementation criterion means that implementations may be more deterministic than specifications.

Organization of the chapter We develop a testing theory which addresses both nondeterminism and real-time.

Firstly, in Section 3.2, we introduce our real-time model, by defining timed labelled input-output transition systems (TLTS). In Section 3.2.1, we present a subset of TLTSs: generated by timed automata (TA). The semantics of a TA is defined in terms of an

TLTS. Even if the TLTS generated by TA is a strict subset of the TLTS, it allows for an intuitive and well-known way of being illustrated. We use this drawing capability for some of our examples through the rest of the chapter. In addition, this first part is completed with Section 3.3, where we introduce further notations, restrictions and definitions for TLTSs.

Secondly, in Section 3.4, we present three relations over TLTSs. Initially, we consider inclusion of normalized timed traces (a handy simplification on the notation of timed traces). Later, we parameterize this relation with quiescence observations. Finally, we present an extension of the **ioco** relation on real-time reactive systems. Our relation is called **tioco**$_M$, and it is based on an operational interpretation of the quiescence notion. This interpretation gives rise to a family of implementation relations parameterized by observation durations (M) of quiescence.

Thirdly, in Section 3.5, we formalize precisely the recognition of quiescence in a black box implementation. We achieve this by using a bound M that represents the time it takes to infer that the implementation is in a quiescent state.

Fourthly, in Section 3.6, we define a non-deterministic test generation framework, parameterized by the bound M. In Section 3.7 the test generation framework is shown to be complete with respect to the **tioco**$_M$ implementation relation. This means that it is sound (Section 3.7.1) and exhaustive (Section 3.7.2) with respect to the **tioco**$_M$ implementation relation. A test generation framework is sound with respect to the **tioco**$_M$ if for each implementation that fails a generated test, the implementation is non-conforming with respect to **tioco**$_M$. Besides, the test generation framework is exhaustive with respect to the **tioco**$_M$ if for every non-conforming implementation, following the test generation procedure a test case can be generated that detects its non-conformance.

Finally, in Section 3.8 we show a result that relates our proposed timed extension with the non-timed approach presented in Chapter 2. Concurrently to the research carried out in this chapter, several other timed testing approaches have been developed independently. We compare the approaches in Section 3.9. We conclude with some summary and outlook of our approach.

3.2 Timed labelled input-output transition systems

Timed labelled input-output transition system are labelled input-output transition systems (as described in Chapter 2) extended with time. Apart from having the transitions labelled just with actions, we now let transitions also be labelled with time values that model delays. Then now, we distinguish three types of labels: time-passage actions (from a set $\mathcal{D}$), external actions (from a set L) and the special internal action (τ). Every label which is not a time-passage action is thought of as occurring instantaneously, i.e. without consuming time. We model time as the nonnegative reals (thus $\mathcal{D} = \mathbb{R}^{\geq 0}$); no a priori lower bounds are imposed on the delays between actions. We denote by $\mathcal{L} \triangleq L \cup \mathcal{D}$ and for arbitrary $\mathcal{L}' \subseteq \mathcal{L}$, $\mathcal{L}'_\tau$ denotes $\mathcal{L}' \cup \{\tau\}$.

Definition 3.2.1. *A* timed labelled input-output transition system *(TLTS) $\mathcal{A}$ is a 4-tuple $\langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$, where*

- $\mathcal{Q}$ *is a non-empty set of states*
- $q^0 \in \mathcal{Q}$ *is the initial state*
- $\mathcal{L} \triangleq L \cup \mathcal{D}$ *are the external actions* L *and time-passage actions* $\mathcal{D}$*; where* $\mathcal{D} = \mathbb{R}^{\geq 0}$ *and* L *is partitioned into* I input labels *(denoted by* $l?$*) and* O output labels *(denoted by* $l!$*); with* $L = I \cup O$ *and* $I \cap O = \emptyset$
- $\mathcal{T} \subseteq \mathcal{Q} \times \mathcal{L}_\tau \times \mathcal{Q}$ *is the transition relation with the following consistency constraints, where* $d, d_1, d_2 \in \mathcal{D}$*:*

Time Determinism:	$\forall\ q, q', q'' \in \mathcal{Q} : q \xrightarrow{d} q'$ *and* $q \xrightarrow{d} q''$ *then* $q' = q''$
Time Additivity:	$\forall\ q, q'' \in \mathcal{Q} : \forall\ d_1, d_2 \geq 0 : \exists\ q' \in \mathcal{Q} :$ $q \xrightarrow{d_1} q' \xrightarrow{d_2} q''$ *if and only if* $q \xrightarrow{d_1+d_2} q''$
Null Delay:	$\forall\ q, q' \in \mathcal{Q} : q \xrightarrow{0} q'$ *if and only if* $q = q'$

We denote the components of $\mathcal{A}$ *by* $\mathcal{Q}_\mathcal{A}$*,* $q^0_\mathcal{A}$*,* $\mathcal{L}_\mathcal{A}$*, and* $\mathcal{T}_\mathcal{A}$ *(we omit the subscript* $\mathcal{A}$ *if its meaning is clear from the context). The class of all timed labelled input-output transition systems over* $\mathcal{L}$ *is denoted as* $\mathrm{TLTS}(\mathcal{L})$*. When we need to mention explicitly the input and output sets we denote it by* $\mathrm{TLTS}(I, O)$*.*

Example 3.2.2. *Consider the timed cash machine, illustrated in Figure 3.1. The behaviour of this machine is simple. We can initially ask for money. Then, after* 5 *time units, we receive it. Despite its simplicity, this example illustrates the general complexity of a TLTS. Formally, it can be expressed as a TLTS* $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ *where* $\mathcal{Q} = \{q_0\} \cup \{q_1(d) \mid\ d \in \mathcal{D} \wedge d \in [0, 5]\}$*;* $q^0 = q_0$*;* $\mathcal{L} = I \cup O \cup \mathcal{D}$ *with* $I = \{$*ask_money*$\}$ *and* $O = \{$*give_money*$\}$*; and* $(q_0,$ *ask_money*$, q_1(0)), (q_1(0), 5, q_1(5)), (q_1(5),$ *give_money*$, q_0) \in \mathcal{T}$*. Note that because the infinitary nature of time the number of transitions is also infinite.*

Even in Figure 3.1 we show only three states, actually this TLTS has an infinite number of states. For instance, not shown but of course still there is the state $q_1(2)$ *such that* $q_1(0) \xrightarrow{2} q_1(2)$ *and* $q_1(2) \xrightarrow{3} q_1(5)$*, dictated by the time density together with the time additivity property.*

Unfortunately, a more precise illustration has only been solved for TA, but not yet for TLTS.

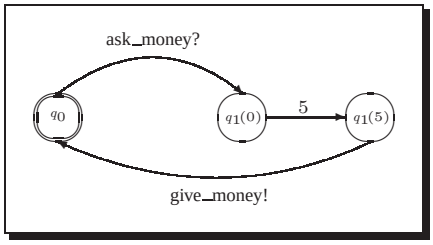


Figure 3.1: A simple cash machine with time as a TLTS

Labels in $\mathcal{L}$ represent the observable actions of a system, i.e. external actions and passage of time; the special label τ represents an unobservable internal action. The set of all

finite sequences of actions over $\mathcal{L}$ is denoted by $\mathcal{L}^*$, while ϵ denotes the empty sequence. If $\sigma_1, \sigma_2 \in \mathcal{L}^*$ then $\sigma_1 \cdot \sigma_2$ is the concatenation of σ_1 and σ_2. A transition $(q, l, q') \in \mathcal{T}$ is denoted as $q \xrightarrow{l} q'$. A path π is a finite, or infinite, sequence of transitions

$$\pi = q_0 \xrightarrow{l_1} q_1 \xrightarrow{l_2} q_2 \xrightarrow{l_3} \cdots \xrightarrow{l_{n-1}} q_{n-1} \xrightarrow{l_n} q_n (\rightarrow \dots)$$

with $q_z \xrightarrow{l_{z+1}} q_{z+1}$ and $z = 0, 1, \dots, n, \dots$. Moreover, we denote with $\mathsf{paths}(\mathcal{A})$ the set of all paths in $\mathcal{A}$, and with $\mathsf{paths}^{\omega}(\mathcal{A})$ the set of all infinite paths in $\mathcal{A}$.

3.2.1 Timed automata

Since TLTSs are quite an expressive formalism, we choose them as the starting point for our testing theory. However, as already mentioned, their infinite set of states makes their illustration difficult (c.f., Example 3.2.2). This problem has been resolved for the subset of TLTSs generated by timed automata. In this section we present the formalism for *timed automata* (or TA) and its well known graphical notation, which is the one we use in future examples. However, note that we use TA indeed only for illustrative purposes. Our testing theory is defined on the far more expressive class of TLTS. As expected, given that our theory relies on TLTS, TA constitute one instance in which our results apply.

Timed automata extend finite LTS with a finite set of clocks over a dense time domain [7]. All clocks increase monotonically with constant rate 1, and measure the amount of time that has elapsed since they started or were reset. The choice of the next state of a timed automaton depends on the action and its occurrence time relative to the occurrence's previous action. Each transition of the system can reset some of the clocks, and has an associated enabling condition. Condition are constraints on the clock values. A transition can be taken only if the current clock values satisfy its enabling condition. Timing constraints on clocks are expressed as in the following definition.

Definition 3.2.3. *For a set C of clock variables, the set $\Phi(C)$ of* clock constraints *φ, where $c \in C$ and $\kappa \in \mathbb{Q}^{\geq 0}$, is defined inductively by*

$$\varphi \quad \triangleq \quad c < \kappa \mid c > \kappa \mid c \leq \kappa \mid c \geq \kappa \mid \varphi_1 \wedge \varphi_2$$

We abbreviate $c \leq \kappa \wedge c \geq \kappa$ as $c = \kappa$, and $0 \leq c$ as true.

Definition 3.2.4. *A* timed automaton *(TA) $\mathcal{F}$ is a tuple $\langle Q, q^0, L, C, Inv, T \rangle$, where*

- *Q is a finite set of locations*
- *$q^0 \in Q$ is the initial location*
- *L is a finite set of actions*
- *C is a finite set of clocks*
- *$Inv : Q \rightarrow \Phi(C)$ associates a clock invariant to each location*
- *$T \subseteq Q \times L_\tau \times \Phi(C) \times 2^C \times Q$ is the set of switches [5]*

As before, we denote the components of a $\mathcal{F}$ by $Q_\mathcal{F}$, $q^0_\mathcal{F}$, $L_\mathcal{F}$, $C_\mathcal{F}$, $Inv_\mathcal{F}$ and $T_\mathcal{F}$. We explicitly exclude TA with $c < 0$ as invariant in its initial state.

A switch $(q, l, \varphi, \lambda, q') \in T$ represents a change of location from $q \in Q$ to $q' \in Q$ on action $l \in L_\tau$. The clock constraint (or guard) $\varphi \in \Phi$ specifies when the switch is enabled, and the set $\lambda \subseteq C$ gives the subset of clocks to be reset when the switch is taken. Clock invariants determine how long the automaton is allowed to stay in a certain location.

In the original theory of timed automata [6, 7], a timed automaton is a finite-state Büchi automaton extended with a set of real-valued variables modeling clocks. Constraints on the clock variables are used to restrict the behaviour of an automaton. A guard on a switch is only an enabling condition, it can not force the switch to be taken. Because of that, Büchi accepting conditions are used to enforce progress properties. A simplified version, namely *timed safety automata* is introduced in [31] to specify progress properties using local invariant conditions. An automaton may remain in a location as long as the clocks values satisfy the invariant condition of the location. Due to their simplicity, timed safety automata have been adopted in several verification tools for timed automata e.g. UPPAAL [41] and Kronos [21]. In this section, we focus on timed safety automata, and following the literature, refer to them as timed automata.

Example 3.2.5. *Consider the TA in Figure 3.2, which is also a timed cash machine. This cash machine can be specified by the following timed automaton* $\mathcal{F} = \langle Q, q^0, L, C, Inv, T \rangle$*, where* $Q = \{q_0, q_1\}$*;* $q^0 = q_0$*;* $L = \{$*ask_money, give_money*$\}$*;* $C = \{c\}$*;* $Inv(q_0) =$ *true*, $Inv(q_1) = c \leq 5$*;* $T = \{(q_0,$ *ask_money*, *true*, $\{c\}, q_1), (q_1,$ *ask_money*, $c < 5, \{c\}, q_1), (q_1,$ *give_money*, $c = 5, \emptyset, q_0)\}$.

We may ask for money at any moment, since the guard in the outgoing switch from q_0 *is true. (Usually, when a guard is true we omit it.) As soon as the machine receives the request for money it resets the clock* c *(denoted as* $\{c\}$ *or* $c := 0$*) and enters location* q_1*. At this location, the machine can either receive another request, and in that case resets the clock again, or in five time units it produces money. The interpretation of the outgoing labelled switch from* q_1 *to* q_0 *is the following. The machine gives money at the precise moment that the clock* c *reaches the value* 5*, and does not reset any clock. Note that the invariant* $c \leq 5$ *in location* q_1*, ensures progress because it forces to take the outgoing switch.*

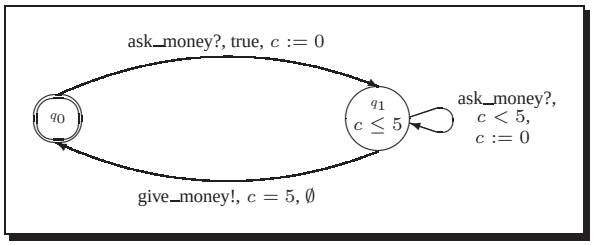


Figure 3.2: A timed automata specification

The behaviour of a timed automaton $\mathcal{F}$ depends on both its current location and the actual values of all its clocks.

Definition 3.2.6. *A* clock valuation *over a set of clocks C is a map $\nu : C \to \mathcal{D}$ that assigns to each clock $c \in C$ a value in $\mathcal{D}$. With $V(C)$ we denote the set of clock valuations over C. For $d \in \mathcal{D}$, $\nu + d$ denotes the clock interpretation which maps every clock c to the value $\nu(c) + d$. When $\lambda \subseteq C$, $\nu[\lambda := 0]$ denotes the clock interpretation for C which assigns 0 to each $c \in \lambda$, and agrees with ν over the rest of clocks.*

TLTSs are used to define the behaviour of timed automata. A state in the TLTS is a pair $\langle q, \nu \rangle$ such that q is a location of $\mathcal{F}$ and ν is a clock valuation for C satisfying the invariant $Inv_{\mathcal{F}}(q)$. Transitions in the TLTS represent either an elapse of time or a switch of $\mathcal{F}$.

Definition 3.2.7. *The* semantics of a timed automaton $\mathcal{F} = \langle Q_{\mathcal{F}}, q^0_{\mathcal{F}}, L_{\mathcal{F}}, C_{\mathcal{F}}, Inv_{\mathcal{F}}, T_{\mathcal{F}} \rangle$ *is given by the TLTS $\mathcal{A} = \langle \mathcal{Q}_{\mathcal{A}}, q^0_{\mathcal{A}}, \mathcal{L}_{\mathcal{A}}, \mathcal{T}_{\mathcal{A}} \rangle$, where*

- $\mathcal{Q}_{\mathcal{A}} = \{\langle q, \nu \rangle \in Q_{\mathcal{F}} \times V(C_{\mathcal{F}}) \mid \nu \models Inv_{\mathcal{F}}(q)\}$
- $\langle q, \nu \rangle = q^0$ *where* $q = q^0_{\mathcal{F}}$ *and* $\nu(c) = 0$ *for all clocks* $c \in C_{\mathcal{F}}$
- $\mathcal{L}_{\mathcal{A}} = L_{\mathcal{F}} \cup \mathcal{D}$
- $\mathcal{T}_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \mathcal{L}_{\mathcal{A}} \times Q_{\mathcal{A}}$, *where*
  - $(\langle q, \nu \rangle, d, \langle q, \nu + d \rangle)$ *if* $d \in \mathcal{D} \wedge \forall\ d' : 0 \leq d' \leq d, \nu + d' \models Inv_{\mathcal{F}}(q)$
  - $(\langle q, \nu \rangle, l, \langle q', \nu[\lambda := 0] \rangle)$ *if* $(q, l, \varphi, \lambda, q') \in T_{\mathcal{F}}$ *and* $\nu \models \varphi$

A significant advantage of timed automata is that it is possible to construct a quotient called the *region automaton*. This construction makes the uncountable state space of a TLTS associated to a TA to be partitioned into finitely many regions [5].

Definition 3.2.8. *A timed automaton $\mathcal{F} = \langle Q, q^0, L, C, Inv, T \rangle$ is called* deterministic *if*

- *$\mathcal{F}$ does not have τ switches*
- *$\forall\ q \in Q : l \in L : \langle q, l, \varphi_1, \lambda_1, q_1 \rangle, \langle q, l, \varphi_2, \lambda_2, q_2 \rangle \in T : \varphi_1 \wedge \varphi_2$ is unsatisfiable*

Deterministic timed automata constitute an important subclass of TA that are strictly less expressive than non-deterministic timed automata [7]. For TAs to be deterministic, multiple transitions starting at the same location with the same label are only permitted when their clock constraints are mutually exclusive. Thus, at most one of the transitions with the same action is enabled at a given time.

To show the expressiveness of TLTSs compared with TAs, as follows we present two examples of systems that can be expressed as a TLTS but not as a TA.

Example 3.2.9. *As an intriguing but very interesting example we can use the uncountable Cantor set*[1] *to specify a cash machine.*

[1]The Cantor set, introduced by German mathematician Georg Cantor, is a construction of a set of points lying on a single line segment, and involving only the real numbers between zero and one. The Cantor set is created by repeatedly deleting the open middle thirds of a set of line segments. One starts by deleting the open middle third $(\frac{1}{3}, \frac{2}{3})$ from the interval $[0, 1]$, leaving two line segments: $[0, \frac{1}{3}]$ and $[\frac{2}{3}, 1]$. Next, the open middle third of each of these remaining segments is deleted. This process is continued ad infinitum. The Cantor set contains all points in the interval $[0, 1]$ that are not deleted at any step in this infinite process. A very important property of the Cantor set is that it is uncountable.

Consider a cash machine where, when we ask for money at times that belong to the Cantor set, the machine gives us money; if the asking time is not in the Cantor set, the machine does not give us money. Interestingly, although this machine can be expressed as a TLTS, it cannot be expressed as a TA. This follows from the fact that we need either an infinite number of locations or to have guards that are able to express special sets as the Cantor set.

It may be argued that this kind of examples are unrealistic, as no one would specify such a system. However, in the following we present a more realistic example in which the expressiveness of TLTSs is still necessary.

Example 3.2.10. *Consider a cash machine that has two clocks; the first one is initialized when we insert the card; the second one is initialized when we introduce the Pin number. In this machine, the order in which these two actions are performed is irrelevant, as they are completely independent. However, the machine gives us money only in the case that these two actions are carried out together closely in time. In Figure 3.3 this machine is shown in TA-like notation; however, it should be noted that this is cash machine is indeed not a TA, since the plus operation does not fall in the format allowed by Definition 3.2.3.*

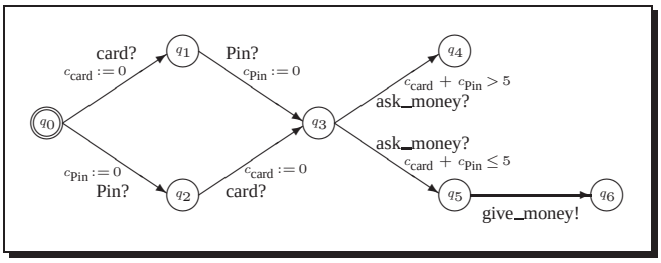


Figure 3.3: A cash machine in TA-like notation

This kind of system requires to control whether the sum of the two clock values is equal or less than a certain constant (5*, in Figure 3.3). Crucially, these kind of guards are not allowed by TAs, and moreover they can not even be expressed in TAs. On the other hand, this poses no difficulty to be expressed as a TLTS.*

Example 3.2.9 is not expressible as a TA because it requires an infinite number of locations and TAs have only a finite number of locations available. On the other hand, Example 3.2.10 requires strictly more expressiveness on TAs guards.

As we already mentioned, the remainder of this chapter is devoted to TLTS, and we only refer to TAs when its restrictions are useful or necessary.

3.3 Definitions, restrictions and notations

In the following we present some simplified notations for transitions in TLTSs.

Definition 3.3.1. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS with $q, q', q_k \in \mathcal{Q}$; $d, d', e \in \mathcal{D}$; $l_k \in \mathcal{L}_\tau$; $\beta \in L$; $\alpha_k \in \mathcal{L}$; $\sigma \in \mathcal{L}^*$, then*

$$
\begin{array}{lcl}
q \xrightarrow{l_1 \dots l_n} q' & \triangleq & \exists\ q_0, \dots, q_n : q = q_0 \xrightarrow{l_1} q_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} q_n = q' \\
q \xrightarrow{l_1 \dots l_n} & \triangleq & \exists\ q' : q \xrightarrow{l_1 \dots l_n} q' \\
q \not\xrightarrow{l_1 \dots l_n} & \triangleq & \nexists\ q' : q \xrightarrow{l_1 \dots l_n} q' \\
q \overset{\epsilon}{\Rightarrow} q' & \triangleq & q = q' \text{ or } q \xrightarrow{\tau \dots \tau} q' \\
q \overset{\beta}{\Rightarrow} q' & \triangleq & \exists\ q_1, q_2 : q \overset{\epsilon}{\Rightarrow} q_1 \xrightarrow{\beta} q_2 \overset{\epsilon}{\Rightarrow} q' \\
q \overset{e}{\Rightarrow} q' & \triangleq & (\exists\ q_1, q_2 : q \overset{\epsilon}{\Rightarrow} q_1 \xrightarrow{e} q_2 \overset{\epsilon}{\Rightarrow} q') \text{ or} \\
 & & (\exists\ q_1, d, d' : d + d' = e : q \overset{d}{\Rightarrow} q_1 \overset{d'}{\Rightarrow} q') \\
q \overset{\alpha_1 \dots \alpha_n}{\Longrightarrow} q' & \triangleq & \exists\ q_0 \dots q_n : q = q_0 \overset{\alpha_1}{\Rightarrow} q_1 \overset{\alpha_2}{\Rightarrow} \cdots \overset{\alpha_n}{\Rightarrow} q_n = q' \\
q \overset{\sigma}{\Rightarrow} & \triangleq & \exists\ q' : q \overset{\sigma}{\Rightarrow} q' \\
q \overset{\sigma}{\not\Rightarrow} & \triangleq & \nexists\ q' : q \overset{\sigma}{\Rightarrow} q'
\end{array}
$$

Definition 3.3.2. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS with $q \in \mathcal{Q}$, $\mathcal{Q}' \subseteq \mathcal{Q}$ and $\sigma \in \mathcal{L}^*$, then*

$$
\begin{array}{lcl}
\mathsf{ttraces}(\mathcal{A}) & \triangleq & \{\sigma \in \mathcal{L}^* \mid q^0 \overset{\sigma}{\Rightarrow} \} \\
\mathsf{init}(q) & \triangleq & \{l \in \mathcal{L}_\tau \mid q \xrightarrow{l} \} \\
\mathsf{der}(q) & \triangleq & \{q' \mid \exists\ \sigma \in \mathcal{L}^* : q \overset{\sigma}{\Rightarrow} q'\} \\
q\ \mathbf{after}\ \sigma & \triangleq & \{q' \mid q \overset{\sigma}{\Rightarrow} q'\} \\
\mathcal{Q}'\ \mathbf{after}\ \sigma & \triangleq & \bigcup_{q \in \mathcal{Q}'} (q\ \mathbf{after}\ \sigma)
\end{array}
$$

$\mathcal{A}$ *is* deterministic *if* $\forall\ \sigma \in \mathcal{L}^* : (q^0\ \mathbf{after}\ \sigma)$ *has at most one element. If $\mathcal{A}$ is deterministic and $(q^0\ \mathbf{after}\ \sigma) \neq \emptyset$, then $(q^0\ \mathbf{after}\ \sigma)$ is overloaded to denote the unique element in $(q^0\ \mathbf{after}\ \sigma)$*

Timed traces, denoted as ttraces, capture the observable behaviour in a TLTS, they are the sequence of observable actions of a path. For a given ttrace σ we denote its length with $|\sigma|$. The set $\mathsf{init}(q)$ contains all labels of outgoing transitions from q. The set $\mathsf{der}(q)$ contains all reachable states from q. In **after** we collect states that are reached from a given state (or a set of states) through a fixed ttrace. As opposed to the untimed case (described in Definition 2.2.5, page 14), all these sets can be infinite, because of the continuous time domain.

Example 3.3.3. *The above definitions, applied to system $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ from Figure 3.1 give us:* $\mathsf{init}(q_0) = \{$*ask_money*$\}$, $\mathsf{der}(q_0) = \mathcal{Q}$ *and* $(q_0\ \mathbf{after}$ *ask_money*$) = \{q_1(0)\}$. *Moreover,* $\sigma =$ *ask_money*$\cdot 5 \cdot$*give_money*$\cdot$*ask_money*$\cdot 3$ *is a timed trace of $\mathcal{A}$, thus* $\sigma \in \mathsf{ttraces}(\mathcal{A})$.

From Definition 3.3.2 of ttraces we can observe that the time actions that appear on a ttrace do not represent absolute time, but stand for relative time with respect to the time of the previous action taken. For instance, in Example 3.3.3, the give_money action happened

5 time units after the ask_money action, and not 5 time units since the initialization of the machine.

3.3.1 Restrictions

In the context of timed systems, particularly TLTSs, there exist three important properties that are necessary, either for characterizing realistic systems or for carrying out proofs in a formal and precise manner.

A TLTS is *time divergent* if it can never reach a time deadlock, i.e. reach a state in which the system does not allow time to pass. A non-time divergent system does not faithfully model the flow of time and thus does not model a system that could exist.

A TLTS is *strongly convergent* if it does not have infinite τ-paths.

The last property, called *no forced inputs*, is specific to timed input-output systems. It says that a system should always have another possibility than blocking if the environment does not provide a particular input action; then if the environment does not provide any input action, the system has the possibility to let time pass. In this way, we exclude the possibility of blocking the time flow when the environment does not provide certain input actions.

Definition 3.3.4. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS, then*

- *$\mathcal{A}$ is* **time divergent***: if for each state q there exists an infinite computation σ from q with infinite cumulative delay, i.e.*

$$\forall\ q \in Q : \exists\ \sigma \in \mathcal{L}_\tau^\omega : \sigma = l_1 \cdot l_2 \cdot l_3 \cdots : q \xrightarrow{\sigma} \wedge \sum_{l_z \in \mathcal{D}} l_z = \infty$$

- *$\mathcal{A}$ is* **strongly convergent***: if there are no infinite τ paths, i.e.*

$$\forall\ \pi \in \mathsf{paths}^\omega(\mathcal{A}) : \exists\ q_z, l \in \pi : q_z \xrightarrow{l} \wedge\ l \neq \tau$$

- *$\mathcal{A}$ has* **no forced inputs***: if for each state q there exists an infinite computation from q without input actions and with infinite cumulative delay, i.e.*

$$\forall\ q \in Q : \exists\ \sigma \in (O_\tau + \mathcal{D})^\omega : \sigma = l_1 \cdot l_2 \cdots : q \xrightarrow{\sigma} \wedge \sum_{l_z \in \mathcal{D}} l_z = \infty$$

From now on, we assume that for all considered $\mathcal{A} \in$ TLTS, $\mathcal{A}$ is time divergent, strongly convergent and has no forced inputs.

A related property to the above ones is the Zeno behaviour property. Briefly, a *Zeno behaviour* is a behaviour that performs an infinite number of actions in finite time. A system is considered to be Zeno when all prefixes of a given Zeno behaviour can not be extended in order to be time divergent. In our case, by simply requiring the time divergent property we are clearly ruling out Zeno systems

Example 3.3.5. *Using TA notation, in Figure 3.4 we present a non-deterministic cash machine. Even though it is a TA, there are no clocks or guards specified, so we can spend as much time as we want in each location. In this machine we have to insert a card, and then we can ask either for 2 or 7 Euros. Internally, the machine can non-deterministically arrive to one of two locations: q_1 and q_2. If the machine is in location q_1 and we ask for 2 Euros, we receive 2 Euros. However, if we ask for 7 Euros the machine does not give it to us. Luckily,*

we can insist and ask again; this time, we will receive the 7 *Euros. In location* q_2 *we observe the opposite behaviour; we can ask and receive* 7 *Euros but we need to insist to receive the* 2 *Euros.*

This cash machine is time divergent because we can spend infinite time in each location. Clearly, this machine is strongly convergent because does not have τ *transitions. Moreover, it does no forced inputs because it does not have state invariants.*

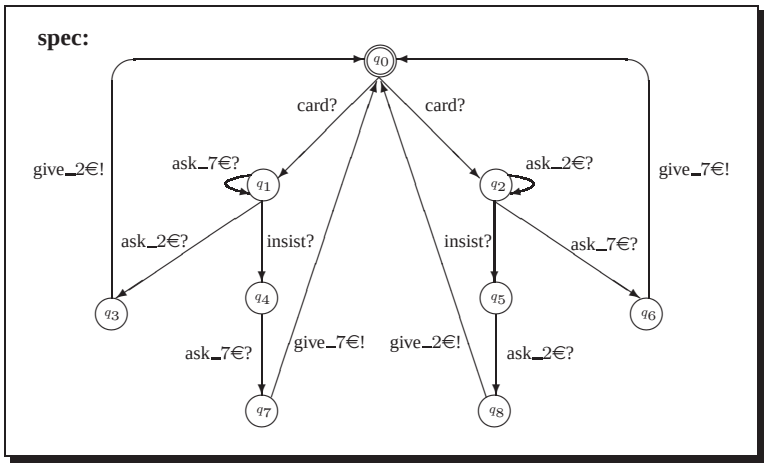


Figure 3.4: A non-deterministic cash machine

3.3.2 Normalized timed traces

In this section we present a useful simplification of the notion of ttraces (Definition 3.3.2).

A ttrace σ is a sequence of actions and delays, e.g. $\sigma = a?\cdot d_1 \cdot d_2 \cdot b!$. Obviously, it would be more natural to avoid consecutive delays, as in $\sigma = a?\cdot d_1 + d_2 \cdot b!$. Such ttraces could alternatively be written as sequences of actions with (relative) time stamps attached to the actions, viz. $\sigma = a?(0)\cdot b!(d_1 + d_2)$, meaning that action $a?$ occurs at time 0 and action $b!$ occurs $d_1 + d_2$ time units after $a?$. This idea motivates the definition of *normalized timed traces*, as follows.

Definition 3.3.6. *A* normalized timed trace σ *is an element of* $(\mathcal{D} \cdot L)^* \cdot (\epsilon + \mathcal{D})$.

Then normalized timed traces are a subset of ttraces which are of the special form $(\mathcal{D} \cdot L)^* \cdot (\epsilon + \mathcal{D})$, they have actions appearing after a time (that could be 0) and there are no consecutive delays. Thus, actions and time alternate. In case that σ is a normalized timed trace, $\sigma = d_0 \cdot l_0 \cdot d_1 \cdot l_1 \cdots d_n \cdot l_n \cdot d_{n+1}$, we write $\sigma = l_0(d_0)\cdot l_1(d_1) \cdots l_n(d_n)\cdot d_{n+1}$.

Definition 3.3.7. *Let* $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ *be a TLTS*(I, O) *then we define the set of* normalized timed traces *of* $\mathcal{A}$*, denoted* nttraces$(\mathcal{A})$*, as*

$$\mathsf{nttraces}(\mathcal{A}) \quad \triangleq \quad \{\sigma \in (\mathcal{D} \cdot L)^* \cdot (\epsilon + \mathcal{D}) \mid q^0 \stackrel{\sigma}{\Rightarrow}\}$$

It is possible to associate to each ttrace a normalized one, given a ttrace σ we denote its associated normalized timed trace as $\hat{\sigma}$. Note that this conversion can always be done combining delays and adding 0 delays if necessary.

We prove that for a TLTS(I, O) system the set of nttraces characterize the set of all ttraces.

Theorem 3.3.8. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS(I, O), then for all $\sigma \in \mathcal{L}^*$*

$$\sigma \in \mathsf{ttraces}(\mathcal{A}) \quad \textit{if and only if} \quad \hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$$

Proof.

[$\Leftarrow$ If $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$ then $\sigma \in \mathsf{ttraces}(\mathcal{A})$]
Because $\hat{\sigma}$ is in $\mathsf{nttraces}(\mathcal{A})$ using Definition 3.3.7 it is easy to check that also σ is in $\mathsf{ttraces}(\mathcal{A})$.

[$\Rightarrow$ If $\sigma \in \mathsf{ttraces}(\mathcal{A})$ then $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$]
By induction over the length ($|\sigma|$) of σ:
Let $|\sigma| = 1$ (the case $\sigma = \epsilon$ is trivial), then
if $\sigma = d$ then $\sigma = \hat{\sigma}$ and $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$;
if $\sigma \neq d$ then $\sigma = l$. Then $\hat{\sigma} = 0 \cdot l$ or $\hat{\sigma} = l(0)$ and $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$.
Suppose that for all σ with $|\sigma| < n$ there exists $\hat{\sigma}$ such that $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$.
Let $|\sigma| = n$ then $\sigma = \sigma' \cdot \mu$ with $|\sigma'| < n$ and $\mu \in (\mathcal{D} + L)$. By inductive hypothesis there exists $\widehat{\sigma}' \in \mathsf{nttraces}(\mathcal{A})$ such that $\widehat{\sigma}' = d_0 \cdot l_0 \cdots d_{k-1} \cdot l_{k-1} \cdot d_k$,
then if $\mu \in \mathcal{D}$ let $\mu = d_{k+1}$ then $\hat{\sigma} = d_0 \cdot l_0 \cdots d_{k-1} \cdot l_{k-1} \cdot (d_k + d_{k+1})$ or
$\hat{\sigma} = l_0(d_0) \cdots l_{k-1}(d_{k-1}) \cdot (d_k + d_{k+1})$;
if $\mu \in L$ let $\mu = l_k$ then $\hat{\sigma} = d_0 \cdot l_0 \cdots d_{k-1} \cdot l_{k-1} \cdot d_k \cdot l_k$ or
$\hat{\sigma} = l_0(d_0) \cdots l_{k-1}(d_{k-1}) \cdots l_k(d_k)$
thus $\hat{\sigma} \in \mathsf{nttraces}(\mathcal{A})$.

□

Example 3.3.9. *As an example, we can convert the ttrace $\sigma = 2 \cdot$card$\cdot 3 \cdot 1 \cdot$ask_2€$\cdot$give_2€$\cdot 5$, from Figure 3.4, in the normalized timed trace $\hat{\sigma} =$ card$(2) \cdot$ask_2€$(4) \cdot$give_2€$(0) \cdot 5$.*

Using Theorem 3.3.8 from now on, we do not distinguish between a ttrace σ and its normalized version $\widehat{\sigma}$. Moreover we lift all defined notations (i.e. $\stackrel{\sigma}{\rightarrow}$, $\stackrel{\sigma}{\Rightarrow}$, etc.) that have been defined for ttraces to nttraces.

3.3.3 Input-enabled timed labelled input-output transition systems

To formalize the notion of input-enabledness in timed systems we ask that if an input action is initiated by the environment, the system is always prepared to participate in such an inter-

action; inputs are accepted without letting time pass. Then, a TLTS is input-enabled if it is ready to accept any input at all times.

Definition 3.3.10. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS(I, O), then $\mathcal{A}$ is* weak input-enabled *if all input actions are accepted in all states, abstracting from τ transitions*

$$\forall\ q \in Q : \forall\ l \in I : q \stackrel{l}{\Rightarrow}$$

Our definition of input-enabled involves a double arrow, since it abstract from τ-transitions. However, there does exists a definition with a simple arrow ($\forall\ q \in Q : \forall\ l \in I \wedge l \neq \tau : q \stackrel{l}{\rightarrow}$) which is known as strong input-enabled. To avoid confusion, when we refer to the definition with a simple arrow we explicitly use the word strong. In other case when we use input-enabled we always mean weak input-enabled.

In the following we assume that implementations can be modelled as weak input-enabled TLTS. Later on, in Chapter 4, we relax this assumption.

Example 3.3.11. *We can observe that the cash machine from Figure 3.4, location q_0 does not allow ask_2€ as a label of any outgoing transition nor ask_7€. Hence, we conclude that this system is not input-enabled.*

Later on in Example 3.5.6, we present a modified version of this machine which is input-enabled, and can be used as an implementation.

3.3.4 Quiescence

In the presence of time, a quiescent state is a state where the system is unable to produce an output immediately or in the future, without receiving an input first.

Definition 3.3.12. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS(I, O). A state $q \in \mathcal{Q}$ is* quiescent, *denoted by $\delta(q)$, if*

$$\forall\ l \in O : \forall\ d \in \mathcal{D} : q \stackrel{l(d)}{\not\Rightarrow}$$

Similarly to the untimed case, we represent quiescence as a special action δ ($\delta \notin L$), and extending the timed transition relation of a TLTS $\mathcal{A}$ to include self-loop transitions $q \stackrel{\delta}{\rightarrow} q$ for each quiescent state q

$$q \stackrel{\delta}{\rightarrow} q \quad \text{if and only if} \quad \delta(q)$$

Definition 3.3.13. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS, we write $\Delta(\mathcal{A})$, to denote the same system with its timed transition relation $\mathcal{T}_{\mathcal{A}}$ extended with self-loop transitions labelled with δ in all quiescent states.*

We lift all concepts and notations (i.e. nttraces, init, etc.) that have been defined for TLTSs to extended TLTSs. In this way, ttraces now have the possibility to perform δ actions.

Example 3.3.14. *Figure 3.5 shows a timed version of Example 2.4.2 (from Chapter 2, page 19). Again two different input-enabled non-deterministic systems are represented. On the left hand side we see an implementation that prescribes that after we introduce a card we will receive* 7 *Euros in less than* 5 *time units or not Euros at all, and in case we introduce two cards we can receive* 2 *or* 7 *Euros in less than* 5 *time units. On the right hand side we see a specification, where after the introduction of a card if the machine internally chooses to go to state* q_2 *then it can only produce* 2 *Euros.*

These two machines have the same set of ttraces (without quiescence), but with quiescence it is possible to distinguish them. Because the nttrace $\sigma = card?(2) \cdot \delta(0) \cdot card?(2) \cdot give\_7€!(3)$ *is in the implementation but it is not in the specifications.*

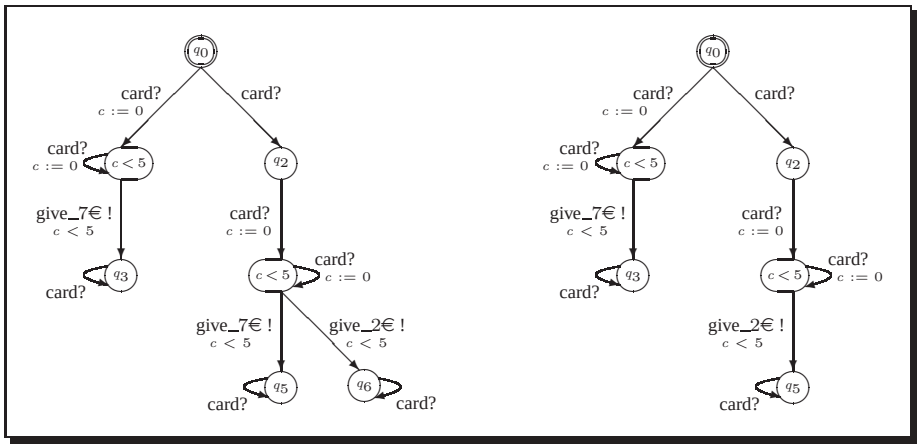


Figure 3.5: Specification of two cash machines

3.3.5 Output set

Given a system in TLTS(I, O) and a state in it, the output set $\mathbf{out}_M(q)$ collects time-stamped output actions that are allowed from state q (possibly after some τ-actions) and the δ-actions that can appear after a delay.

In a specification $S \in$ TLTS, the quiescent states can be identified by analyzing the timed transition system, i.e. we can assume that $\Delta(S)$ is at our disposal. However, given that we work in a black box framework, for an input-enabled implementation $i \in$ TLTS we can only detect quiescence by waiting for outputs. Obviously, we can not wait forever. Therefore, there must exist a maximal duration M, setting the time needed to recognize a quiescent state. We use this bound M in the output set to recognize quiescence. Thus, we are concluding that the system is in a quiescent state, if after M time units, no output has occurred.

Definition 3.3.15. *Let $\Delta(\mathcal{A}) = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS(I, O) with $q \in Q$, $\mathcal{Q}' \subseteq \mathcal{Q}$ and $M \in \mathcal{D}$ be the bound to recognize quiescence, then*

$$\begin{array}{lcl} \mathbf{out}_M(q) & \triangleq & \{b(d) \mid b \in O \wedge d \in \mathcal{D} \wedge q \stackrel{b(d)}{\Rightarrow}\} \bigcup \{\delta(M) \mid q \stackrel{\delta(M)}{\Rightarrow}\} \\ \mathbf{out}_M(\mathcal{Q}') & \triangleq & \bigcup_{q \in \mathcal{Q}'} \mathbf{out}_M(q) \end{array}$$

Therefore, the output set with parameter M collects all the output actions that a system can produce, including observation of quiescence, in a given state or set of states.

Example 3.3.16. *In the system from Figure 3.4, if we fix $M = 6$, we observe that: give_2€(5) $\in \mathbf{out}(q_3)$ and $\delta(6) \in \mathbf{out}(q_1)$.*

3.4 Timed implementation relations

Since we develop the timed testing theory inspired by the untimed case, we define our final testing relation **tioco**$_M$ through two auxiliary relations, namely $\leq_{tiorf}$ (inclusion of nttraces) and $\leq^M_{tiorf}$ (inclusion of nttraces with quiescence appearing only after M time units).

The extension of the timed transition relation over TLTSs allows us to define a relation over input-enabled implementations in TLTS(I, O) and specifications in TLTS(I, O) as inclusion of nttraces.

Definition 3.4.1. *Let i be an input-enabled implementation in TLTS(I, O) and S be a specification in TLTS(I, O), then*

$$i \leq_{tiorf} S \quad \triangleq \quad \mathsf{nttraces}(\Delta(i)) \subseteq \mathsf{nttraces}(\Delta(S))$$

As we mentioned before, in specifications S quiescent states can be identified, then $\Delta(S)$ is at our disposal. However, this is not the case in implementations where we need the bound M to recognize quiescence. Consequently, the following definition characterizes nttraces in which quiescence (δ) can only appear after exactly M time units.

Definition 3.4.2. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS, then*

$$\mathsf{nttraces}^{\Delta}_M(\mathcal{A}) \quad \triangleq \quad \mathsf{nttraces}(\Delta(\mathcal{A})) \cap (\mathcal{D} \cdot L + M \cdot \delta)^* \cdot (\epsilon + \mathcal{D})$$

Thus, $\mathsf{nttraces}^{\Delta}_M$ are those nttraces where quiescence comes after exactly M time units. This definition motivates the following parameterized version of our tiorf-relation, which checks for inclusion on $\mathsf{nttraces}^{\Delta}_M$.

Definition 3.4.3. *Let i be an input-enabled implementation in TLTS(I, O) and S be a specification in TLTS(I, O) with $M \in \mathcal{D}$, then*

$$i \leq^{M}_{tiorf} \; S \quad \triangleq \quad \mathsf{nttraces}^{\Delta}_{M}(i) \subseteq \mathsf{nttraces}^{\Delta}_{M}(S)$$

The $\mathsf{nttraces}^{\Delta}_{M}$ set only takes into account observations of quiescence that are made after a delay of M time units. Hence, we leave out all nttraces with the form: $\sigma \cdot \delta(k) \cdot \sigma'$ were $k \neq M$.

The following proposition states that the previous parametric tiorf relation (parameterized by M) can be characterized by the inclusion of output sets from Definition 3.3.15.

Proposition 3.4.4. *Let i be an input-enabled implementation in TLTS(I, O) and S be a specification in TLTS(I, O), then*

$$i \leq^{M}_{tiorf} S \quad \textit{if and only if} \quad \forall\ \sigma \in (\mathcal{D} \cdot L + M \cdot \delta)^* \cdot (\epsilon + \mathcal{D}) : \mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$$

Proof.

[$\Rightarrow$ If $i \leq^{M}_{tiorf} S$ then $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$]
Let $\sigma \in (\mathcal{D} \cdot L + M \cdot \delta)^* \cdot (\epsilon + \mathcal{D})$, then
if $\sigma \notin \mathsf{nttraces}(\Delta(i))$, then $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) = \emptyset$;
if $\sigma \in \mathsf{nttraces}(\Delta(i))$, then $\forall\ b : b \in \mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma)$ then
$\sigma \cdot b \in \mathsf{nttraces}^{\Delta}_{M}(i)$. Using $i \leq^{M}_{tiorf} S$ we have
$\sigma \cdot b \in \mathsf{nttraces}^{\Delta}_{M}(S)$, then $b \in \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$.

[$\Leftarrow$ If $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$ then $i \leq^{M}_{tiorf} S$]
Let $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(i)$, i.e. $\sigma \in \mathsf{nttraces}(\Delta(i)) \cap (\mathcal{D} \cdot L + M \cdot \delta)^* \cdot (\epsilon + \mathcal{D})$, then $(\Delta(i)\ \mathbf{after}\ \sigma) \neq \emptyset$. Hence, using the no forced inputs property from Definition 3.3.4 we know that $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \neq \emptyset$. Because, $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$, then $\mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma) \neq \emptyset$. So $\sigma \in \mathsf{nttraces}(\Delta(S))$ and $\sigma \in (\mathcal{D} \cdot L + M \cdot \delta)^* \cdot (\epsilon + \mathcal{D})$. Then, we have $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(S)$.

□

3.4.1 The $\mathbf{tioco}_M$ implementation relation

Finally, we are in the position to define the relation we use to test timed labelled input-output transition systems, the $\mathbf{tioco}_M$ relation. For an input-enabled implementation i and a specification S, i is $\mathbf{tioco}_M$ correct w.r.t. S, if the output set of i after every nttrace of S, including observations of quiescence, is a subset of the output set of S after the same nttrace.

Definition 3.4.5. *Let i be an input-enabled implementation in TLTS(I, O) and S be a specification in TLTS(I, O), then*

$$i\ \mathbf{tioco}_M\ S \quad \triangleq \quad \forall\ \sigma \in \mathsf{nttraces}^{\Delta}_{M}(S) \text{: } \mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$$

The $\mathbf{tioco}_M$ relation is a restriction of the tiorf-relation that only takes into account nttraces that belong to the specification. This relation allows us to test black-box implementations assuming that the systems are input-enabled TLTSs, which is our aim. Hence, $\mathbf{tioco}_M$

is our target relation from which we define and derive our tests.

3.5 Operational model

To obtain an effective theory of quiescence in a timed setting, we need more than stipulating that observing quiescence takes time. Since with physical implementations we can only observe the absence of outputs over finite time intervals, we must precisely stipulate when such observations can be interpreted as quiescence.

Definition 3.5.1. *Let $i = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be an input-enabled implementation in TLTS(I, O) and $M \in \mathcal{D}$, then*

$$\begin{array}{lcl} q \in \mathcal{Q} \textit{ is } M\textit{-quiescent} & \triangleq & \forall\ q' \in \mathcal{Q} : q' \in (q\ \mathbf{after}\ M) : q' \textit{ is quiescent} \\ i \textit{ is } M\textit{-quiescent} & \triangleq & \forall\ q \in \mathcal{Q} : q \textit{ is } M\textit{-quiescent} \end{array}$$

The strategy to formalize how nttraces of an input-enabled TLTS may be enriched with δ-actions is as follows. Whenever an nttrace allows an action with a delay of more than M time-units, this creates a possibility to observe quiescence. For example, if a system is M-quiescent and $M = 4$ and the nttrace $\sigma = a?(2) \cdot b?(5) \cdot c!(3)$ is observed, then it is also possible to observe $\sigma' = a?(2) \cdot \delta(4) \cdot b?(1) \cdot c!(3)$. We formalize the addition of δ-observations to nttraces as a relation δ_M between nttraces.

Definition 3.5.2. *Let $i = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a M-quiescent input-enabled implementation in TLTS(I, O). Let $\sigma, \sigma', \sigma_1, \sigma_2 \in (\mathcal{D} \cdot L_\delta)^* \cdot (\epsilon + \mathcal{D})$ be* nttraces*, $l \in L$ and $d \in \mathcal{D}$, then*

- $\sigma\ \delta_M\ \sigma' \quad \triangleq \quad \exists\ \sigma_1, \sigma_2 : \exists\ l : \exists\ d \geq M :$
$$\sigma = \sigma_1 \cdot l(d) \cdot \sigma_2 \wedge \sigma' = \sigma_1 \cdot \delta(M) \cdot l(d - M) \cdot \sigma_2$$

- *let Σ be a set of* nttraces$(\mathcal{A})$*, then*
$$\delta_M(\Sigma) \quad \triangleq \quad \textit{pref}\,(\bigcup_{\sigma \in \Sigma} \{\sigma' \mid \sigma\ \delta_M^*\ \sigma'\})$$
where pref(Σ') is interpreted as the prefix-closure of a set of traces Σ' *and δ_M^* is the reflexive and transitive closure of the relation δ_M.*

In the previous definition the action $l(d)$ can only be an input action. This is because two reasons: firstly the time d is bigger or equal to the bound M which is the bound for quiescence observations and secondly the system i is M-quiescent. An informal interpretation of the δ_M relation is that, given a nttrace σ its collects all ttraces similar to it with more occurrences of $\delta(M)$.

As follows we check the consistency in our proposed δ saturation. If a δ-action is introduced in an nttrace, on the basic observations of a delay of (at least) M time units, we must have the following property: "there never appears an output action immediately after a $\delta(M)$".

Lemma 3.5.3. *Let i be a M-quiescent input-enabled implementation in TLTS(I, O), then* $\forall\ \sigma \in \delta_M(\mathsf{nttraces}(i))$

$$\textit{if} \quad \sigma = \sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2 \quad \textit{then} \quad l \notin O$$

Proof.
By contradiction.

Suppose that there exists a $\sigma \in \delta_M(\mathsf{nttraces}(i))$ with $\sigma = \sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2$ and l is an output action ($l \in O$). Then $\sigma' = \sigma_1 \cdot l(d+M) \cdot \sigma_2$ is also a ttrace of the system i. Hence, there exists a states $q \in (i \ \mathbf{after} \ \sigma_1 \cdot M)$ such that $q \stackrel{l(d)}{\Rightarrow}$. Using definition of output set (Definition 3.3.15) we have that $l(d) \in \mathbf{out}(i \ \mathbf{after} \ \sigma_1 \cdot M)$. Then, state q is not M-quiescent, and i is not M-quiescent, so a contradiction. Then, a $\sigma = \sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2$ with $l \in O$ does not exist.

□

Using the previous corollary, we show that if an input-enabled implementation i can be assumed to be M-quiescent, then we may use the set of enriched observations $\delta_M(\mathsf{nttraces}(i))$ to obtain the $\mathsf{nttraces}^{\Delta}_M(i)$ set, whose definition is based on the unobservable TLTS, $\Delta(i)$. This result is the basis for our test derivation algorithm presented in the next section.

Theorem 3.5.4. *Let i be a M-quiescent input-enabled implementation in TLTS(I, O) with $I \neq \emptyset$, then*

$$\delta_M(\mathsf{nttraces}(i)) \quad = \quad \mathsf{nttraces}^{\Delta}_M(i)$$

Proof.
To prove this result we use that for an nttrace $\sigma \in \mathsf{nttraces}(i)$ the function δ_M only introduces occurrences of $\delta(M)$. Similarly, in $\mathsf{nttraces}^{\Delta}_M(i)$ all occurrences of $\delta(d)$ with $d \neq M$ in $\mathsf{nttraces}$ have been filtered out. Then, we prove this intuitive argument by induction over the number of occurrences of $\delta(M)$ in a nttrace σ (denoted $|\sigma|_\delta$).

Let $|\sigma|_\delta = 0$ then

		$\sigma \in \delta_M(\mathsf{nttraces}(i))$
(By Definition 3.5.2)	if and only if	$\sigma \in \mathsf{nttraces}(i)$
(By Definition 3.4.3 and $\lvert\sigma\rvert_\delta = 0$)	if and only if	$\sigma \in \mathsf{nttraces}^{\Delta}_M(i)$

Suppose that for all σ with $|\sigma|_\delta < n$ we have

$$\sigma \in \delta_M(\mathsf{nttraces}(i)) \quad \text{if and only if} \quad \sigma \in \mathsf{nttraces}^{\Delta}_M(i)$$

Let $|\sigma|_\delta = n$, then there are two cases

- Suppose that σ is of the form $\sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2$, with $|\sigma_1|_\delta = n-1 \wedge |\sigma_2|_\delta = 0$. We prove that such a σ is in $\delta_M(\mathsf{nttraces}(i))$ if and only if σ is in $\mathsf{nttraces}^{\Delta}_M(i)$. First note that:

		$\sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2 \in \delta_M(\mathsf{nttraces}(i))$
(by def of δ_M)	if and only if	$\sigma_1 \cdot l(d+M) \cdot \sigma_2 \in \delta_M(\mathsf{nttraces}(i))$
(Ind. Hyp. on σ_1)	if and only if	$\sigma_1 \cdot l(d+M) \cdot \sigma_2 \in \mathsf{nttraces}^{\Delta}_M(i)$

[$\Leftarrow$] If $\sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2 \in \mathsf{nttraces}^{\Delta}_{M}(i)$ then by skipping δ-observations $\sigma_1 \cdot \delta(M) \cdot l(d) \cdot \sigma_2 \in \delta_M(\mathsf{nttraces}(i))$. Then, clearly we have that $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(i)$ implies $\sigma \in \delta_M(\mathsf{nttraces}(i))$.

[$\Rightarrow$] Let $\sigma \in \delta_M(\mathsf{nttraces}(i))$, then by the above implications $\sigma_1 \cdot \mu(d + M) \cdot \sigma_2 = \sigma$ is in $\mathsf{nttraces}^{\Delta}_{M}(i)$. Now suppose that $\sigma_1 \cdot \delta(M) \cdot \mu(d) \cdot \sigma_2$ is not in $\mathsf{nttraces}^{\Delta}_{M}(i)$. This means that there is a non-quiescent state in $(i \textbf{ after } \sigma_1 \cdot M)$, i.e. there is an output action $\mu \in O$ with $\sigma_1 \cdot \delta(M) \cdot \mu(d') \in \mathsf{nttraces}^{\Delta}_{M}(i)$. Again, by the above implications we have that $\sigma_1 \cdot \delta(M) \cdot \mu(d') \in \delta_M(\mathsf{nttraces}(i))$. Using Lemma 3.5.3, this contradicts that i is M-quiescent.

- If σ is not of the required format, it must be of the form $\sigma_1 \cdot \delta(M)$, and by input-enabledness we can extend it to the form $\sigma_1 \cdot \delta(M) \cdot \mu(0)$ by appending some input action μ. Thus, the above result together with the prefix-closure of $\delta_M(\mathsf{nttraces}(i))$ and $\mathsf{nttraces}^{\Delta}_{M}(i)$, implies that these sets also coincide for such ttraces, i.e. the theorem holds.

□

As a final property, before presenting the derivation algorithm we present Theorem 3.5.5. This theorem reveals that the **tioco**$_M$ relation, in presence of M-quiescent implementation, implies an M based pre-order (is a pre-order with respect to the magnitude of M).

Theorem 3.5.5. *Let i be a M_1-quiescent input-enabled implementation in TLTS(I, O) and S be a specification in TLTS(I, O) with $M_1 < M_2$, then*

$$\textit{if} \quad i \ \textbf{tioco}_{M_1} \ S \quad \textit{then} \quad i \ \textbf{tioco}_{M_2} \ S$$

Proof.
Let assume that i **tioco**$_{M_1}$ S and σ is a nttrace in $\mathsf{nttraces}^{\Delta}_{M_2}(i)$. Then transform σ into σ' by replacing every subsequence of the form $\delta(M_2) \cdot l(d)$ in σ by $\delta(M_1) \cdot l(d + (M_2 - M_1))$, and possibly $\delta(M_2)$ by $\delta(M_1) \cdot (M_2 - M_1)$ in case $\delta(M_2)$ is the last action of σ. Moreover, we saturated the nttrace σ' with $\delta(M_1)$ in case there exists in σ' a time stamp d bigger than M_1, recursively. Then σ' is also an nttrace of $\Delta(i)$, and therefore of $\mathsf{nttraces}^{\Delta}_{M_1}(i)$. As $i \leq^{M_1}_{\text{tiorf}} S$, it follows that σ' is a nttrace in $\mathsf{nttraces}^{\Delta}_{M_1}(S)$. By postponing back all observations of δ by $M_2 - M_1$ we get $\sigma \in \mathsf{nttraces}^{\Delta}_{M_2}(S)$, i.e. $\mathsf{nttraces}^{\Delta}_{M_2}(i) \subseteq \mathsf{nttraces}^{\Delta}_{M_2}(S)$ then i **tioco**$_{M_2}$ S.

This postponing back observation of δ can be done because the original nttrace, σ, was from $\mathsf{nttraces}^{\Delta}_{M_2}(i)$. Then σ has its observation of quiescence done after M_2 time units. Using this information we can be sure that there is enough time to postpone.

□

Example 3.5.6. *Figure 3.6 presents a new cash machine in TA notation. This system is similar to the one from Figure 3.4, but it can be an implementation, because it is input-enabled. Furthermore, it is M-quiescent for $M = 5$. The M-quiescent property is archived by the invariants ($c < 5$) on all states with outgoing output transitions. In this machine we can be sure that if we receive money (give_2 € or give_7 €) then this action happen before*

5 *time units have passed. This is the certainly provided by the* M*-quiescent property.*

To keep the illustration readable we assume that all actions reset the clock c *each time they are taken.*

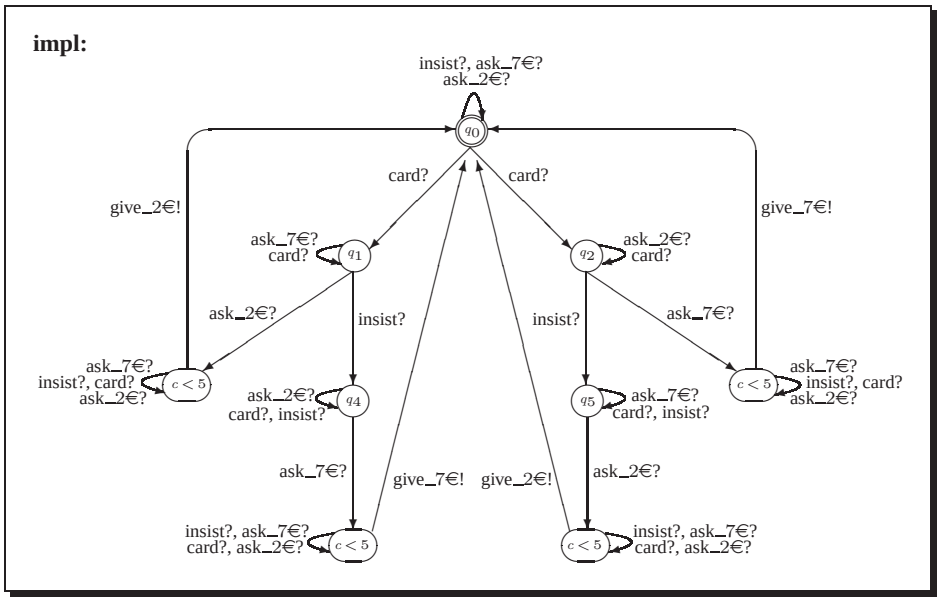


Figure 3.6: A non-deterministic cash machine implementation. We assume that every transition reset the clock c.

3.6 Timed test generation framework

In this section we define the concept of timed test cases, the nature of their execution, and the evaluation of their verdict: pass or failure.

A *timed test case* t is a deterministic TLTS(I, O) with actions in $\mathcal{L}_\delta$ (where $\mathcal{L}_\delta = \mathcal{L} \cup \{\delta\}$) such that it has bounded behaviour, in the sense that all computations have finitely many action occurrences. The set of states in a test contains the terminal states **pass** and **fail** without outgoing transitions, except self loops allowing time to pass. For any state different from **pass** and **fail** there exists a bounded time to observe quiescence or to be able to make an input action. Moreover, because tests under consideration are deterministic τ-transitions are not allowed. A *timed test suite* $\mathbf{T}$ is a set of test cases. To simplify notation we represent tests similarly as TA.

Definition 3.6.1. *Given* $M \in \mathcal{D}$ *a fixed bound to recognize quiescence in* $\mathcal{D}$*, then*

- *A* timed test case $t = \langle \mathcal{Q}, q_0, \mathcal{L}_\delta, \mathcal{T} \rangle$ *is a TLTS such that*
  - t *is deterministic and has* bounded behaviour*, i.e.*

$$\exists\ N > 0 : \forall\ \sigma \in \mathsf{ttraces}(t) : \sigma = l_1 \cdot l_2 \cdot l_3 \cdots : |\{z\ \mid\ l_z \in L\}| < \infty$$

- *$\mathcal{Q}$ contains the terminal states* $\mathbf{pass}$ *and* $\mathbf{fail}$*, with* $\mathsf{init}(\mathbf{pass})$ *and* $\mathsf{init}(\mathbf{fail})$ *without outgoing transitions except self loops allowing time to pass*
- *for any state $q \in \mathcal{Q}$ of the test case, with $q \neq \mathbf{pass}, \mathbf{fail}$ there exists $0 \leq d \leq M$ with*

$$\begin{array}{lcl} q^0(q\ \mathbf{after}\ d') & = & O \cup \{e\ \mid\ e \leq d - d'\} \text{ for all } d' < d, \text{ or} \\ q^0(q\ \mathbf{after}\ d) & = & \{l\} \text{ with } l \in I \text{ or } l = \delta \end{array}$$

- *t does not have τ-transitions*

The class of test cases over I and O is denoted by $\mathcal{TEST}(I, O)$

- *A* timed test suite $\mathbf{T}$ *is a set of test cases:* $\mathbf{T} \subseteq \mathcal{TEST}(I, O)$

For the description of test cases we use, as already done in Chapter 2, a process-algebraic behaviour notation with a syntax inspired by LOTOS [35]

$$B \quad \triangleq \quad l;\ B\ \mid\ B + B\ \mid\ \Sigma\, \mathcal{B}$$

where $l \in \mathcal{L}$, $\mathcal{B}$ is a set of behaviour expressions, where the axioms and the inference rules are:

$$\begin{array}{lcl} l \in L & \vdash & l;\ B \xrightarrow{l} B \\ l = d, d' < d & \vdash & l;\ B \xrightarrow{d'} d - d';\ B \\ l = d & \vdash & l;\ B \xrightarrow{d} B \\ B_1 \xrightarrow{l} B_1', l \in \mathcal{L}_\delta & \vdash & B_1 + B_2 \xrightarrow{l} B_1' \\ B_2 \xrightarrow{l} B_2', l \in \mathcal{L}_\delta & \vdash & B_1 + B_2 \xrightarrow{l} B_2' \\ B \xrightarrow{l} B', B \in \mathcal{B}, l \in \mathcal{L}_\delta & \vdash & \Sigma\, \mathcal{B} \xrightarrow{l} B' \end{array}$$

A test run of an implementation with a test case is modelled by the synchronous parallel execution of the test case with the implementation under test. This run continues until no more interactions are possible, except letting the time pass.

Definition 3.6.2. *Let i be a M-quiescent input-enabled implementation in TLTS(I, O), t be a test in $\mathcal{TEST}(I, O)$ and $\mathbf{T}$ be a test suite in $\mathcal{TEST}(I, O)$, then*

- *Running t with i is modelled by the parallel operator*

$$||\quad :\quad \mathcal{TEST}(I, O) \times \mathit{TLTS}(I, O) \rightarrow \mathit{TLTS}(I, O)$$

which is defined by the following inference rules

$$\begin{array}{lcl} i \xrightarrow{\tau} i' & \vdash & t\ ||\ i \xrightarrow{\tau} t\ ||\ i' \\ t \xrightarrow{\delta} t', \forall\ l \in O : i \overset{l}{\not\rightarrow} & \vdash & t\ ||\ i \xrightarrow{\delta} t'\ ||\ i \\ t \xrightarrow{l} t', i \xrightarrow{l} i', l \in L & \vdash & t\ ||\ i \xrightarrow{l} t'\ ||\ i' \\ t \xrightarrow{d} t', i \xrightarrow{d} i' & \vdash & t\ ||\ i \xrightarrow{d} t'\ ||\ i' \end{array}$$

- *A test run of t with i, is a σ of $t \mid\mid i$ leading to a terminal state of t. Then, σ is a test run of t and i if*

$$\exists\ i' : (t \mid\mid i \stackrel{\sigma}{\Rightarrow} \mathbf{pass} \mid\mid i') \text{ or } (t \mid\mid i \stackrel{\sigma}{\Rightarrow} \mathbf{fail} \mid\mid i')$$

- i **passes** t*, if all their test runs do not lead to the* **fail** *state of* t

$$i \ \mathbf{passes}\ t \quad \triangleq \quad \forall\ \sigma : \forall\ i' : t \mid\mid i \stackrel{\sigma}{\not\Rightarrow} \mathbf{fail} \mid\mid i'$$

- i **passes** $\mathbf{T}$*, if* i**passes** *all test cases in* $\mathbf{T}$

$$i \ \mathbf{passes}\ \mathbf{T} \quad \triangleq \quad \forall\ t \in \mathbf{T} : i \ \mathbf{passes}\ t$$

If i does not pass the test suite, it fails it:

$$i \ \mathbf{fails}\ \mathbf{T} \quad \triangleq \quad \neg(i \ \mathbf{passes}\ \mathbf{T})$$

Since an implementation can behave non-deterministically, different test runs of the same test case with the same implementation may lead to different terminal states and hence to different verdicts. An implementation **passes** a test case if an only if all possible test runs lead to the verdict **pass**.

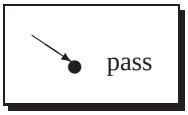
3.6.1 Test generation procedure

We define a procedure to generate test cases from a given extended specification $\Delta(S) = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ in TLTS(I, O), we call this test generation procedure as TTGP (from timed test generation procedure). Similar as in Chapter 2, test cases result from the non-deterministic, recursive application of three test generation steps from $\mathcal{Q}'$, corresponding to

1. *termination*,
2. *generation of an input*, and
3. *observation of outputs* (including quiescence).

Where the set of $\mathcal{Q}' \in \mathcal{Q}$ represents the set of all possible states in which the specification can be at the current stage of the test case execution. Initially $\mathcal{Q}'$ is equal to $\{q^0\}$.

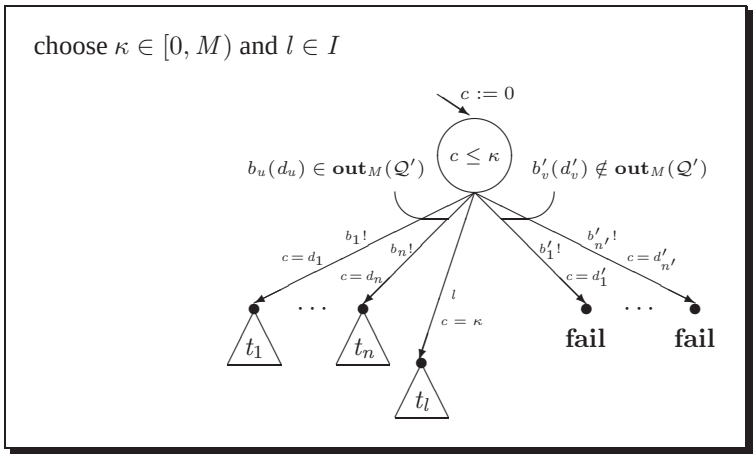
1. *termination*



$$t \quad := \quad \mathbf{pass}$$

The single state test case **pass** is always a sound test case. It stops the recursion in the algorithm, and thus terminates the test case.

2. *inputs*

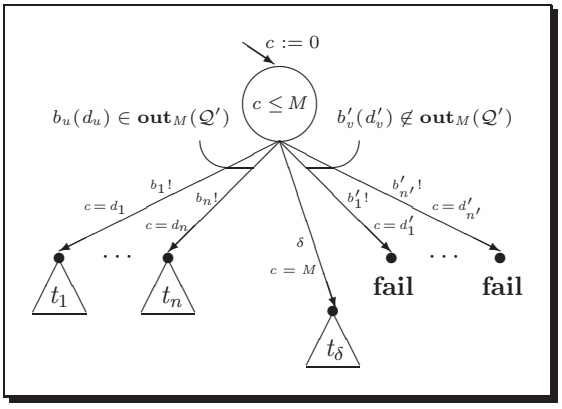


$$
\begin{array}{rcll}
t := & & \Sigma\,\{b_u(d_u);\ t_u & \mid \quad b_u \in O \wedge d_u < \kappa \wedge b_u(d_u) \in \mathbf{out}_M(\mathcal{Q}')\} \\
& + & l(\kappa);\ t_l & \\
& + & \Sigma\,\{b'_v(d'_v);\ \mathbf{fail} & \mid \quad b'_v \in O \wedge d'_v < \kappa \wedge b'_v(d'_v) \notin \mathbf{out}_M(\mathcal{Q}')\}
\end{array}
$$

where c is a clock, κ is a time constant, M is the bound to detect quiescence, u is in $[1, n]$, v is in $[1, n']$ and t_u and t_l are obtained by recursively applying the algorithm to $(\mathcal{Q}'\ \mathbf{after}\ b_u(d_u))$ and $(\mathcal{Q}'\ \mathbf{after}\ l(\kappa))$, respectively.

The test case t is waiting for κ time units, and trying to perform an input (l). If an output from the implementation arrives before time κ, the test checks whether the output is an invalid response, i.e. $b'_v(d'_v) \notin \mathbf{out}_M(\mathcal{Q}')$; in that case, the test case terminates with **fail**. If the output is valid the test case continues recursively with t_u. If the intended time has passed (i.e. $c = \kappa$) then the test produces the input (l) and continues with the test case t_l.

3. *observations of outputs*



$$
\begin{array}{rlll}
t := & \Sigma\,\{b_u(d_u);\ t_u & \mid & b_u \in O \wedge d_u < M \wedge b_u(d_u) \in \mathbf{out}_M(\mathcal{Q}')\} \\
+ & \Sigma\,\{\delta(M);\ t_\delta & \mid & \delta \in \mathbf{out}_M(\mathcal{Q}' \mathbf{\ after\ } M)\} \\
+ & \Sigma\,\{\delta(M);\ \mathbf{fail} & \mid & \delta \notin \mathbf{out}_M(\mathcal{Q}' \mathbf{\ after\ } M)\} \\
+ & \Sigma\,\{b'_v(d'_v);\ \mathbf{fail} & \mid & b'_v \in O \wedge d'_v < M \wedge b'_v(d'_v) \notin \mathbf{out}_M(\mathcal{Q}')\}
\end{array}
$$

where c is a clock, M is the bound to detect quiescence, u is in $[1, n]$, v is in $[1, n']$ and t_u and t_δ are obtained by recursively applying the algorithm to $(\mathcal{Q}' \mathbf{\ after\ } b_i(d_i))$ and $(\mathcal{Q}' \mathbf{\ after\ } \delta(M))$, respectively.

The test case t is waiting for M time units; if an output arrives from the implementation it checks whether it is an invalid response, i.e. $b'_u(d'_u) \notin \mathbf{out}_M(\mathcal{Q}')$; in that case, the test case terminates in **fail**. If it is a valid response after the time passed, the test case continues recursively. The observation of quiescence δ is treated separately, using the constant M given by the M-quiescent property.

The construction steps involve (negations of) predicates of the form $b(d) \in \mathbf{out}_M(\mathcal{Q}')$, which at the general level of TLTSs are undecidable. The procedure given here, therefore, should be seen as a meta-algorithm that can be used to generate tests effectively for subclasses of TLTSs for which these predicates are decidable, such as timed automata [38, 39].

We present an example of our test case generation based on a timed automaton model of our cash machine, similar to the previous one.

Example 3.6.3. *In Figure 3.7 we present three systems in TA notation, where again the initial locations are represented with a double circle. On the left upper part we see the specification as already shown in Example 3.3.5, augmented with an explicit denotation for quiescence with straight self-loops. On the left bottom part is the implementation, where we do not represent quiescence because we suppose it is a black box implementation. We suppose that every action (in both systems) reset the clock c.*

We can see that the implementation is slightly different to the one presented in Example 3.5.6. In this one after asking for 7 Euros, if the machine does not give it to us even if we insist we can not obtain them. A similar behaviour happen for 2 Euros in the other branch of the machine.

On the right of the figure, we present a test case that was derived using the TTGP from Section 3.6.1. We assume $M = 5$.

We suppose the timing between each action is one unit of time or M. Assume that we let one time unit to pass and then insert the card. After that, we ask the machine for 7 Euros. Then, we observe that there is no reaction, so we infer quiescence; then, we insist (for example by quirking the machine) and ask again for 7 Euros. In normalized ttrace notation, this is as follows

$$\mathit{card}(1) \cdot \mathit{ask\_7€}(1) \cdot \delta(M) \cdot \mathit{insist}(1) \cdot \mathit{ask\_7€}(1)$$

In this situation, we would like to obtain 7 Euros, as we see in the specification this is the case, because

$$\mathbf{out}(spec \textbf{ after } \mathit{card}(1)\cdot\mathit{ask\_7€}(1)\cdot\delta(M)\cdot\mathit{insist}(1)\cdot\mathit{ask\_7€}(1)) \quad = \quad \{\mathit{give\_7€}[0,\infty)\}$$

Since the output set in the timed case is infinite, we use the give_7€$[0,\infty)$ notation to denote that the output give_7€ can be produced at any time between 0 and ∞.

The test on the right hand side of Figure 3.7 was produced to test the nttrace that we just described. If we apply this test to the implementation of the left bottom hand side of Figure 3.7, we obtain

$$\mathbf{out}(\textbf{impl after } \mathit{card}(1)\cdot\mathit{ask\_7€}(1)\cdot\delta(M)\cdot\mathit{insist}(1)\cdot\mathit{ask\_7€}(1)) \quad = \quad \{\delta(5)\}$$

In this way we know that this implementation is not **tioco**$_M$ *correct with respect to the specification, because it produces the unspecified output $\delta(5)$.*

3.7 Completeness

In this section we prove that our test generation procedure (TTGP) presented in Section 3.6.1 is complete with respect to the **tioco**$_M$ implementation relation, i.e. it is sound and exhaustive. To be precise we adapt Definition 2.6.1 (from Chapter 2, page 23) to the particular case of **tioco**$_M$.

Definition 3.7.1. *Let S be a specification in TLTS(I, O). Then for all i a M-quiescent input-enabled implementation in TLTS(I, O) and for $\mathbf{T}$ composed by all test cases obtained from S by the TTGP*

$$\begin{array}{lcl} \mathbf{T} \textit{ is sound w.r.t. to } \textbf{tioco}_M & \triangleq & \forall\ t \in \mathbf{T} : \textit{if } i\ \textbf{tioco}_M\ S \textit{ then } i\ \mathbf{passes}\ t \\ \mathbf{T} \textit{ is exhaustive w.r.t. to } \textbf{tioco}_M & \triangleq & \textit{if } i\ \textbf{tioco}_M\ S \textit{ then } \exists\ t \in \mathbf{T} : i\ \mathbf{passes}\ t \end{array}$$

A test suite, generated by the TTGP, is sound with respect to **tioco**$_M$, if any implementation that fails a test in the set is incorrect with respect to **tioco**$_M$ relation. A test suite is exhaustive with respect to **tioco**$_M$ relation, if for every incorrect implementation the test suite contains a test case that detects the non-conformance.

In order to archive these two results we first have to establish some technicalities. We saturate an nttrace with δ. An nttrace is considered saturated when for each opportunity of a δ symbol to appear, it actually does it.

Definition 3.7.2. *Let $\mathcal{A}$ be a system in TLTS(I, O) and $\sigma \in \mathsf{nttraces}^{\Delta}_M(\mathcal{A})$, then*

$$\begin{array}{lcl} \sigma \textit{ is } \delta(M)\textit{-saturated} & \triangleq & \nexists\, \sigma' \in \mathsf{nttraces}^{\Delta}_M(\mathcal{A}) : \sigma = \sigma_1\cdot l(d)\cdot\sigma_2 \wedge \\ & & \qquad\qquad \sigma' = \sigma_1\cdot\delta(M)\cdot l(d-M)\cdot\sigma_2 \end{array}$$

Proposition 3.7.3. *Let i be a M-quiescent input-enabled implementation in TLTS(I, O), then*

if σ *is* $\delta(M)$*-saturated* *then* $\forall\ l(d) \neq \delta(M)$ *occurring in* $\sigma : d < M$

Proof.
By contradiction. Assume that σ is $\delta(M)$-saturated and there exists $l(d) \neq \delta(M)$ with $d \geq M$. Then, because l can not be an output, since i is M-quiescent, l is an input, and $\sigma = \sigma_1 \cdot l(d) \cdot \sigma_2$ and $d \geq M$. So, there exists σ' such that $\sigma' = \sigma_1 \cdot \delta(M) \cdot l(d-M) \cdot \sigma_2$. Thus, σ is not $\delta(M)$-saturated, which is a contradiction to our initial assumption.

□

3.7.1 Soundness

The TTGP is sound in the sense that it does not wrongly reject implementations by mistake. This section formalizes and proves this property.

Theorem 3.7.4. *Let* S *be a specification in TLTS*(I, O)*. Then for all* i M*-quiescent input-enabled implementations in TLTS*(I, O) *and for all* t *a test case obtained from* S *by the TTGP*

if $\quad i$ **tioco**$_M$ S $\quad$ *then* $\quad i$ **passes** t

Proof.
Let i be a M-quiescent input-enabled implementation with i **tioco**$_M$ S. We prove that for all $\sigma \in \Delta_M(S)$ and for all t a test cases, generated by the TTGP from S, the following holds:

$$\text{if} \quad t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \quad \text{then} \quad t' \neq \mathbf{fail}$$

Without loss of generality we can assume that σ is $\delta(M)$-saturated. Moreover, we can assume that $t \,||\, i \stackrel{\sigma}{\Rightarrow}$, because otherwise there is nothing to show. We prove the theorem by induction over the length of σ

- if $\sigma = \epsilon$ and $t \,||\, i \stackrel{\epsilon}{\Rightarrow} t' \,||\, i'$
  if t was constructed using case 1 in the first step, then $t \,||\, i \stackrel{\epsilon}{\Rightarrow} \mathbf{pass} \,||\, i'$
  if t was constructed using cases 2 or 3 in the first step, then
  $t = t' \neq$ **fail** and all derivations of $\stackrel{\epsilon}{\Rightarrow}$ have the form: $t \,||\, i \stackrel{\epsilon}{\Rightarrow} t \,||\, i'$
- if $\sigma = \sigma' \cdot a$ and $t \,||\, i \stackrel{\sigma'}{\Rightarrow} t'' \,||\, i'' \stackrel{a}{\Rightarrow} t' \,||\, i' \wedge a = l(d)$,
  there are only two possibilities to construct t':
  from case 2: $a = l(d) \wedge l \in I \wedge d < M$ and
  $t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \wedge\ t' \neq$ **fail**
  from case 3: $a = l(d) \wedge a \in \mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma)$. Since
  i **tioco**$_M$ S, $a \in \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$, and thus $t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \wedge\ t' \neq$ **fail**

□

3.7.2 Exhaustiveness

The TTGP presented in Section 3.6.1 is also exhaustive with respect to the **tioco**$_M$ testing relation. This means that for each non-conforming implementation, a test case can be gener-

ated that exposes the error. Hence, if we test enough (perhaps infinite times), it is possible to recognize all erroneous implementations.

Lemma 3.7.5. *Let S be a specification in TLTS(I, O), $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(S)$ be $\delta(M)$-saturated, and t' be a test case generated by the TTGP from $(\Delta(S)\ \mathbf{after}\ \sigma)$. Then there exists t a test case generated from S with $t \stackrel{\sigma}{\Rightarrow} t'$.*

Proof.
By induction over the length of σ, $(|\sigma|)$:

- If $|\sigma| = 0$ then take $t = t'$.
- Suppose there exists t for all σ with $|\sigma| < n$.
- Let $|\sigma| = n$ with $\sigma = \sigma' \cdot a$ and $a = l(d)$

  if $l \in I$, using case 2 for the input $l : t \stackrel{\sigma'}{\Rightarrow} t'' \stackrel{a}{\Rightarrow} t'$

  if $a \in \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$, using case 3 : $t \stackrel{\sigma'}{\Rightarrow} t'' \stackrel{a}{\Rightarrow} t'$

□

Theorem 3.7.6. *Let S be a specification in TLTS(I, O). Then for all i M-quiescent input-enabled implementation in TLTS(I, O) with i $\mathbf{tioco}_M$ S, there exists t a test case generated by the TTGP from S such that*

$$i \quad \mathbf{pass\!\!\!/es} \quad t$$

Proof.
If i $\mathbf{tioco}_M$ S then there exists $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(S)$ such that

$$\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \quad \not\subseteq \quad \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$$

Without loss of generality we can assume that σ is $\delta(M)$-saturated. Then, let $l(d)$ be in $\mathbf{out}_M(\Delta(i)\ \mathbf{after}\ \sigma) \setminus \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$ and $i \stackrel{\sigma}{\Rightarrow} i' \stackrel{l(d)}{\Rightarrow} i''$.

Let t' be the result of applying case 3 of the procedure to $(\Delta(S)\ \mathbf{after}\ \sigma)$, and let t be the test case constructed out of t' and σ by the Lemma 3.7.5. Then, because $l(d) \notin \mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$ then $t \parallel i \stackrel{\sigma \cdot l(d)}{\Longrightarrow} \mathbf{fail} \parallel i''$. Then using Definitions 3.6.2, i $\mathbf{pass\!\!\!/es}$ t.

□

The exhaustiveness of our test generation procedure is less useful than the corresponding result in the untimed case. There, the exhaustiveness implies that the test generation algorithm, if repeatedly executed in a fair non-terminating manner, will generate all test cases in the limit, and therefore, in the limit, will achieve full coverage with respect to a given specification S and the **ioco** relation.

In our timed case, as presented, the number of potential test cases is not only infinite but also uncountable, resulting from our underlying continuous timed model. Moreover, a non-countable repetition of test generations are necessary to be performed. Nevertheless, it is still possible to obtain a version of the stronger form of exhaustiveness for real-timed test generation as well. This is done by considering equivalence classes of (minimal) error

traces. It can be shown that with reasonable assumptions in our test generation procedure, in the limit, we will hit each such equivalence class.

In the timed automata setting, the decision of the predicate $l_u(d_u) \in \mathbf{out}_M(\mathcal{Q}')$ amounts to reachability analysis. For the simpler version of **tioco**$_M$ based on timed trace inclusion (i.e. excluding quiescence) this has already been implemented in the tool environment IF [38], and in the UPPAAL-based testing tool T-UPPAAL. Also within a timed automata setting, although with the addition of quiescence, there exists a real-time extension of the TORX tool [12].

3.8 Relation with ioco

All our work was inspired by the **ioco** testing relation, as we already mentioned. Here, we present a theorem that characterizes the relation between the **tioco**$_M$ and the **ioco** testing relations.

Given that **tioco**$_M$ is defined over TLTSs and **ioco** over LTSs, we need to relate a TLTS with a LTS. To this end, initially, we define the function un, which transforms a trace in $\mathsf{nttraces}^{\Delta}_M$ into a simple trace.

Definition 3.8.1. *The function* $\mathrm{un} : \mathsf{nttraces}^{\Delta}_M \rightarrow \mathsf{traces}$*, is recursively define as*

$$\begin{array}{lcl} \mathrm{un}(\varepsilon) & \triangleq & \varepsilon \\ \mathrm{un}(\sigma \cdot l(d)) & \triangleq & \mathrm{un}(\sigma) \cdot l \end{array}$$

Thus, function un extracts the untimed trace from a timed one. Now, let $\mathcal{A}_1$ be a M-quiescent input-enabled TLTS with $\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_1)$ (the set of $\mathsf{nttraces}^{\Delta}_M$ from $\mathcal{A}_1$) and let $\mathcal{A}_2$ be a TLTS with $\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)$ (the set of $\mathsf{nttraces}^{\Delta}_M$ from $\mathcal{A}_2$). We can always obtain through the un function the set of untimed traces from $\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_1)$ and $\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)$. Now, in case that there exists an input-enabled LTS A_1 such that its set of traces $\mathsf{traces}(A_1)$ is equal to $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_1))$ and a LTS A_2 such that its set of traces $\mathsf{traces}(A_2)$ is equal to $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2))$ then we can define our relation.

Theorem 3.8.2. *Let $\mathcal{A}_1$ be a M-quiescent input-enabled implementation and $\mathcal{A}_2$ be a specification both in TLTS(I, O). Let A_1 be an input-enabled implementation and A_2 be a specification both in LTS(I, O) with* $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_1)) = \mathsf{traces}(A_1)$ *and* $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)) = \mathsf{traces}(A_2)$*. Then,*

$$\textit{if} \quad \mathcal{A}_1 \ \mathbf{tioco}_M \mathcal{A}_2 \quad \textit{then} \quad A_1 \ \mathbf{ioco} \ A_2$$

Proof.
We assume that $\mathcal{A}_1$ **tioco**$_M$ $\mathcal{A}_2$ and we prove this theorem proving that:

$$\forall\ \sigma \in \mathsf{traces}(A_2) : \forall\ b \in O : \quad \text{if } b \in \mathbf{out}(A_1 \ \mathbf{after}\ \sigma) \text{ then } b \in \mathbf{out}(A_2 \ \mathbf{after}\ \sigma)$$

Let $\sigma \in \mathsf{traces}(A_2)$ and $b \in \mathbf{out}(A_1 \ \mathbf{after}\ \sigma)$. Then because $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)) = \mathsf{traces}(A_2)$, we know that there exists $\sigma' \in \mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)$ such that $\mathrm{un}(\sigma') = \sigma$. Now,

because $b \in \mathbf{out}(A_1 \mathbf{after} \sigma)$ it follows that there exists $d \in \mathcal{D}$ such that $b(d) \in \mathbf{out}(\mathcal{A}_1 \mathbf{after} \sigma')$. Thus, since $\mathcal{A}_1$ **tioco**$_M$ $\mathcal{A}_2$ and $\sigma' \in \mathsf{nttraces}^{\Delta}_M(\mathcal{A}_2)$, so we have $b(d) \in \mathbf{out}(\mathcal{A}_2 \mathbf{after} \sigma')$, thus $b \in \mathbf{out}(A_2 \mathbf{after} \mathrm{un}(\sigma'))$, then $b \in \mathbf{out}(A_2 \mathbf{after} \sigma)$.

□

For all (possibly infinite) sets of traces of the form $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}))$, it is possible to construct a LTS A such that $\mathsf{traces}(A) = \mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}))$. Moreover, our construction only uses traces in which at most one δ occurs between subsequent actions, we call this property *simple quiescence occurrence*.

Let B be the set of simple quiescence occurrences in $\mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}))$ then we construct the LTS $A = \langle Q, q^0, L, T \rangle$, where Q is the set of states of the form q_σ for all traces $\sigma \in B$; L is the same set of labels as $L_{\mathcal{A}}$; and T is the set of transitions such that for each trace σ and each action l if $\sigma{\cdot}l{\cdot}\delta \notin B$ then we create a transition $q_\sigma \xrightarrow{l} q_{\sigma l}$, and if $\sigma{\cdot}l{\cdot}\delta \in B$ then we create two transitions $q_\sigma \xrightarrow{l} q_{\sigma l}$ and $q_\sigma \xrightarrow{l} q_{\sigma l \delta}$. Following this construction it is straightforward to see that $\mathsf{traces}(A) = \mathrm{un}(\mathsf{nttraces}^{\Delta}_M(\mathcal{A}))$.

The result presented in this section means that the **tioco**$_M$ relation is a sound extension of the **ioco** relation. Note that to obtain this result, it is crucial to consider quiescence in the timed relation.

3.9 Related work

There are several approaches that focus on the development of testing methods for timed systems [56, 52, 45, 32, 24, 18, 16, 20, 31]. Most of them, in contrast with our work, which assumes generic TLTS as the basic model, are developed over a subclass of TA. In the following, we discuss some of the most interesting previous results.

- Nielsen and Skou [52] describe a fully automatic method for the generation of real-time test sequences from a subclass of timed automata called Event-Recording Automata (ERA). Their technique is based on a symbolic analysis of TA inspired by the UPPAAL model-checker. Test sequences are selected by covering an equivalence class partitioning of the state space. They argue that this approach provides a heuristic that guarantees that a well-defined set of interesting scenarios in the specification is automatically, completely, and systematically explored. However, this method lacks a suitable notion of an implementation relation.
- Springintveld, Vaandrager, and D'Argenio [56] show that exhaustive testing (with respect to bisimulation[2]) of deterministic timed automata with a dense time interpretation is theoretically possible. The testing of timed systems is described as a variant of the bounded time-domain automaton (BTA). The BTA describing the specification is transformed into a region automaton, which in turn is transformed into another finite state automaton, the Grid Automaton. Test sequences are then generated from the Grid Automaton. The idea behind the construction of the Grid Automaton is to represent each clock region with a finite set of clock valuations, referred to as the representatives of the clock region. However, although being exact, their grid method is impractical because as said in [56] it generates "an astronomically large number of test sequences".

[2] Actually, for the deterministic case, bisimulation and trace equivalence coincide [62].

- Cardell-Oliver [16] presents a testing method for networks of deterministic TA extended with integer data variables. She checks for trace equivalence, but this is done only for those parts of a system that are visibly observable. In addition to the usual time-discretization, test views are used to discriminate between states depending on a test-purpose. Test views partition variables and events into visible and hidden ones. Equivalence on visible clocks and variables induces an equivalence relation on states. States that are evidently different, i.e. that are in different visible equivalence classes, need to be distinguished from each other. This significantly reduces the length of test suites, at the price that the test is not always exhaustive.

Further details discussing the above these approaches can be found in the survey [4]. Most of these works have not been implemented they are more theoretical approaches.

More recently, these theoretical works have been improved to be more practical as our proposed theory. Closely related to our work is the one done by Larsen et al. [39] and by Krichen et al. [38]. Both of them are extensions of the **ioco** testing theory, with quiescent-free interpretations over TA and their relations is inclusions of timed traces. The more important difference between them is that in [39] the environment is an explicit part of the model. In contrast in our work we provide a framework at the TLTS level and we include quiescence. As follows, in Figure 3.8, we present a comparative table over the **ioco** testing theory extensions with time.

3.10 Conclusion

In this chapter, we present an extension with real-time of Tretmans' **ioco** testing theory and test generation framework. The extension allows us to test timed systems, particularly TLTSs. Our treatment is based on an operational interpretation of the notion of quiescence that gives rise to a family of implementation relations parameterized by observation durations M of quiescence. These relations detect differences in behaviour after the execution of timed traces (extended with quiescence) provided that all observations of quiescence take longer than the stipulated duration M.

We explain how this theory can be used to test TLTSs, under the assumption that the absence of system interaction with its environment for M time units implies quiescence. We have defined a non-deterministic (M-parameterized) test generation framework that generates sound test cases with respect to the corresponding implementation relation $\mathbf{tioco}_M$. The test generation is also exhaustive. This means that for each non-conforming implementation, a test case detecting the non-conformance can be generated.

Our framework can be effectively instantiated for subclasses of TLTSs, as long as the instance satisfies that $\mathbf{out}_M(\Delta(S)\ \mathbf{after}\ \sigma)$ is computable; this is the case, for instance, on TA. Using standard symbolic state space representation in the form of difference bound matrices [23], a real-time version of TORX for TA models has also been recently developed in [12] which incorporate quiescence.

Our work, as presented here, can be extended in several ways. Unlike the untimed case, here the test cases are not only infinite but also uncountable. Then our notion of completeness guarantees that for a non-conforming implementation a test case "can" be generated. But, we can not be sure that this test case "will" be generated. We believe it is possible to show a stronger exhaustiveness result for the test generation procedure, based on an appropriate

notion of equivalence of error traces. The generation procedure will hit each such class in the limit, provided that the error class is not negligible, i.e. it must have positive measure in some appropriate sense.

Another interesting extension is to relax the requirement that there must be a uniform observation deadline M for quiescence. Alternatives are:

- To make the observation parameter $M(\sigma)$ a function of the behaviour (trace) σ observed so far. This would allow us to model sequential phases of quiescence, i.e. slow vs. quick response times.
- To divide the output set in subset of communication channels and make the observation parameter $M(C_i)$ a function of the communication channel C_i on which output (from that channel) are being observed. This allows us to model different kinds of response times for different communication channels with the system under test. This idea corresponds to a real-time extension of the **mioco** implementation relation of [28]. This approach is the one we present in the next Chapter 4.

Our real-time theory inherits its focus on control aspects of system behaviour from the existing **ioco** theory. Ultimately, it will be important to combine this testing theory with methods for testing the static data aspects of systems. It will be interesting to see up to what extent the symbolic representation of data types can be combined with symbolic representations of time.

Thanks to several enriching discussions on the application of our theory in the TORX tool, we found that in a more general vein, it can be stated that the development of a real-time testing theory forces us to confront modeling issues with respect to physical aspects of time and implementation. From a physical point of view, for example, it is questionable whether negligible behaviour can be implemented (e.g., Example 3.2.9). This has also implications for specification formalisms that can be used to specify such behaviour, e.g. TA can define negligible behaviour by using guards that force behaviour to go through specific points in time, such as $\kappa = 3$. Perhaps, realistic specifications and/or implementation relations should allow for tolerance in the evaluation of clock conditions. This would then introduce another source of nondeterminism in the testing theory of real-time systems. At any rate, a more systematic study of the formal aspects of tolerance and robustness is a useful and interesting future work.

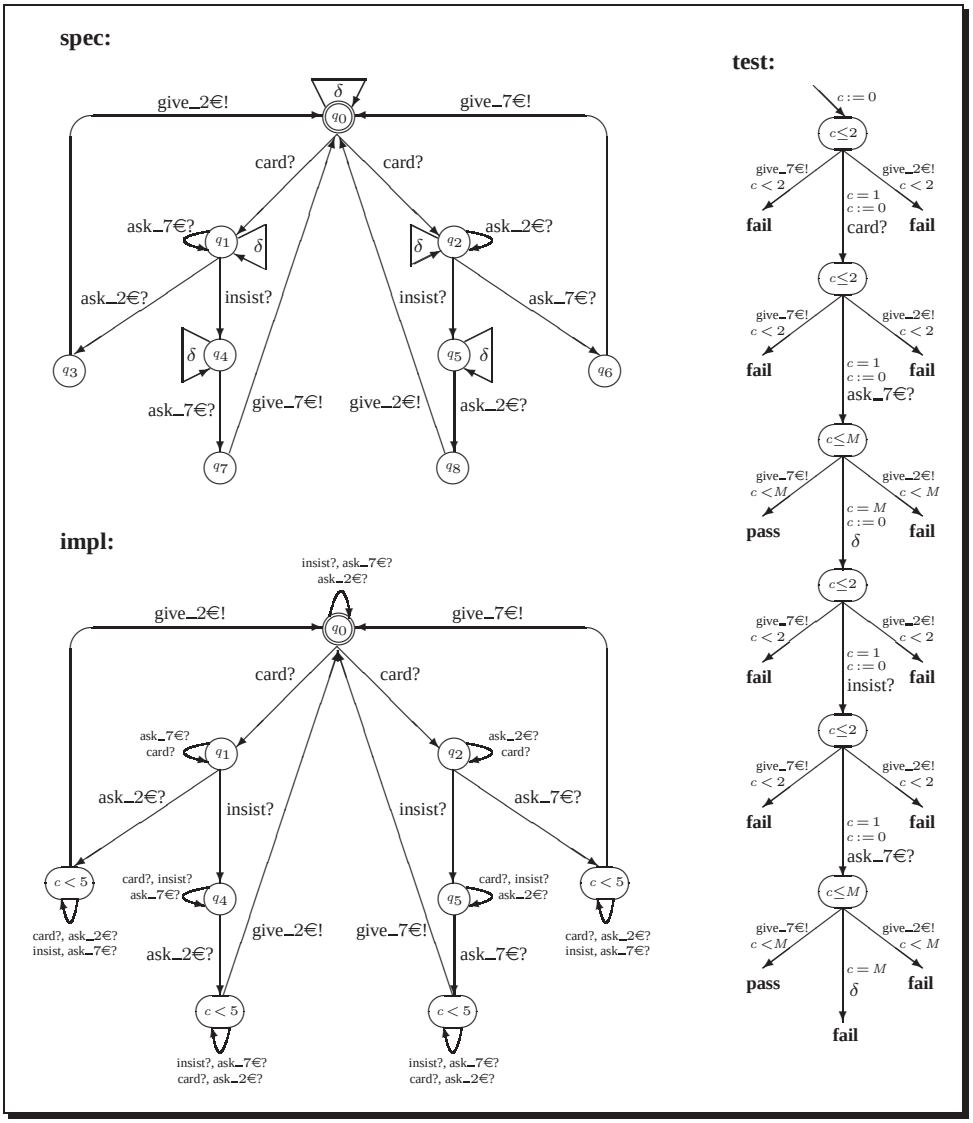


Figure 3.7: A specification of a cash machine (**spec**), an M-quiescent implementation (**impl**), and a test case derived from the specification (**test**). We suppose that every transition reset the clock c

	Spec Model	Impl Model	Conformance relation	Test
Larsen et al.	TA with environment e are non-deterministic, have τ, are strong input-enabled, and are non-blocking e is output-enabled	TA are non-deterministic, have τ, are strong input-enabled, and are non-blocking	rtioco$_e$ out set: outputs and time	are tree in the **ioco** style
Krichen et al.	TA with lazy, delayable, and eager deadlines, has τ, is non-blocking	TA with lazy, delayable, and eager deadlines, has τ, is non-blocking, strong input-enabled	tioco out set: outputs and time	are total function
Brandán Briones et al.	TLTS is non-deterministic, has τ, is no forced inputs, has time divergent	TLTS is non-deterministic, has τ, is no forced inputs, has time divergent, and is weak input-enabld	**tioco**$_M$ out set: outputs with time and δ with (bounded) time	are tree in the **ioco** style

Figure 3.8: Timed **ioco** extensions

CHAPTER 4

Testing timed labelled multi input-output transition systems

4.1 Introduction

As mentioned in Chapter 2, we perform black box testing by communicating with the environment in terms of inputs and outputs. In the previous chapters we always assumed that implementations are systems where input actions are never refused. More precisely, we assumed that IUTs are input enabled, i.e., IUTs accept all inputs at any time. Although many systems can be modelled with that requirement, there is still a significant portion of realistic systems that cannot. As an example, consider a cash machine equipped with a card slot. After a card is inserted in the slot (which can be seen as an input) another card can not be inserted; the input card is no longer enabled.

In this new setting, a system may have set of actions enabled or disabled depending on its particular execution stage. It seems convenient to think that the actions between a system and its environment are triggered through communication channels. In the example of the cash machine, the slot to enter cards is a possible channel, where the action of inserting a card happens. Similarly, the cash machine may have a keyboard, which can be considered as another channel in which the action of entering information (e.g., a Pin number) happens. To model the fact that actions are enabled or disabled at different execution stages, it is sufficient to enable or disable the communication channels in which the actions occur.

To deal with this new setting, in the untimed case, the multi input-output transition systems (called MIOTS) were proposed. This work was done by Heerink [28]. There, the **ioco** testing theory is extended to deal with multiple channels.

It is natural to investigate whether the timed testing theory presented in Chapter 3 can also be extended to deal with multiple channels, thus completing Figure 4.1 (bottom right). This means that both features, namely communication channels and time, can be combined together in a richer model. Fortunately, in this chapter we answer this question affirmatively by extending our timed testing theory to account for multiple channels.

The resulting model is the timed labelled multi input-output transition systems (TLMTS), and it allows us to consider input enabledness and quiescence properties not only for the entire system but on a per channel basis, thus relaxing global system assumptions.

Formally, channels are represented as a partition on the input and output sets of actions, respectively. Each partition class defines the inputs (outputs) belonging to an individual input

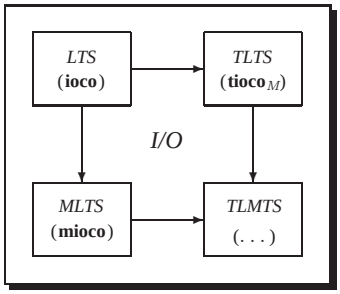


Figure 4.1: Relation between test generation approaches

(output) channel. Following Heerink's work in the untimed case, we replace the input enabling requirement of a system by the following weaker requirement: for each input channel, either all inputs are allowed, or they are all blocked. For example, recall the cash machine example with an input channel where the action of inserting a card may occur. There, the requirement naturally follows the intuition: initially, the slot accepts a card (that is, the input channel is enabled). However, after a card is inserted in the slot, no further cards are accepted (that is, the input channel is blocked).

Additionally, we relax the treatment of quiescence. We replace the global bound M from the **tioco**$_M$ testing relation by a vector of bounds $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$; where M_j represents the bound on the output channel j. In the **tioco**$_M$ relation the global bound M is a parameter that tells us how long a system must be silent before we conclude quiescence. Relaxing the global bound M for a vector of bounds means that we do not have to wait for the slowest response time to conclude the quiescence on a faster channel.

The combination of these ideas is formalized as the **mtioco**$_{\mathcal{M}}$ conformance implementation relation. We develop a test derivation procedure for **mtioco**$_{\mathcal{M}}$, which is shown to be sound and exhaustive with respect to the **mtioco**$_{\mathcal{M}}$ implementation relation.

Therefore, our results can be seen as a real-time extension of Heerink's **mioco** testing theory, where we introduce the channel-based treatment of input enabling-blocking and quiescence in the timed setting.

Organization of the chapter The chapter consists of two main parts: models and relations (Sections 4.2, 4.3, and 4.4) and the test generation framework (Sections 4.5 and 4.6).
Initially, in Section 4.2, we introduce the timed labelled input-output transition systems. In this model we suppose that the input set and output set can be partitioned into subsets, the so called channels. However, so far we do not require the system to have any special property on these sets. Over this model we define the tmior conformance relation, which relates systems with respect to a subset of failure timed traces where the refusals can only be complete channels. Refusals of complete channels mean that if one action in a channel is refused, then every other action in that channel is also refused. This is in contrast to the definition of refusals given in Chapter 2 (in Definition 2.2.8, page 15), where any subset of actions can be refused, e.g. one part of a channel may be refused while another is not.
In Section 4.3, the timed transition relation of the previous model is extended with

quiescence and refusals. As a consequence, states of the implementation are required to be input enabled or blocked per channel. Furthermore, since we consider quiescence, we introduce the bounds vector that is necessary to recognize quiescence per channel. This new feature allows us to define a parameterized conformance relation in terms of the bounds vector. The resulting relation is called mtiorf. In Section 4.4, the concept of observed output set is introduced and the **mtioco**$_{\mathcal{M}}$ conformance relation is given, a relation that only takes into account timed traces from the specification with quiescence and blocking per channel.
Subsequently, in Section 4.5, we develop a parameterized non-deterministic test derivation procedure. Moreover, the set of tests obtained by our proposed procedure are proved to be sound and exhaustive with respect to our **mtioco**$_{\mathcal{M}}$ relation in Section 4.6. Finally, in Section 4.7 we prove a result that relates the proposed timed extension with channels and the timed relation **tioco** presented in Chapter 3. Then, Section 4.9 presents the conclusions.

4.2 A basic model and a basic conformance relation

We start from the model described in Chapter 3, assuming only one extra feature: we ask the input and output sets to be partitioned in disjoint subsets, that we call *channels*.

4.2.1 Timed labelled transition systems with partitioned input and outputs

Through all this chapter we use Definition 3.2.1 (from Chapter 3, page 28) where TLTSs are defined. Recall that $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ a TLTS(I, O) is a labelled transition system where labels can be: external actions (L), an internal action (τ) or time passage actions ($\mathcal{D}$). In addition, all other definition and restrictions from Section 3.3 (page 33) also apply to this chapter.

Up to now we have considered TLTSs with external actions partitioned into only two set: inputs and outputs. In the following we consider TLTSs with the external actions sub-partitioned in to subsets, the channels.

Channels Given a TLTS(I, O), we assume that there exists partitions of the input and output sets into channels, i.e. $I = I_1 \cup \ldots \cup I_n$, $O = O_1 \cup \cdots \cup O_m$ and for all $k \neq z$ then $I_j \cap I_z = \emptyset$ and for all $j \neq z'$ then $O_j \cap O_{z'} = \emptyset$. We denote the partitions as $\mathcal{I} = \{I_1, \cdots, I_n\}$, $\mathcal{O} = \{O_1, \cdots, O_m\}$, and the TLTS is denoted as TLTS$(\mathcal{I}, \mathcal{O})$.

Although our framework is based on TLTSs, for simplicity on the graphical notation all the examples we present are given as timed automata (TA). The definition of a TA and its corresponding semantics in terms of TLTSs can be found in Chapter 3, Section 3.2.1.

Example 4.2.1. *Our example, illustrated on Figure 4.2 and already anticipated above, is an adapted version, with time, of the cash machine from [28]. In the machine, a card may be inserted in the slot, and for* 5 *time units a Pin number can be typed in. After that, the machine decides whether the Pin number is correct; if it is not, the machine returns the card. If the Pin is correct, a desired amount of money can be requested. In case the machine has*

sufficient money, it will return the card and then gives the requested money. However, if there is not enough money it will produce an error and return the card.

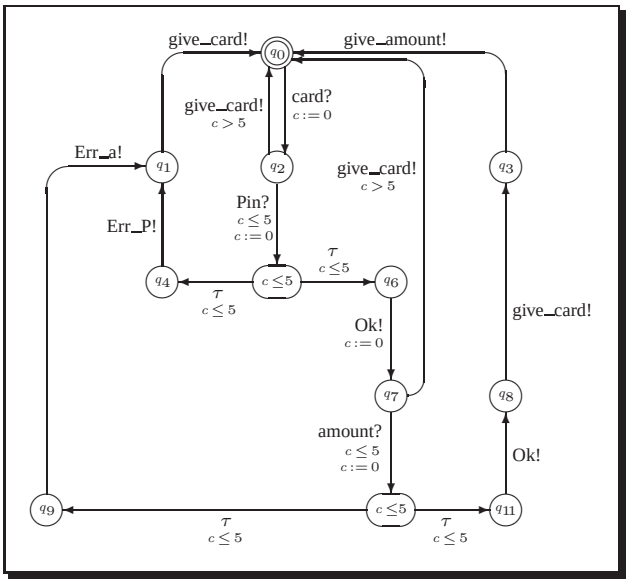


Figure 4.2: A timed cash machine

We can describe Figure 4.2 in terms of Definition 3.2.1, where the cash machine is specified as the $\mathcal{A} \in TLTS(I, O)$ *where* $\langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$*, with* $I = \{card, Pin, amount\}$ *and* $O = \{give\_card, give\_amount, Ok, Err\_P, Err\_a\}$.

We partition the input set in two sets, with respect to the access points of introducing the card in the slot and typing in the numbers in the keyboard, for the Pin and desired amount money. We obtain two subsets: $I_1 = \{card\}$ *and* $I_2 = \{Pin, amount\}$*. In a similar way, the output set can be partitioned with respect to the access points: the location in which we receive the card, where we receive the money and where we receive other messages (e.g., the screen). We obtain three subsets:* $O_1 = \{give\_card\}$, $O_2 = \{give\_amount\}$ *and* $O_3 = \{Ok, Err\_P, Err\_a\}$.

As in Chapter 2 (in Definition 2.2.8, page 15) where refusals are introduced, in the timed setting we can also explicitly encode the inability for a state to perform an action in the set L' or an internal action τ. Even though we consider time as a kind of action, the definition of refusal only considers the refusal of input or output actions, while it discards the time as a possible refutable action.

Definition 4.2.2. *Let* $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ *be a TLTS with* $l \in L$, $q \in \mathcal{Q}$ *and* $L' \subseteq L$*, then*

$$q \textbf{ refuses } L' \quad \triangleq \quad \forall\ l \in L'_\tau : q \overset{l}{\not\rightarrow}$$

Moreover we extend all previous definitions (like $\Rightarrow$, $\rightarrow$) to our new setting. As before, we extend the timed transition relation with self-loop transitions: $q \xrightarrow{L'} q$, in case L' is refused in the state q (slightly relaxed by abstracting from τ internal actions):

$$q \xrightarrow{L'} q' \quad \triangleq \quad \forall\ l \in L'_\tau : q \overset{l}{\not\Rightarrow} \wedge\ q = q'$$

As a result, the timed traces (ttraces, recall Definition 3.3.2, page 34) can be extended to express not only the actions that can be taken but also the sets of actions which are not allowed.

Definition 4.2.3. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS, then*

$$\mathsf{Fttraces}(\mathcal{A}) \quad \triangleq \quad \{\sigma \in (\mathcal{L} + \mathcal{P}(L))^* \mid q^0 \overset{\sigma}{\Rightarrow}\}$$

As expected, a failure timed trace (Fttrace) is a ttrace extended with sets of actions that can not be performed; in other words, with sets of actions that are refused.

Example 4.2.4. *In Figure 4.2 we can observe that in state q_0 input actions: Pin and amount are not enabled, but card is. Then this system has the Fttrace σ with $\sigma = \{$Pin, amount$\}$·card.*

4.2.2 The tmior conformance relation

The channels' partition of the input and output sets gives us the possibility to introduce the tmior-conformance relation.

Definition 4.2.5. *Let i be an implementation in TLTS$(\mathcal{I}, \mathcal{O})$ and S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$, then*

$$i \leq_{tmior} S \quad \triangleq \quad \mathsf{Fttraces}(i) \cap (\mathcal{L} + \mathcal{I} + \mathcal{O})^* \subseteq \mathsf{Fttraces}(S)$$

This conformance relation refers to the inclusion of Fttraces where the refusals can only be a set in $\mathcal{I}$ or in $\mathcal{O}$.

4.3 An extended model and its conformance relation

The tmior-conformance relation inspired us to define an extension of TLTSs where the input and output sets are subdivided in channels and where each input channel is either input-enabled or blocked.

4.3.1 Timed labelled multi input-output transition system

A *timed labelled multi input-output transition system,* TLMTS$(\mathcal{I}, \mathcal{O})$, is a TLTS$(\mathcal{I}, \mathcal{O})$ where in each reachable state, each input channel is either blocked or all inputs of that channel are

accepted (input-enabling for a particular channel).

Definition 4.3.1. *Let $\mathcal{I} = \{I_1, \cdots, I_n\}$ and $\mathcal{O} = \{O_1, \cdots, O_m\}$ with for all $k \neq z$ then $I_j \cap I_z = \emptyset$ and for all $j \neq z'$ then $O_j \cap O_{z'} = \emptyset$. Then a timed labelled multi input-output transition system (TLMTS) over $(\mathcal{I}, \mathcal{O})$ is a $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ in TLTS(I, O) with $I = \bigcup_{1 \leq k \leq n} I_k$ and $O = \bigcup_{1 \leq j \leq m} O_j$, where*

$$\forall\ q \in \mathsf{der}(q^0) : \forall\ 1 \leq k \leq n : (\forall\ l \in I_k : q \overset{l}{\not\Rightarrow}) \vee (\forall\ l \in I_k : q \overset{l}{\Rightarrow})$$

Moreover, whenever a channel I_k is blocked in state q, it is denoted as $\gamma_k(q)$.

In a similar way to the refusals, we can extend the timed transition relation in TLMTSs to denote explicitly when a channel is blocked. Hence, we add a self-loop transition in a state q with a γ_k label whenever channel I_k is blocked in state q

$$q \xrightarrow{\gamma_k} q \quad \text{if and only if} \quad \gamma_k(q)$$

Obviously, every TLTS can be interpreted as a TLMTS by having one channel for each action in the TLTS.

Example 4.3.2. *Figure 4.3 is a modified version of Figure 4.2. It is possible to see it as a specification of a cash machine with $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ in TLMTS$(\mathcal{I}, \mathcal{O})$. Where $\mathcal{I} = \{I_1, I_2\}$ and $\mathcal{O} = \{O_1, O_2, O_3\}$ with $I_1 = \{card\}$, $I_2 = \{Pin, amount\}$ and $O_1 = \{card\}$, $O_2 = \{amount\}$, $O_3 = \{Ok, Err\_P, Err\_a\}$.*

The differences between Figure 4.2 and Figure 4.3 are that now we have per state and per channel input-enabledness or blocking. In Figure 4.3 each state is input-enabled per channel or it has a state self-loop with the corresponding γ_k denoting that the input channel k is blocked in that state.

Since the model of TLMTS$(\mathcal{I}, \mathcal{O})$ from Definition 4.3.1 implies that input channels are either input-enabled or blocked, we only use this model when these properties are necessary. Otherwise, we use the most general model of TLTS$(\mathcal{I}, \mathcal{O})$, which only assumes a TLTS with its input and output sets partitioned in channels.

4.3.2 Quiescence

We take advantage of the channel partition in the output set to define quiescence. Hence, we have a definition of quiescence particularly per each channel.

In TLTS$(\mathcal{I}, \mathcal{O})$ there are two possible approaches to deal with quiescence. Firstly, we may consider the situation in which the observer can only see one channel. In this case, it is not relevant for the notion of quiescence whether the remaining channels stay silent or not. Secondly, we may consider the environment to be able to observe all possible channels.

We adopt the latter direction, assuming that the observer can see simultaneously all chan-

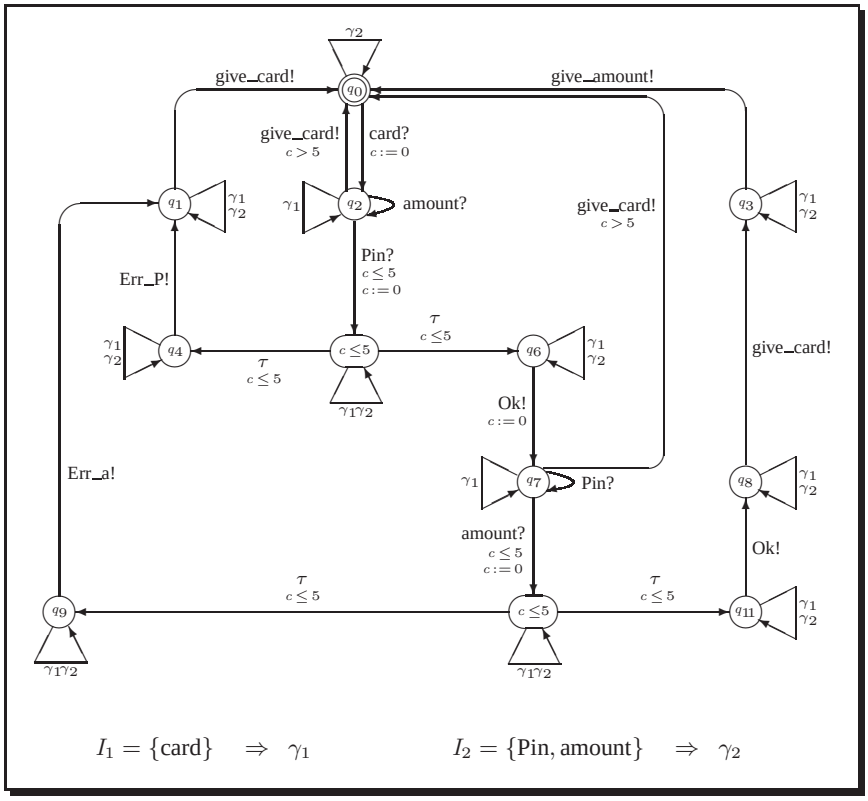


Figure 4.3: A timed cash machine with input-enabled or blocked channels

nels. This choice fits well with the testing framework of Chapter 3, where tests synchronize on all output actions. Partial observations of system outputs can be dealt with by considering modified IUTs where the unobservable channels became internal actions to the system.

Definition 4.3.3. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS$(\mathcal{I}, \mathcal{O})$ with $q \in \mathcal{Q}$, then q is called O_j-quiescent, denoted $\delta_j(q)$, if*

$$\forall\ l \in O_j : \forall\ d \in \mathcal{D} : q \overset{l(d)}{\not\Rightarrow}$$

With the definition of O_j-quiescent we can extend the timed transition relation (as we do it with refused channels) to include self-loop transitions for quiescence per channel. We use the δ_j symbol to denote that the channel O_j is quiescent.

$$q \overset{\delta_j}{\rightarrow} q \quad \text{if and only if} \quad \delta_j(q)$$

In order to denote explicitly when $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ a TLTS$(\mathcal{I}, \mathcal{O})$ has its timed transition relation $\mathcal{T}$ extended with quiescence and refusals, we define $\Delta(\mathcal{A})$.

Definition 4.3.4. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS$(\mathcal{I}, \mathcal{O})$, we write $\Delta(\mathcal{A})$, to denote the system $\langle \mathcal{Q}, q^0, \mathcal{L}, \Delta(\mathcal{T}) \rangle$, where $\Delta(\mathcal{T})$ is the timed transition relation $\mathcal{T}$ extended with self-loop transitions labelled with δ_j in all O_j-quiescent states and with γ_k in all states that have channel I_k blocked.*

We lift all concepts and notations (i.e. ttraces, init, etc.) that have been defined for TLTS$(\mathcal{I}, \mathcal{O})$ to extend TLTS$(\mathcal{I}, \mathcal{O})$.

A direct consequence of this extension is that representing I_k as γ_k for all input channels and O_j as δ_j for all output channels, we have:

$$\mathsf{Fttraces}(\mathcal{A}) \cap (\mathcal{L} + \mathcal{I} + \mathcal{O})^* \quad = \quad \mathsf{ttraces}(\Delta(\mathcal{A}))$$

Therefore, using this notation we can rewrite the tmior conformance relation as:

$$i \leq_{tmior} S \quad \text{if and only if} \quad \mathsf{ttraces}(\Delta(i)) \subseteq \mathsf{ttraces}(\Delta(S))$$

Example 4.3.5. *Figure 4.4 illustrates the cash machine with the extended timed transition relation with refusals(γ_k) and quiescence(δ_j).*

4.3.3 Operational model

In a black box implementation we can not know if a state is quiescent or not, instead we can only observe the absence of output actions. To resolve the problem of the detection of quiescence in black box implementations we adopt a solution inspired by Chapter 3. There, to recognize quiescence, the implementations are required to be M-quiescent. An M-quiescent implementation is an implementation in which all output responses take place before M time units. In other words, M is the amount of time a tester has to wait for outputs before it can conclude that the system is in a quiescent state.

As mentioned earlier in TLMTS$(\mathcal{I}, \mathcal{O})$ we relax the M-quiescent requirement and allow each channel to have its particular (and possibly different) value for M. Therefore, the M-quiescent requirement can be seen as imposing the same bound for all channels.

We start by defining what it means for a state to be quiescent with respect to a channel and a particular time bound. Intuitively, the fact that a state is quiescent on a channel with respect to a particular bound means that all reachable states after the given bound delaying are quiescent on that particular channel. More precisely, an implementation's state q is considered M_j-quiescent, for an output channel j, if and only if all states reachable from q by delays $d \geq M_j$ are quiescent.

Definition 4.3.6. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLMTS$(\mathcal{I}, \mathcal{O})$ with $q, q' \in \mathcal{Q}$ and $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$ with $M_j \in \mathcal{D}$ for all $j = 1, \cdots, m$ be an ordered set of bounds, then*

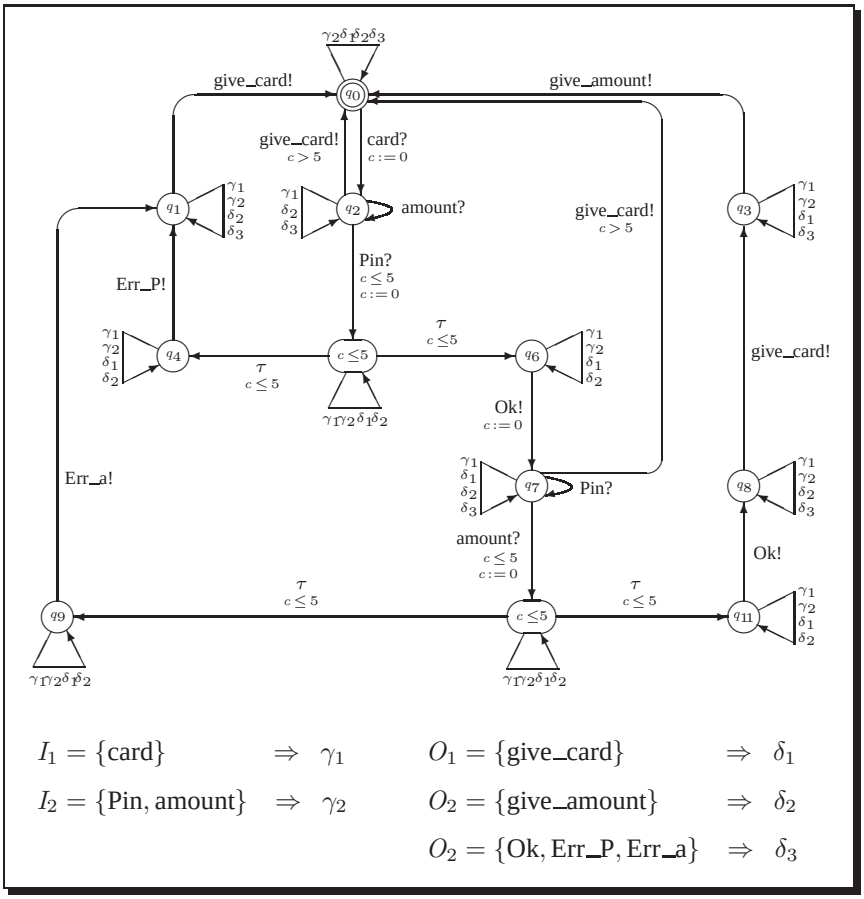


Figure 4.4: A timed cash machine with blocked channels and quiescence

$$
\begin{array}{lcl}
q \textit{ is } \mathcal{M}_j\textit{-quiescent} & \triangleq & \forall\ q' \in (q\ \mathbf{after}\ M_j) : \delta_j(q') \\
\mathcal{A} \textit{ is } \mathcal{M}_j\textit{-quiescent} & \triangleq & \forall\ q \in \mathcal{Q} : q \textit{ is } \mathcal{M}_j\textit{-quiescent} \\
\mathcal{A} \textit{ is } \mathcal{M}\textit{-quiescent} & \triangleq & \forall\ 1 \leq j \leq m : \forall\ q \in \mathcal{Q} : q \textit{ is } \mathcal{M}_j\textit{-quiescent}
\end{array}
$$

An interpretation of this definition is that for a tester to check for quiescence in channel j, it is enough to wait a period of time equal to M_j, for outputs to occur. Note that the set of bounds $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$ is a set with an order, that assign M_j to channel j.

There are two important principles involved in the previous definition. We are spending different times for detecting quiescence for different channels and we are assuming that after the corresponding delay there will not be any spontaneous output on that channel. The next corollary and proposition prove that Definition 4.3.6 is well defined.

Corollary 4.3.7. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a $\mathcal{M}_j$-quiescent system in TLMTS$(\mathcal{I}, \mathcal{O})$ with*

$q \in \mathcal{Q}$, *then*

$$\delta_j(q) \quad \textit{if and only if} \quad \forall\ l \in O_j : \forall\ d \in \mathcal{D} : d \leq M_j : q \overset{l(d)}{\not\Rightarrow}$$

Proof.

[$\Rightarrow$ If $\delta_j(q)$ then $\forall\ l \in O_j : \forall\ d \in \mathcal{D} : d \leq M_j : q \overset{l(d)}{\not\Rightarrow}$]
Let $\delta_j(q)$, then by Definition 4.3.3, of O_j-quiescent, we have that for all action $l \in O_j$ and for all $d \in \mathcal{D}$ such that $q \overset{l(d)}{\not\Rightarrow}$ then for all $l \in O_j$ and for all $d \in \mathcal{D}$ with $d \leq M_j$ we have $q \overset{l(d)}{\not\Rightarrow}$.

[$\Leftarrow$ If $\forall\ l \in O_j : \forall\ d \in \mathcal{D} : d \leq M_j : q \overset{l(d)}{\not\Rightarrow}$ then $\delta_j(q)$]
Because $\mathcal{M}_j$-quiescent($\mathcal{A}$) using Definition 4.3.6, of M_j-quiescent, we have that $\mathcal{M}_j$-quiescent(q), then again by Definition 4.3.6, $\forall\ q' \in (q\ \mathbf{after}\ M_j) : W$ $\delta_j(q')$. Now, because for all $l \in O_j$ and for all $d \in \mathcal{D}$ with $d \leq M_j$ then $q \overset{l(d)}{\not\Rightarrow}$, we have $\delta_j(q)$.

□

Proposition 4.3.8. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a $\mathcal{M}$-quiescent system in TLMTS($\mathcal{I}, \mathcal{O}$) with $q \in Q$, and $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$, then $\forall\ 1 \leq j \leq m$:*

$$\delta_j(q) \quad \textit{if and only if} \quad \forall\ l \in O_j : \forall\ d \in \mathcal{D} : d \leq M_j : q \overset{l(d)}{\not\Rightarrow}$$

Proof.

$\mathcal{M}$-quiescent($\mathcal{A}$)
$\Leftrightarrow$ {Definition 4.3.3}
$\forall\ 1 \leq j \leq m : \mathcal{M}_j$-quiescent($\mathcal{A}$)
$\Leftrightarrow$ {Corollary 4.3.7}
$\delta_j(q)$ if and only if $\forall\ l \in O_j : \forall\ d \in \mathcal{D} : d \leq M_j : q \overset{l(d)}{\not\Rightarrow}$

□

Example 4.3.9. *In the cash machine of Figure 4.4 for $\mathcal{M} = \{M_1, M_2, M_3\}$ with $M_1 = 6$, $M_2 = 5$ and $M_3 = 7$ we recognize that state q_0 is M_1-quiescent.*

4.3.4 The mtiorf conformance relation

Using the $\mathcal{M}$-quiescent property, we can detect quiescence in implementations by the absence of outputs in the stipulated period of time per channel. Therefore, requiring $\mathcal{M}$-quiescent input-enabled implementations we define the mtiorf conformance relation, a relation that is parameterized by $\mathcal{M}$.

Definition 4.3.10. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS$(\mathcal{I}, \mathcal{O})$ with $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$, $M_j \in \mathcal{D}$ for all $j = 1, \cdots, m$ be an ordered set of bounds, then*

$$\mathsf{ttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A}) \quad \triangleq \quad \mathsf{ttraces}(\Delta(\mathcal{A})) \cap \bigcup_k \bigcup_j (\mathcal{L} + \gamma_k + M_j \cdot \delta_j)^*$$

Definition 4.3.11. *Let i be a $\mathcal{M}$-quiescent input-enabled implementation in TLMTS$(\mathcal{I}, \mathcal{O})$ and S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$, then*

$$i \leq^{\mathcal{M}}_{mtiorf} S \quad \triangleq \quad \mathsf{ttraces}^{\Delta}_{\mathcal{M}}(i) \subseteq \mathsf{ttraces}^{\Delta}_{\mathcal{M}}(S)$$

The mtiorf relation considers ttraces extended with the information about blocked input channels and the quiescence in output channels. Moreover, the quiescence information on a channel can only appear after the time's period stipulated by the channel bounds. Then a δ_j can only occur after M_j time units.

4.4 The ultimate model and its conformance relation

All the relations that we considered up to now are built up on information based on behavioural knowledge of both specifications and implementations. In this section we define a relation that uses information (or knowledge) from the specification behaviour and only observations from the implementation behaviour. In this way, our approach became more desirable in the context of black box testing.

4.4.1 Normalized timed traces

Similarly as in Section 3.4 (from Chapter 3, page 40), we define a practical notation in the form of normalized timed traces. Normalized timed trace, denoted nttraces, are a subset of ttraces with a particular form; they do not have consecutive delays.

Definition 4.4.1. *A* normalized timed trace *σ is a trace such that*

$$\sigma \quad \in \quad \bigcup_k \bigcup_j (\mathcal{D} \cdot (L + \gamma_k + \delta_j))^* \cdot (\varepsilon + \mathcal{D})$$

Definition 4.4.2. *Let $\Delta(\mathcal{A}) = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T} \rangle$ be a TLTS$(\mathcal{I}, \mathcal{O})$, then*

$$\mathsf{nttraces}(\mathcal{A}) \quad \triangleq \quad \{\sigma \in \bigcup_k \bigcup_j (\mathcal{D} \cdot (L + \gamma_k + \delta_j))^* \cdot (\varepsilon + \mathcal{D}) \mid q^0 \stackrel{\sigma}{\Rightarrow}\}$$

for nttraces $\sigma = d_0 \cdot \delta_1 \cdot d_1 \cdot \gamma_1 \cdot d_2 \cdot a$ we also write $\sigma = \delta_1(d_0) \cdot \gamma_1(d_1) \cdot a(d_2)$.

Moreover, the definition of nttraces already assumes that given $\mathcal{A}$, a system in TLTS$(\mathcal{I}, \mathcal{O})$, has its timed transition relation extended with quiescence and refusals, implying that

$$\mathsf{nttraces}(\Delta(\mathcal{A})) \quad = \quad \mathsf{nttraces}(\mathcal{A})$$

Definition 4.4.2 and Definition 3.3.7 (from Chapter 3, page 36) have one similarity and one difference. The similarity is that in both definitions ttraces do not have consecutive delays. The difference is that now we assume the system to be already extended with quiescence and refusals.

Example 4.4.3. *In the cash machine of Figure 4.4, we observe the nttrace σ with $\sigma =$ card$(3)\cdot$Pin$(2)\cdot$Err-P$(5)\cdot\gamma_1(6)\cdot$give_card(0).*

For consistency, we need to show that by using nttraces we are not losing expressiveness. Hence, in Theorem 4.4.4, we prove that the inclusion of nttraces for two systems and inclusion of ttraces for the corresponding extended systems are equivalent. As in Chapter 3, given a ttrace σ we can always associate it a normalized one, denoted $\hat{\sigma}$, combining delays and adding 0 delay if they are necessary.

Theorem 4.4.4. *Let $\mathcal{A}_1, \mathcal{A}_2 \in$ TLTS$(\mathcal{I}, \mathcal{O})$, then*

$$\mathsf{ttraces}(\Delta(\mathcal{A}_1)) \subseteq \mathsf{ttraces}(\Delta(\mathcal{A}_2)) \quad \textit{if and only if} \quad \mathsf{nttraces}(\mathcal{A}_1) \subseteq \mathsf{nttraces}(\mathcal{A}_2)$$

Proof.

[$\Rightarrow$ If $\mathsf{ttraces}(\Delta(\mathcal{A}_1)) \subseteq \mathsf{ttraces}(\Delta(\mathcal{A}_2))$ then $\mathsf{nttraces}(\mathcal{A}_1) \subseteq \mathsf{nttraces}(\mathcal{A}_2)$]
Direct using Definition 4.4.2 of nttraces.

[$\Leftarrow$ If $\mathsf{nttraces}(\mathcal{A}_1) \subseteq \mathsf{nttraces}(\mathcal{A}_2)$ then $\mathsf{ttraces}(\Delta(\mathcal{A}_1)) \subseteq \mathsf{ttraces}(\Delta(\mathcal{A}_2))$]
If $\sigma \in \mathsf{ttraces}(\Delta(\mathcal{A}_1))$, and $l \in I_k$ or $l = \gamma_k$. Then
$\sigma\cdot l \in \mathsf{ttraces}(\mathcal{A}_1)$
$\Rightarrow$ {Definition 4.4.2, adding consecutive times and putting 0 times when is necessary}
$\widehat{\sigma\cdot l} \in \mathsf{nttraces}(\Delta(\mathcal{A}_1))$
$\Rightarrow$ {property ofnttraces}
$\widehat{\sigma\cdot l} \in \mathsf{nttraces}(\mathcal{A}_1)$
$\Rightarrow$ {hypothesis}
$\widehat{\sigma\cdot l} \in \mathsf{nttraces}(\mathcal{A}_2)$
$\Rightarrow$ {property ofnttraces}
$\widehat{\sigma\cdot l} \in \mathsf{nttraces}(\Delta(\mathcal{A}_2))$
$\Rightarrow$ {Definition 4.4.2 and turning back the procedure done before}
$\sigma\cdot l \in \mathsf{ttraces}(\Delta(\mathcal{A}_2))$
$\Rightarrow$ {density of $\mathcal{D}$}
$\sigma \in \mathsf{ttraces}(\Delta(\mathcal{A}_2))$

□

Using this result from now on we do not distinguish between a ttrace or its normalization. Moreover, we can extend Definition 4.3.10 to nttraces.

Definition 4.4.5. *An σ is an* $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ *if σ is a nttrace and σ is an element of*

$$\sigma \quad \in \quad \bigcup_k \bigcup_j (\mathcal{D}\cdot(L+\gamma_k) + M_j\cdot\delta_j)^*\cdot(\varepsilon+\mathcal{D})$$

Thus, $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ are nttraces, meaning they do not have consecutive delays actions and for each channel an occurrence of δ appears exactly after its particular bound.

Definition 4.4.6. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T}\rangle$ be a TLTS$(\mathcal{I},\mathcal{O})$ with $\mathcal{M} = \langle M_1, \cdots, M_m\rangle$, $M_j \in \mathcal{D}$ for all $j = 1, \cdots, m$ be an ordered set of bounds, then*

$$\mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A}) \quad = \quad \mathsf{nttraces}(\mathcal{A}) \cap \bigcup_k \bigcup_j (\mathcal{D}\cdot(L+\gamma_k) + M_j\cdot\delta_j)^*\cdot(\varepsilon+\mathcal{D})$$

4.4.2 Output set

The observable output set of a given set of states $\mathcal{Q}'$, denoted $\mathbf{out}_{\mathcal{M}}(\mathcal{Q}')$, is defined as the union of two sets: the set of output actions enriched with quiescence, denoted $\mathbf{out}^o_{\mathcal{M}}$, and the set of refusals, denoted $\mathbf{out}^r_{\mathcal{M}}$. Thus, $\mathbf{out}^o_{\mathcal{M}}$ is the set of outputs that could happen after a period of time plus the special symbol $\delta_j(M_j)$ expressing quiescence on output channel j in case a reachable state after M_j is quiescent on channel j. Further, the set $\mathbf{out}^r_{\mathcal{M}}$ is the set of refusals $\gamma_k(d)$ for each input channel k that is refused after d time units.

Definition 4.4.7. *Let $\Delta(\mathcal{A}) = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T}\rangle$ be TLTS$(\mathcal{I},\mathcal{O})$ with $\mathcal{Q}' \subseteq \mathcal{Q}$ and $\mathcal{M} = \langle M_1, \cdots, M_m\rangle$, $M_j \in \mathcal{D}$ for all $j = 1, \cdots, m$ be an ordered set of bounds, then*

$$\begin{array}{rlll} & \mathbf{out}_{\mathcal{M}}(\mathcal{Q}') & \triangleq & \bigcup\limits_{q\in\mathcal{Q}'} \mathbf{out}^o_{\mathcal{M}}(q) \;\cup\; \bigcup\limits_{q\in\mathcal{Q}'} \mathbf{out}^r_{\mathcal{M}}(q) \\ \textit{where} & \mathbf{out}^o_{\mathcal{M}}(q) & \triangleq & \{l(d) \mid l \in O \wedge q \stackrel{l(d)}{\Rightarrow}\} \cup \bigcup\limits_j \{\delta_j(M_j) \mid q \stackrel{\delta_j(M_j)}{\Rightarrow}\} \\ & \mathbf{out}^r_{\mathcal{M}}(q) & \triangleq & \bigcup\limits_k \{\gamma_k(d) \mid \forall\ l \in I_k :\ q \stackrel{l(d)}{\not\Rightarrow}\} \end{array}$$

An immediate and useful consequence of this definition is that a TLTS$(\mathcal{I},\mathcal{O})$ has an nttrace if and only if the observed output set, $\mathbf{out}_{\mathcal{M}}$, of the system after that nttrace is not empty.

Corollary 4.4.8. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T}\rangle$ in TLTS$(\mathcal{I},\mathcal{O})$ with $\sigma \in \mathsf{nttraces}$, then*

$$\mathbf{out}_{\mathcal{M}}(q^0 \textbf{ after } \sigma) = \emptyset \quad \textit{if and only if} \quad \sigma \notin \mathsf{nttraces}(q^0)$$

Proof.

[$\Leftarrow$ If $\mathbf{out}_{\mathcal{M}}(q^0 \textbf{ after } \sigma) = \emptyset$ then $\sigma \notin \mathsf{nttraces}(q^0)$]
By contradiction.
Suppose $O_j \in \mathcal{O}$ and $\sigma \in \mathsf{nttraces}(q^0)$, then $\exists\ q' : q^0 \stackrel{\sigma}{\Rightarrow} q'$, then

if $\exists\ l \in O_j : \exists\ d \in \mathcal{D} : q' \overset{l(d)}{\Rightarrow}$ then
$l(d) \in \mathbf{out}_{\mathcal{M}}(q^0\ \mathbf{after}\ \sigma)$
if $\forall\ l \in O_j : \forall\ d \in \mathcal{D} : q' \overset{l(d)}{\not\Rightarrow}$ then
$\delta_j(M_j) \in \mathbf{out}_{\mathcal{M}}(q^0\ \mathbf{after}\ \sigma)$. Contradiction, then if
$\mathbf{out}_{\mathcal{M}}(q^0\ \mathbf{after}\ \sigma) = \emptyset$ then $\sigma \notin \mathsf{nttraces}(q^0)$
$[\Rightarrow$ If $\sigma \notin \mathsf{nttraces}(q^0)$ then $\mathbf{out}_{\mathcal{M}}(q^0\ \mathbf{after}\ \sigma) = \emptyset]$
Direct from Definition 4.4.7 of $\mathbf{out}_M$ sets.

□

We also prove that the parameterized mtiorf relation is equivalent to checking the inclusion of observed output set for all nttraces that only have δ_j after M_j time units.

Proposition 4.4.9. *Let i be a $\mathcal{M}$-quiescent input-enabled implementation in TLMTS$(\mathcal{I}, \mathcal{O})$ and S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$. Then,*

$$i \leq^{\mathcal{M}}_{mtiorf} S \quad \textit{if and only if} \quad \forall\ \sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}\text{: } \mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma)$$

Proof.
$[\Rightarrow$ If $i \leq^{\mathcal{M}}_{mtiorf} S$ then $\forall\ \sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}\text{: } \mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma)]$
Let $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}$, then
if $\sigma \notin \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i)$, then
$\mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma) = \emptyset$
if $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i)$, then
$\forall\ l(d) \in \mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma)$
$\Rightarrow$ {Definition 4.4.7}
$\sigma \cdot l(d) \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i)$
$\Rightarrow$ {hypothesis}
$\sigma \cdot l(d) \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$
$\Rightarrow$ {Definition 4.4.7}
$l(d) \in \mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma)$
$[\Leftarrow$ If $\forall\ \sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}\text{: } \mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma)$ then $i \leq^{\mathcal{M}}_{mtiorf} S]$
Let $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i) : \sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i) \cap \mathsf{nttraces}^{\Delta}_{\mathcal{M}}$then, $(\Delta(i)\ \mathbf{after}\ \sigma) \neq \emptyset$
let $I_k \in \mathcal{I}$, then
$(\exists\ a \in I_k : l = a(d) \wedge \exists\ q : q \in (\Delta(i)\ \mathbf{after}\ \sigma) \wedge q \overset{l}{\Rightarrow}) \vee$
$(l = \gamma_k(d) \wedge \exists\ q : q \in (\Delta(i)\ \mathbf{after}\ \sigma) \wedge q \overset{l}{\Rightarrow})$
$l \in \mathbf{out}_{\mathcal{M}}(i\ \mathbf{after}\ \sigma)$
$\Rightarrow$ {hypothesis}
$l \in \mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma)$
$\Rightarrow$
$\mathbf{out}_{\mathcal{M}}(S\ \mathbf{after}\ \sigma) \neq \emptyset$
$\Rightarrow$ {Corollary 4.4.8}
$\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$

□

4.4.3 The mtioco$_{\mathcal{M}}$ implementation relation

We are ready to define the **mtioco**$_{\mathcal{M}}$ relation, based solely on information from the observed output set and the specification. Crucially, the definition does not rely on any internal knowledge of the implementation, which complies with out requirement of black box testing.

For i a $\mathcal{M}$-quiescent input-enabled implementation in TLMTS$(\mathcal{I},\mathcal{O})$ and S a specification in TLTS$(\mathcal{I},\mathcal{O})$ then i is **mtioco**$_{\mathcal{M}}$ correct with respect to S if and only if the observable output set of i, after every $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ of S is a subset of the observable output set of S after the same nttrace.

Definition 4.4.10. *Let i be a $\mathcal{M}$-quiescent input-enabled implementation in TLMTS$(\mathcal{I},\mathcal{O})$ and S be a specification in TLTS$(\mathcal{I},\mathcal{O})$, then*

$$i \ \mathbf{mtioco}_{\mathcal{M}} \ S \quad \triangleq \quad \forall\ \sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S) : \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after}\ \sigma) \subseteq \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after}\ \sigma)$$

The **mtioco**$_{\mathcal{M}}$ relation is a parameterized timed relation (with parameter $\mathcal{M}$) that considers quiescence for each particular channel. We use this relation to build our test derivation framework over TLTS$(\mathcal{I},\mathcal{O})$.

4.5 Multi timed test generation framework

In this section we define the concept of time multi test cases; further, we detail the nature of their execution and the evaluation of their verdict: pass or failure.

A *timed multi test case* t is a deterministic TLTS$(\mathcal{I},\mathcal{O})$ with actions in $\mathcal{L}_{\delta\gamma}$ (where $\mathcal{L}_{\delta\gamma} \triangleq \mathcal{L} \cup \{\delta_1,\ldots,\delta_n,\gamma_1,\ldots,\gamma_m\}$) such that it has bounded behaviour, i.e. all computations have finitely many action occurrences. The set of states in a test contains the terminal states **pass** and **fail** without outgoing transitions, except self loops which let the time pass. For any state different from **pass** and **fail**, there exists a bounded time to observe quiescence in a channel or for being able to perform an input action. Moreover, since tests under consideration are deterministic, τ-transitions are not allowed. A *timed multi test suite* $\mathbf{T}$ is a set of test cases. Similarly as done in Chapter 3, in order to simplify notation we represent tests as TA. The difference between multi timed test cases and timed tests cases as defined in Chapter 3 (in Section 3.6, page 45) is precisely the channel and blocking treatments. Now a test is able to recognize not only quiescence per channels but also a blocked channel.

Definition 4.5.1. *Given $\mathcal{M} = \langle M_1,\cdots,M_m\rangle$ with $M_j \in \mathcal{D}$ for all $j = 1,\cdots,m$ an ordered set of bounds to recognize quiescence, then*

- *A* timed multi test case $t = \langle \mathcal{Q}, q_0, \mathcal{L}_{\delta\gamma}, \mathcal{T}\rangle$ *is a TLTS$(\mathcal{I},\mathcal{O})$ such that*
  - *t is deterministic and has* bounded behaviour, *i.e.*

    $$\exists\ N > 0 : \forall\ \sigma \in \mathsf{ttraces}(t) : \sigma = l_1 l_2 l_3 \cdots : |\{z \mid l_z \in L\}| < \infty$$

  - *$\mathcal{Q}$ contains the terminal states* **pass** *and* **fail**, *with* $\mathsf{init}(\mathbf{pass})$ *and* $\mathsf{init}(\mathbf{fail})$ *without outgoing transitions expect self loops allowing time to pass*

– *for any state* $q \in \mathcal{Q}$ *of the test case with* $q \neq$ **pass**, **fail**, *there exists* d *in* $0 \leq d \leq max\langle M_1, \cdots, M_m \rangle$ *with*

$$\begin{array}{lcl} \mathsf{init}(q \textbf{ after } d') & = & O \cup \{e \mid e = d - d'\} \text{ for all } d' < d, \text{ or} \\ \mathsf{init}(q \textbf{ after } d) & = & \{l\} \text{ with } l \in I \text{ or } l = \delta_j \text{ or } l = \gamma_k \\ & & \text{for all } 1 \leq j \leq m \text{ and } 1 \leq k \leq n \end{array}$$

– t *does not have* τ*-transitions*

The class of test cases over $\mathcal{I}$ *and* $\mathcal{O}$ *is denoted by* $\mathcal{MTEST}(\mathcal{I}, \mathcal{O})$

- *A* multi timed test suite $\mathbf{T}$ *is a set of test cases:* $\mathbf{T} \subseteq \mathcal{MTEST}(\mathcal{I}, \mathcal{O})$

For the description of test cases, as before, we use a process algebraic behavioural notation: $B \triangleq l;\ B \mid B + B \mid \Sigma\, \mathcal{B}$. Where $l \in \mathcal{L}_{\delta\gamma}$, $\mathcal{B}$ is a countable set of behaviour expressions, and the axioms and the inference rules are:

$$\begin{array}{lcl} l \in L & \vdash & l;\ B \xrightarrow{l} B' \\ l = d, d' < d & \vdash & d;\ B \xrightarrow{d'} d - d';\ B \\ l = d & \vdash & l;\ B \xrightarrow{d} B' \\ B_1 \xrightarrow{l} B_1', l \in \mathcal{L}_{\delta\gamma} & \vdash & B_1 + B_2 \xrightarrow{l} B_1' \\ B_2 \xrightarrow{l} B_2', l \in \mathcal{L}_{\delta\gamma} & \vdash & B_1 + B_2 \xrightarrow{l} B_2' \\ B \xrightarrow{l} B', B \in \mathcal{B}, l \in \mathcal{L}_{\delta\gamma} & \vdash & \Sigma\, \mathcal{B} \xrightarrow{l} B' \end{array}$$

A test run of an implementation with a test case is modelled by the synchronous parallel execution of the test case together with the implementation under test. This run continues until no more interactions are possible, except letting the time pass.

We reuse the definition of run, passes and fails from Chapter 3 (from Definition 3.6.2, page 46), but we have to adapt the notion of test composition, this is done in the next definition.

Definition 4.5.2. *Let* i *be a* $\mathcal{M}$*-quiescent input-enabled implementation in TLMTS*$(\mathcal{I}, \mathcal{O})$*,* t *be a test in* $\mathcal{MTEST}(\mathcal{I}, \mathcal{O})$ *and* $\mathbf{T}$ *be a test suite included in* $\mathcal{MTEST}(\mathcal{I}, \mathcal{O})$ *, then*

- *Running* t *with* i *is modelled by the parallel operator*

$$||\quad :\quad \mathcal{MTEST}(\mathcal{I}, \mathcal{O}) \times \mathit{TLMTS(I, O)} \rightarrow \mathit{TLTS(I, O)}$$

which is defined by the following inference rules. For all $1 \leq j \leq m$ *and* $1 \leq k \leq n$*:*

$$\begin{array}{lcl} i \xrightarrow{\tau} i' & \vdash & t \mid\mid i \xrightarrow{\tau} t \mid\mid i' \\ t \xrightarrow{\delta_j} t', \forall\ l \in O_j : i \overset{l}{\not\rightarrow} & \vdash & t \mid\mid i \xrightarrow{\delta_j} t' \mid\mid i \\ t \xrightarrow{\gamma_k} t', i \overset{l}{\not\longrightarrow} i', l \in I_k & \vdash & t \mid\mid i \xrightarrow{\gamma_k} t' \mid\mid i \\ t \xrightarrow{l} t', i \xrightarrow{l} i', l \in L & \vdash & t \mid\mid i \xrightarrow{l} t' \mid\mid i' \\ t \xrightarrow{d} t', i \xrightarrow{d} i', d \in \mathcal{D} & \vdash & t \mid\mid i \xrightarrow{d} t' \mid\mid i' \end{array}$$

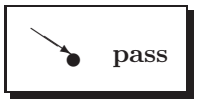
4.5.1 Test generation procedure

We define a procedure to generate test cases from a given extended specification $\Delta(S)$ in TLTS$(\mathcal{I}, \mathcal{O})$, we call this test generation procedure as MTGP. Again (as in before chapters) test cases result from the nondeterministic, recursive application of three test generation steps:

1. *termination*,
2. *inputs and blocking*, and
3. *observation of outputs* (including quiescence).

The set of $\mathcal{Q}' \in \mathcal{Q}$ represents the set of all possible states in which the specification can be at the current stage of the test case execution. Initially $\mathcal{Q}'$ is equal to $\{q^0\}$.

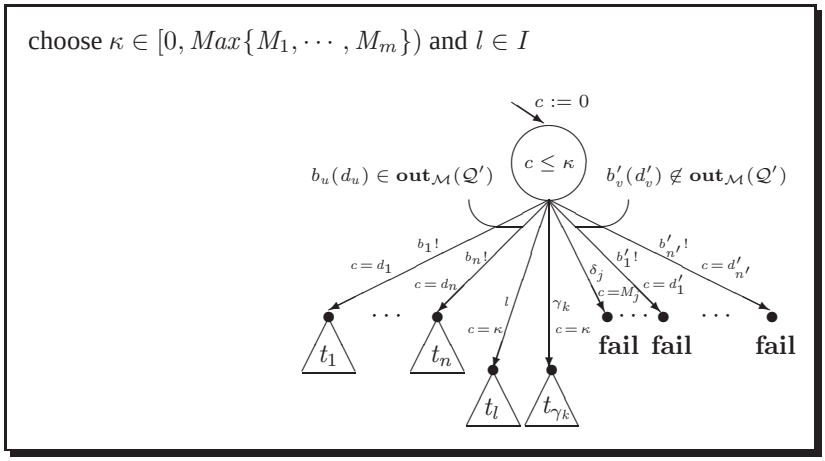
1. *termination*



t := **pass**

The single state test case **pass**. It is possible to stop the recursion at any time using this step.

2. *inputs and blocking*

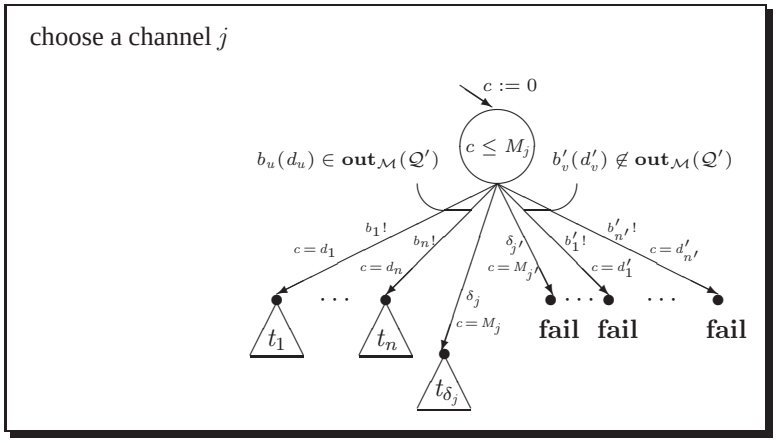


$$
\begin{array}{rlll}
t := & \Sigma\{b_u(d_u);\ t_u & \mid & b_u \in O \wedge d_u < \kappa \wedge b_u(d_u) \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \{l(\kappa);\ t_l & \mid & l \in I_k \wedge \exists\ q \in \mathcal{Q}' : \gamma_k(\kappa) \notin \mathbf{out}_{\mathcal{M}}(q)\} \\
+ & \{l(\kappa);\ \mathbf{fail} & \mid & l \in I_k \wedge \forall\ q \in \mathcal{Q}' : \gamma_k(\kappa) \in \mathbf{out}_{\mathcal{M}}(q)\} \\
+ & \{\gamma_k(\kappa);\ t_{\gamma_k} & \mid & \gamma_k(\kappa) \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \{\gamma_k(\kappa);\ \mathbf{fail} & \mid & \gamma_k(\kappa) \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \Sigma\{\delta_j(M_j);\ \mathbf{fail} & \mid & M_j \in \mathcal{M} \wedge M_j < \kappa \wedge \delta_j(M_j) \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \Sigma\{b'_v(d'_v);\ \mathbf{fail} & \mid & b'_v \in O \wedge b'_v(d'_v) \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\}
\end{array}
$$

where c is a clock, κ is a timed constant, M_j is the bound to detect quiescence in channel j, u is in $[1, n]$, v is in $[1, n']$ and t_u, t_l and t_{γ_k} are obtained by recursively applying the algorithm to $(\mathcal{Q}'$ **after** $b_u(d_u))$, $(\mathcal{Q}'$ **after** $l(\kappa))$ and $(\mathcal{Q}'$ **after** $\gamma_k(M_k))$, respectively.

The test case t is waiting for κ time units, and trying to perform an input (l) or to observe a blocking channel (γ_k). If an output arrives from the implementation, the test checks whether the output is an invalid response, i.e. $b'_v(d'_v) \notin \mathbf{out}_M(\mathcal{Q}')$; in that case, the test case terminates in **fail**. If the output is a valid response after the time passed, then the test case continues recursively. If the intended time pass (i.e. $c = \kappa$) then the test produces the input (l) or it observe the blocking channel (γ_k), and continues recursively.

3. *observation of outputs*



$$
\begin{array}{rlll}
t := & \Sigma\{b_u(d_u);\ t_u & \mid & b_u \in O \wedge d_u < M_j \wedge b_u(d_u) \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \Sigma\{\delta_j(M_j);\ t_{\delta_j} & \mid & \delta_j \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}' \textbf{ after } M_j)\} \\
+ & \Sigma\{\delta_j(M_j);\ \textbf{fail} & \mid & \delta_j \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}' \textbf{ after } M_j)\} \\
+ & \Sigma\{\delta_{j'}(M_{j'});\ \textbf{fail} & \mid & M_{j'} \in \mathcal{M} \wedge M_{j'} < M_j \wedge \delta_{j'}(M_{j'}) \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\} \\
+ & \Sigma\{b'_v(d'_v);\ \textbf{fail} & \mid & b'_v \in O \wedge b'_v(d'_v) \notin \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')\}
\end{array}
$$

where c is a clock, κ is a timed constant, M_j and $M_{j'}$ are the bounds to detect quiescence in channel j and j' respectively; u is in $[1, n]$, v is in $[1, n']$ and t_u and t_{δ_j} are obtained by recursively applying the algorithm to $(\mathcal{Q}'$ **after** $b_u(d_u))$ and $(\mathcal{Q}'$ **after** $\delta_j(M_j))$, respectively.

The test case t is waiting for M_j time units; if an output arrives from the implementation it checks whether it is an invalid response, i.e. $b'_u(d'_u) \notin \mathbf{out}_M(\mathcal{Q}')$; in that case, the test case terminates in **fail**. If it is a valid response, the test case continues with test case t_u, generated from $(Q'$ **after** $b'_u(d'_u))$. The observation of quiescence in channel j (δ_j) is treated separately, using the constant M_j. Moreover, the test reports a failure in case a channel with a smaller bound ($M_{j'} < M_j$) shows incorrectly to be quiescent.

We use case 1 to stop the recursion and bound the behaviour of the test.

Case 2 is used for two purposes: either try to test the acceptance of an input or try to test the blocking of an input channel; both trials are done in a particular time (κ). Since we need to wait until this particular time happens, we use this waiting period for observations. There are two kinds of observations: outputs observations and quiescent observations. For the former, if we observe an output we check its correctness. For the latter, if we observe quiescence in a channel whose quiescent observation bound is smaller than our waiting period, then we check its incorrectness.

There are two possible approaches to check the blocking of a channel (γ_k). Firstly, we can repeat a test for each input in channel k. Secondly, using the input-enabled property assumed on implementations, we can try to apply any input from channel k and if it is not accepted we conclude that the channel is blocked.

We use case 3 for the observation of outputs or quiescence. In case we want to observe quiescence in channel j, using the M_j-quiescent property, it is enough to wait until M_j to recognize it. If no output comes from that channel we conclude that it is quiescent. On the other hand, if we want to observe an output from channel j, again using the M_j-quiescent property, it is enough to wait for M_j time units. If no output actions, from channel j, appears again we conclude quiescence in channel j. If an output appears we check it correctness.

It is possible to recognize that case 3 has an overlapping with the waiting period from case 2, as follows. Suppose we derive a test using case 2 and we find out, during the derivation procedure, that there exists an arrow labelled with δ_j. This means that the bound M_j from channel j is smaller that κ (the intended time to apply the input) and moreover channel j should not be quiescent. Thus, if an implementation produces an output before κ, the intended input will not be applied. Besides, if the implementation does not produce any output from channel j before M_j, we must have found an error and the intended input will not be applied neither. Hence, if an arrow lebelled with δ_j appears deriving case 2 then we know that the intended input will not be applied. Fortunately, we can use this knowledge. Once it is known that there exists an arrow for δ_j deriving case 2, we could suggest to choose case 3 and wait for outputs on channel j. But, because the recognition of this knowledge could take time, our proposed improvement may only be used on batch derivation test (that is, tests that are derived before they are applied).

As a final remark, the construction steps involve negations of predicates of the form $b(d) \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')$ or $\gamma_k(d) \in \mathbf{out}_{\mathcal{M}}(\mathcal{Q}')$; which in general for TLTS$(\mathcal{I}, \mathcal{O})$ are undecidable. Hence, the procedure given here can be seen as a meta-algorithm which is useful for generating tests effectively for the subclasses of TLTS$(\mathcal{I}, \mathcal{O})$ in which these predicates are decidable; for example TA [38, 39], with sub-partitioning of the input and output sets.

Example 4.5.3. *Figure 4.5 shows a test for the cash machine. The test checks that it is not possible to ask for money before a card is authenticated. This is done using case* 2 *for* γ_2*. Then, a card and a Pin are inserted using case* 2 *for the inputs card and Pin, respectively. Subsequently, using case* 3 *we expect an answer from the machine; if the Pin is correct, we then ask for an amount of money, using case* 2 *with input amount. We expect the answer from the machine using case* 3*; if the answer is positive we wait for the card and the amount of money. We finally end the recursive procedure of the test with case* 1*.*

We suppose that this test will be used with an implementation that we know is $\mathcal{M}$*-quiescent with* $\mathcal{M} = \{M_1, M_2, M_3\}$ *and* $M_1 > 3$*,* $M_2 > 4$ *and* $M_3 > 5$*.*

4.6 Completeness

In this section we show that the test generation framework (MTGP) presented in Section 4.5.1 is complete with respect to the **mtioco**$_{\mathcal{M}}$ implementation relation. We first recall when an implementation is sound and exhaustive with respect to the **mtioco**$_{\mathcal{M}}$.

A set of tests, generated by the MTGP, is sound with respect to **mtioco**$_{\mathcal{M}}$ if for any implementation that fails a test in the set, the implementation is incorrect with respect to **mtioco**$_{\mathcal{M}}$. Furthermore, a set of tests is exhaustive with respect to **mtioco**$_{\mathcal{M}}$ if for every incorrect implementation a test case can be generated, following the MTGP, that detects the non-conformance.

Definition 4.6.1. *Let S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$. Then for all i a $\mathcal{M}$-quiescent input-enabled implementation in TLMTS$(\mathcal{I}, \mathcal{O})$ and for $\mathbf{T}$ the test set of all test cases obtained from S by the MTGP*

$$\begin{array}{lcl} \mathbf{T} \textit{ is sound w.r.t. to } \textbf{mtioco}_{\mathcal{M}} & \triangleq & \textit{if } \forall\ t \in \mathbf{T} : i\ \textbf{mtioco}_{\mathcal{M}}\ S \textit{ then } i\ \textbf{passes}\ t \\ \mathbf{T} \textit{ is exhaustive w.r.t. to } \textbf{mtioco}_{\mathcal{M}} & \triangleq & \textit{if } i\ \not\textbf{mtioco}_{\mathcal{M}}\ S \textit{ then } \exists\ t \in \mathbf{T} : t\ \not\textbf{passes}\ i \end{array}$$

In the proofs of soundness and exhaustiveness we use the term *saturation* to refer to saturations of δ's, in $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$. The following definition gives a relation between $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ with δ in a particular location and a similar one, without δ in that location.

Definition 4.6.2. *Let $\mathcal{A}$ be a system in TLTS$(\mathcal{I}, \mathcal{O})$ with $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$, $M_j \in \mathcal{D}$ for all $j = 1, \cdots, m$ be an ordered set of bounds and $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$, then*

$$\begin{array}{lcl} \sigma \textit{ is } \delta(\mathcal{M})\textit{-saturated} & \triangleq & \nexists\, \sigma' \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A}) : \exists\, M_j \in \mathcal{M} : \sigma = \sigma_1 \cdot l(d) \cdot \sigma_2 \wedge \\ & & \qquad l \neq \delta_j \wedge \sigma' = \sigma_1 \cdot \delta_j(M_j) \cdot l(d - M_j) \cdot \sigma_2 \end{array}$$

A $\delta(\mathcal{M})$-saturated $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ is a $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ that does not permit an action to come after $\max\{M_1, \cdots, M_m\}$ without observing quiescence.

4.6.1 Soundness

The MTGP presented is sound with respect to **mtioco**$_{\mathcal{M}}$ testing relation.

Theorem 4.6.3. *Let S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$, then for all i a $\mathcal{M}$-quiescent implementations in TLMTS$(\mathcal{I}, \mathcal{O})$ and for all t a test cases obtained from S by the MTGP*

$$\textit{if} \quad i\ \textbf{mtioco}_{\mathcal{M}}\ S \quad \textit{then} \quad i\ \textbf{passes}\ t$$

Proof.
Let i be $\mathcal{M}$-quiescent with $(i\ \textbf{mtioco}_{\mathcal{M}}\ S)$, then we show that for all $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$ and for all t test cases generated from S by the MTGP the following holds

$$\text{if} \quad t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \quad \text{then} \quad t' \neq \mathbf{fail}$$

Without loss of generality we can assume that σ is $\delta(\mathcal{M})$-saturated. We prove the theorem by induction over the length of σ

- If $\sigma = \epsilon$ and $t \,||\, i \stackrel{\epsilon}{\Rightarrow} t' \,||\, i'$
  if t was constructed using case 1 in the first step, then
  $t \,||\, i \stackrel{\epsilon}{\Rightarrow} \mathbf{pass} \,||\, i'$
  if t was constructed using cases 2 or 3 in the first step, then
  $t = t' \neq \mathbf{fail}$ and all derivations of $\stackrel{\epsilon}{\Rightarrow}$ have the form:
  $t \,||\, i \stackrel{\epsilon}{\Rightarrow} t \,||\, i'$
- If $\sigma = \sigma' \cdot a$ and $t \,||\, i \stackrel{\sigma'}{\Rightarrow} t'' \,||\, i'' \stackrel{a}{\Rightarrow} t' \,||\, i' \wedge a = l(d)$,
  because t can do a there are only two possibilities to construct t':
  from case 2: $(l \in I_k)$ or $(l = \gamma_k) \wedge d < Max\{M_1, \cdots, M_m\}$, then
  because $(i \ \mathbf{mtioco}_{\mathcal{M}} \ S)$
  if $l \in I_k$ then $\gamma_k(d) \notin \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ \sigma')$, then
  $t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \wedge \ t' \neq \mathbf{fail}$
  if $l = \gamma_k$ then $\gamma_k(d) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ \sigma')$, then
  $t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \wedge \ t' \neq \mathbf{fail}$
  from case 3 for j: $(l \in O_j)$ or $(l = \delta_j) \wedge \ a \in \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after} \ \sigma')$, then
  because $(i \ \mathbf{mtioco}_{\mathcal{M}} \ S)$: $l(d) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ \sigma')$, and thus
  $t \,||\, i \stackrel{\sigma}{\Rightarrow} t' \,||\, i' \wedge \ t' \neq \mathbf{fail}$.

□

4.6.2 Exhaustiveness

The MTGP is also exhaustive, in the sense that for each non-conforming implementation, a test case can be generated that detects the non-conformance.

Before proving the theorem of exhaustiveness, we establish a useful property. For every specification S in TLTS$(\mathcal{I}, \mathcal{O})$ and every $\delta(\mathcal{M})$-saturated $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$ such that there exists a test case t' for (S after σ), then there also exists a test case t such that from t doing σ it is possible to obtain t'. This property is reflected in the next lemma.

Lemma 4.6.4. *Let S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$, $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$ be $\delta(\mathcal{M})$-saturated, and t' be a test case generated by the MTGP for (S $\mathbf{after}$ σ). Then, there exists t a test case generated from S with $t \stackrel{\sigma}{\Rightarrow} t'$.*

Proof.
By induction over the length of σ:

- Let $|\sigma| = 0$ then take $t = t'$
- Suppose there exists t for all σ with $|\sigma| < n$
- Let $|\sigma| = n$ with $\sigma = \sigma' \cdot a$ and $a = l(d)$
  if $(l \in I$ or $l = \gamma_k)$ using case 2 for the input $l : t \stackrel{\sigma'}{\Rightarrow} t'' \stackrel{a}{\Rightarrow} t'$
  if $a \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ \sigma)$, using case 3 for channel j $(l \in O_j) : t \stackrel{\sigma'}{\Rightarrow} t'' \stackrel{a}{\Rightarrow} t'$.

□

Theorem 4.6.5. *Let S be a specification in TLTS$(\mathcal{I},\mathcal{O})$. Then for every i a $\mathcal{M}$-quiescent implementation in TLMTS$(\mathcal{I},\mathcal{O})$ with i* **mtióco**$_{\mathcal{M}}$ *S, there exists t a test case generated by the MTGP from S such that*

$$i \quad \mathbf{pa\not{s}ses} \quad t$$

Proof.
If i **mtióco**$_{\mathcal{M}}$ S then there exists $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(S)$ such that

$$\mathbf{out}_{\mathcal{M}}(i \mathbf{\ after\ } \sigma) \quad \not\subseteq \quad \mathbf{out}_{\mathcal{M}}(S \mathbf{\ after\ } \sigma)$$

Without loss of generality, we can assume that σ is $\delta_{\mathcal{M}}$-saturated. Then, let $a = l(d)$ such that $a \in \mathbf{out}_{\mathcal{M}}(i \mathbf{\ after\ } \sigma) \backslash \mathbf{out}_{\mathcal{M}}(S \mathbf{\ after\ } \sigma)$ and $i \stackrel{\sigma}{\Rightarrow} i' \stackrel{a}{\Rightarrow} i''$.

- If $l \in O_j$ then let t' be the result of applying case 3 for j of the procedure to $(\Delta(S)$ **after** $\sigma)$, and let t be the test case constructed out of t' and σ by Lemma 4.6.4. Because $a \notin \mathbf{out}_{\mathcal{M}}(S \mathbf{\ after\ } \sigma)$ then $(t \parallel i \stackrel{\sigma \cdot a}{\Longrightarrow} \mathbf{fail} \parallel i'')$, so i $\mathbf{pa\not{s}ses}$ t
- If $l = \gamma_k$ then let t' be the result of applying case 2 for γ_k of the procedure to $(\Delta(S)$ **after** $\sigma)$, and let t be the test case constructed out of t' and σ by Lemma 4.6.4. Because $a \notin \mathbf{out}_{\mathcal{M}}(S \mathbf{\ after\ } \sigma)$ then $(t \parallel i \stackrel{\sigma \cdot a}{\Longrightarrow} \mathbf{fail} \parallel i'')$, so i $\mathbf{pa\not{s}ses}$ t.

□

The exhaustiveness of our test generation procedure, similarly as in Chapter 3, is less useful than the corresponding result in the untimed case. There, the repeated execution of the test generation algorithm in a fair, nondeterministic manner, will generate for every error a test exposing it in finite time. This is not feasible for the real-time case, since the number of potential test cases is uncountable due to the underlying continuous time domain. It is possible to recover such limit-completeness by considering suitable equivalent classes of errors (i.e., an implementation has either all or no errors of a given class), such that a repeated test generation procedure will automatically expose an error in every equivalence class.

4.7 Relation with tioco$_M$

In this section we present the relation between the **mtioco**$_{\mathcal{M}}$ and the **tioco**$_M$ testing relations. Note that even that the **mtioco**$_{\mathcal{M}}$ relation is defined over TLMTS$(\mathcal{I},\mathcal{O})$ implementations and TLTS$(\mathcal{I},\mathcal{O})$ specifications, because TLMTS$(\mathcal{I},\mathcal{O})$ and TLTS$(\mathcal{I},\mathcal{O})$ are TLTS(I, O) (where $I = \bigcup_{1 \le k \le n} I_k$ and $O = \bigcup_{1 \le j \le m} O_j$) we can define **tioco**$_M$ over them.

To prove this result we use $\mathsf{nttraces}^{\Delta}_{M}$ as defined in Definition 3.4.2 (from Chapter 3, page 40) and $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ as defined in Definition 4.4.6 (from Chapter 4, page 71). Moreover, as follows we define how to transform a $\mathsf{nttraces}^{\Delta}_{M}$ to a $\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$.

Definition 4.7.1. *Let $\mathcal{A}$ be a TLTS$(\mathcal{I},\mathcal{O})$ with $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$ and $M = \max\langle M_1, \cdots, M_m \rangle$, we define the function $()_{\mathcal{M}} : \mathsf{nttraces}^{\Delta}_{M} \to \mathsf{nttraces}^{\Delta}_{\mathcal{M}}$ as*

$$
\begin{array}{lcl}
(\epsilon)_{\mathcal{M}} & \triangleq & \epsilon \\
(\sigma\cdot l(d))_{\mathcal{M}} & \triangleq & \left\{ \begin{array}{ll} (\sigma)_{\mathcal{M}}\cdot l(d) & l \neq \delta \\ (\sigma)_{\mathcal{M}}\cdot \delta_z(M_z) & l = \delta \wedge M_z = \max\langle M_1, \cdots, M_m\rangle \end{array} \right.
\end{array}
$$

Then the only difference between $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(\mathcal{A})$ and $(\sigma)_{\mathcal{M}}$ is in the δ-actions.

Lemma 4.7.2. *Let $\mathcal{A} = \langle \mathcal{Q}, q^0, \mathcal{L}, \mathcal{T}\rangle$ be a TLTS$(\mathcal{I}, \mathcal{O})$ with $\mathcal{M} = \langle M_1, \cdots, M_m\rangle$ and $M = \max\langle M_1, \cdots, M_m\rangle$, then*

$$\textit{if} \quad \sigma \in \mathsf{nttraces}^{\Delta}_{M}(\mathcal{A}) \quad \textit{then} \quad (\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$$

Proof.
We prove this lemma by induction over the length of a σ in $\mathsf{nttraces}^{\Delta}_{M}(\mathcal{A})$.

- Let $\sigma = l(d)$ then if $l \neq \delta$ then $\sigma = (\sigma)_{\mathcal{M}}$ and immediately $\sigma \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$. If $l = \delta$ then $d = M$ and there exists $q \in \mathcal{Q}$ such that $q^0 \stackrel{M}{\Rightarrow} q$ and q is M-quiescent. Because $M = \max\langle M_1, \cdots, M_m\rangle$, there exists $M_z \in \langle M_1, \cdots, M_m\rangle$ such that $M_z = M$ and q is $\mathcal{M}_z$-quiescent. So $\delta_z(M_z) \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$ and then since $(\sigma)_{\mathcal{M}} = \delta_z(M_z)$ we have $(\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$.
- Suppose that for all $|\sigma| < n$ with $\sigma \in \mathsf{nttraces}^{\Delta}_{M}(\mathcal{A})$ then $(\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$.
- Let $|\sigma| = n$ then there exists $\sigma' : \sigma = \sigma'\cdot l(d)$. If $l \neq \delta$ then using hypothesis inductive is direct that $(\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$. If $l = \delta$ then $d = M$, so there exists $q \in \mathcal{Q}$ such that $q^0 \stackrel{\sigma'.M}{\Longrightarrow} q$ and q is M-quiescent. But since $M = \max\langle M_1, \cdots, M_m\rangle$ also there exists $M_z \in \langle M_1, \cdots, M_m\rangle$ such that $M_z = M$ and q is $\mathcal{M}_z$-quiescent. So using hypothesis inductive $(\sigma')_{\mathcal{M}}\cdot\delta_z(M_z) \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$ and then since $(\sigma)_{\mathcal{M}} = (\sigma')_{\mathcal{M}}\cdot\delta_z(M_z)$ we have $(\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(\mathcal{A})$.

□

Theorem 4.7.3. *Let S be a specification in TLTS$(\mathcal{I}, \mathcal{O})$, i be an input-enabled $\mathcal{M}$-quiescent implementation in TMLTS(I, O) and let $M = \max\langle M_1, \cdots, M_m\rangle$ then*

$$\textit{if} \quad i \ \mathbf{mtioco}_{\mathcal{M}}\ S \quad \textit{then} \quad i \ \mathbf{tioco}_M S$$

Proof.
We prove the theorem proving that for all σ in $\mathsf{nttraces}^{\Delta}_{M}(S)$

$$\text{if} \quad l(d) \in \mathbf{out}_M(i \ \mathbf{after}\ \sigma) \quad \text{then} \quad l(d) \in \mathbf{out}_M(S \ \mathbf{after}\ \sigma)$$

We prove it by induction over the length of σ, then

- Let $\sigma = \epsilon$ and $l(d) \in \mathbf{out}_M(i \ \mathbf{after}\ \epsilon)$, then if $l \neq \delta$ then $l(d) \in \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after}\ \epsilon)$. So, because i **mtioco**$_{\mathcal{M}}$ S we know that $l(d) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after}\ \epsilon)$, then $l(d) \in \mathbf{out}_M(S \ \mathbf{after}\ \epsilon)$. If $l = \delta$ then because $M = \max\langle M_1, \cdots, M_m\rangle$ we know that there exists z such that $\delta_z(M_z) \in \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after}\ \epsilon)$. So using that i **mtioco**$_{\mathcal{M}}$ S we know that $\delta_z(M_z) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after}\ \epsilon)$. But, again since $M = \max\langle M_1, \cdots, M_m\rangle$ we have $\delta(M) \in \mathbf{out}_M(S \ \mathbf{after}\ \epsilon)$.

- Suppose that for all σ with $|\sigma| < n$, if $l(d) \in \mathbf{out}_M(i \ \mathbf{after} \ \sigma)$ then $l(d) \in \mathbf{out}_M(S \ \mathbf{after} \ \sigma)$.
- Let $\mid \sigma \mid = n$ and $l(d) \in \mathbf{out}_M(i \ \mathbf{after} \ \sigma)$, then using Lemma 4.7.2 we know that $(\sigma)_{\mathcal{M}} \in \mathsf{nttraces}^{\Delta}_{\mathcal{M}}(i)$. If $l \neq \delta$ then $l(d) \in \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after} \ (\sigma)_{\mathcal{M}})$. So because i **mtioco**$_{\mathcal{M}}$ S we know that $l(d) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ (\sigma)_{\mathcal{M}})$ then $l(d) \in \mathbf{out}_M(S \ \mathbf{after} \ \sigma)$. If $l = \delta$ then $\delta(M) \in \mathbf{out}_M(i \ \mathbf{after} \ \sigma)$ and because $M = \max\langle M_1, \cdots, M_m\rangle$ we know that there exists z such that $\delta_z(M_z) \in \mathbf{out}_{\mathcal{M}}(i \ \mathbf{after} \ (\sigma)_{\mathcal{M}})$. Now using that i **mtioco**$_{\mathcal{M}}$ S we have that $\delta_z(M_z) \in \mathbf{out}_{\mathcal{M}}(S \ \mathbf{after} \ (\sigma)_{\mathcal{M}})$. So since $M = \max\langle M_1, \cdots, M_m\rangle$ we obtain that $\delta(M) \in \mathbf{out}_M(S \ \mathbf{after} \ \sigma)$.

□

The difference between the **mtioco**$_{\mathcal{M}}$ testing relation and the **tioco**$_M$ testing relation is that the former relaxes the input-enabled assumption in implementations and allows different bounds to detect quiescence per each channel. So the Theorem 4.7.3 shows that requiring input-enabledness and the same bound to detect quiescence for all channels (the maximum of all the previous bound) is enough to prove that a **mtioco**$_{\mathcal{M}}$-correct implementation is also a **tioco**$_M$-correct implementation with respect to a given specification.

4.8 Related work

The work of Heerink [28] is an extension of Tretmans' **ioco** testing theory [58] to deal with channels. In his untimed work, a testing theory is presented based on singular observers; only one output channel is observed at the time. Li et al. in [43] develop a similar theory with an alternative type of observers, called all-observers. The all-observers can see every output channels simultaneously. Later in [44] they extend their approach with queues. However, in contrast to the work presented here, both approaches are concerned with untimed systems.

Recently, another approach for the test generation on real-time systems was presented [38, 40], as we discussed in Chapter 3. But, their techniques consider neither quiescence nor multiple channels.

4.9 Conclusions

To the best of our knowledge, we propose the first attempt to generate test cases from multi input-output real-time specifications. More specifically, our contributions are:

- We show how the concept of multi input-output transition systems can be applied to the modeling of real-time systems.
- We develop a new parameterized conformance relation using the enriched real-time multi input-output transition systems: the **mtioco**$_{\mathcal{M}}$ testing relation.
- We relax the M-quiescent assumption by allowing different bounds for different channels.
- We relax the input-enabled assumption allowing input channels to be blocked.
- We relate our results with the timed relation **tioco** without channels.

We are continuing our work along two lines. Firstly, we are studying the *limit-completeness* of our approach, as explained in Section 4.6.2. Secondly, we are working on an implementation of the timed multiple input-output theory as an extension of the TorX tool [10, 12].

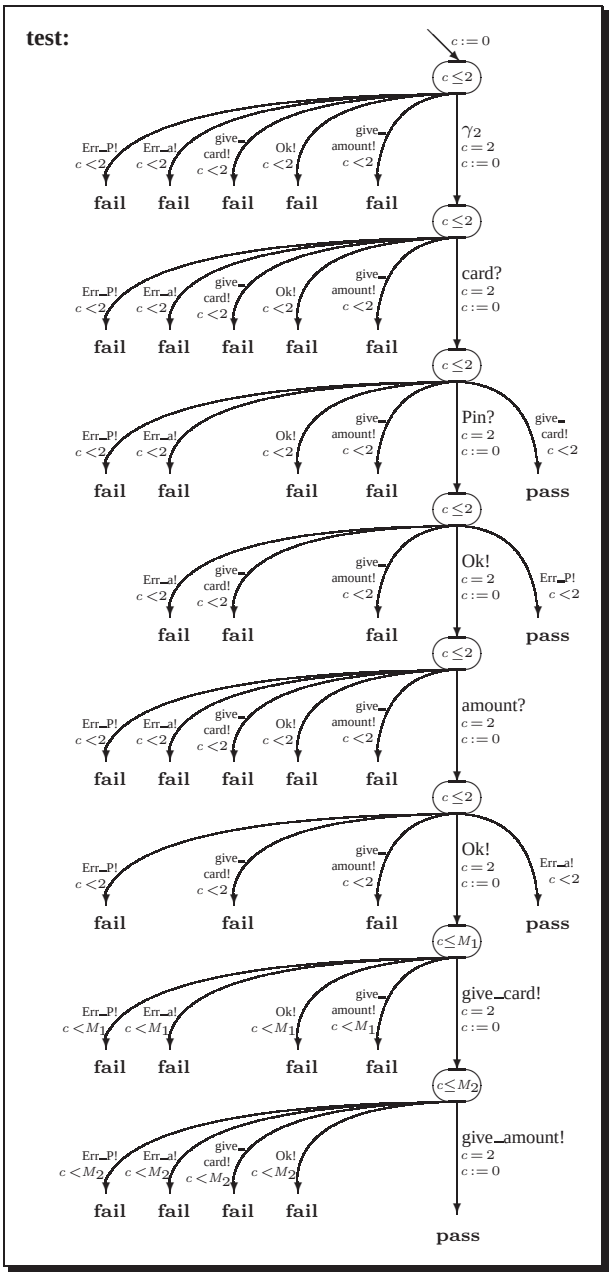


Figure 4.5: A test case for the cash machine considering $M_1 > 3$, $M_2 > 4$ and $M_3 > 5$

CHAPTER 5

Semantic coverage in testing

5.1 Introduction

As we anticipated in the first chapter, another interesting direction (studied in this chapter) is to investigate testing coverage. Even though so far we have considered extensions of labelled transition systems with time, to study coverage we start from regular, unextended labelled transition systems.

Since testing is inherently incomplete, test selection has vital importance. Coverage measures evaluate the quality of a test suite and help the tester to select test cases with maximal impact or minimum cost.

Existing coverage criteria for test suites are usually defined in terms of syntactic characteristics of the implementation under test or its specification. Typical black-box coverage metrics are state and transition coverage of the specification that would be visited by executing a test suite against it [61, 42, 49]. White-box testing often considers the number of statements, conditional branches, and paths through the implementation code that are touched by the test suite execution [47, 48, 8]. A disadvantage of this syntactic approach is that different coverage figures are assigned to systems that are behaviorally equivalent, but syntactically different. The approaches are based on syntactic model features, i.e. coverage figures are based on a specific model or program used as a reference. As a consequence, we may get different coverage results when we replace the model with one behaviorally equivalent but syntactically different.

Moreover, those coverage metrics do not take into account that certain failures are more severe than others, and that more testing effort should be devoted to uncover the most important bugs, while less critical system parts can be tested less thoroughly. In other words, these approaches fail to account for the non-uniform gravity of failures, whereas it would be natural to select test cases in such a way that the most critical system parts are tested most thoroughly.

It is important to realize that the weight of a failure cannot be extracted from a purely behavioural model, as it may depend, in an essential way, on the particular application of the implementation. The importance of the same bug may vary considerably between its occurrence in a part of an electronic game or in a part of the control of a nuclear power plant.

In practice, the exhaustiveness notion is usually problematic, since exhaustive test suites will contain infinitely many test cases. This raises the question of test selection, i.e. the selection of well-chosen, finite test suites that can be generated (and executed) within the

available resources. Test case selection is naturally related to a measure of coverage, indicating how much of the required conformance is tested for a given test selection. In this way, coverage measures can assist the tester in choosing test cases with maximal impact against some optimization criterion, e.g. number of tests, execution time, or cost.

Organization of the chapter This chapter introduces a semantic approach for test coverage that aims to overcome the two points mentioned above. Our point of departure is the weighted fault model (WFM) that assigns a weight in the specification to each potential error in an implementation. We define our coverage measures relative to these WFMs. Since WFMs are augmented specifications, our coverage framework qualifies as black-box. Moreover, since WFMs are infinite semantic objects, we need to represent them finitely if we want to model them or use them in algorithms. We provide such representations by the fault automata (Section 5.4). Fault automata are rooted in **ioco** testing theory [58] (recapitulated in Chapter 2), but their principles apply to a much wider setting.

We provide two ways of deriving WFMs from fault automata, namely the finite depth WFM(Section 5.5) and the discounted WFM(Section 5.6). The coverage measures obtained for these fault automata are invariant under behavioural equivalence.

For both weighted fault models, we provide algorithms that calculate and optimize test coverage (Section 5.8). In particular, we compute the (total, absolute and relative) coverage of a test suite with respect to a WFM. Also, given a test length k, we present an algorithm that finds the test of length k with maximal coverage and an algorithm that finds the shortest test with coverage exceeding a given coverage bound. Moreover, in Section 5.9 we apply our theory to the analysis and comparison of several test suites derived for a chat protocol. Related work is discussed in Section 5.10 and we end providing conclusions and suggestions for further research in Section 5.11.

5.2 Coverage measures in weighted fault models

Let L be any set. Then L^* denotes the set of all finite sequences over L. As in before chapters the empty sequence is denoted by ε and $|\sigma|$ denotes the length of a trace $\sigma \in L^*$. Moreover, we use $L^+ = L^* \setminus \{\varepsilon\}$. For σ, $\sigma' \in L^*$, we say that σ is a *prefix* of σ', if there exists a trace $\sigma'' \in L^*$ such that $\sigma' = \sigma \cdot \sigma''$. On the other hand, if σ is a prefix of σ', then σ' is a *suffix* of σ. We call σ a *proper prefix* of σ' and σ' a *proper suffix* of σ if σ is a prefix of σ', but $\sigma \neq \sigma'$. For any function $f : L \to \mathbb{R}^{\geq 0}$, we use the convention that $\sum_{x \in \emptyset} f(x) = 0$ and $\prod_{x \in \emptyset} f(x) = 1$.

5.2.1 Weighted fault models

A weighted fault model specifies the desired behaviour of a system by not only providing the correct system traces, but also giving the severity of the erroneous traces. In this section, we work with a fixed action alphabet L.

Definition 5.2.1. *A* weighted fault model *over L, denoted WFM(L), is a function $f : L^* \to \mathbb{R}^{\geq 0}$ such that*

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

Thus, f a WFM(L) assigns a non-negative error weight to each trace $\sigma \in L^*$. If $f(\sigma) = 0$, then σ represents correct system behaviour. If $f(\sigma) > 0$, then σ represents incorrect behaviour and $f(\sigma)$ denotes the severity of that error. In this way, the higher the value of $f(\sigma)$ is the worse the error. We refer to traces $\sigma \in L^*$ with $f(\sigma) > 0$ as *error traces* and traces with $f(\sigma) = 0$ as *correct traces* in f.

In order to define coverage measures relative to the total error weight, we require the total error weight $\sum_{\sigma \in L^*} f(\sigma)$ to be finite and non-zero.

5.2.2 Coverage measures

In this section we abstract from the exact shape of test cases and test suites. Given f a WFM over the action alphabet L, we only use that a test is a trace set, $t \subseteq L^*$. As well, a test suite is a collection of trace sets, $\mathbf{T} \subseteq \mathcal{P}(L^*)$. Then, we define the absolute and relative coverage with respect to f of a test and of a test suite. In this way, our coverage measures apply in all settings where test cases can be characterized as trace sets (in which case test suites can be characterized as collections of trace sets). This is the case for tests in TTCN [25], **ioco** testing theory [58] and FSM testing [61, 42].

Definition 5.2.2. *Let $f : L^* \to \mathbb{R}^{\geq 0}$ be a WFM(L), let $t \subseteq L^*$ be a set of traces and let $\mathbf{T} \subseteq \mathcal{P}(L^*)$ be a collection of sets of traces, then*

$$\begin{array}{lcl}
\mathsf{abscov}(t, f) & \triangleq & \sum_{\sigma \in t} f(\sigma) \\
\mathsf{abscov}(\mathbf{T}, f) & \triangleq & \mathsf{abscov}(\bigcup_{t \in \mathbf{T}} t, f) \\
\mathsf{totcov}(f) & \triangleq & \mathsf{abscov}(L^*, f) \\
\mathsf{relcov}(t, f) & \triangleq & \frac{\mathsf{abscov}(t, f)}{\mathsf{totcov}(f)} \\
\mathsf{relcov}(\mathbf{T}, f) & \triangleq & \frac{\mathsf{abscov}(\mathbf{T}, f)}{\mathsf{totcov}(f)}
\end{array}$$

The coverage of a test suite $\mathbf{T}$, with respect to f, measures the total weight of the errors that can be detected by tests in $\mathbf{T}$. The absolute coverage $\mathsf{abscov}(\mathbf{T}, f)$ simply accumulates the weights of all error traces in $\mathbf{T}$. Note that the weight of each trace is counted only once, since one test case is enough to detect the presence of an error trace in an IUT. The relative coverage, $\mathsf{relcov}(\mathbf{T}, f)$, yields the error weight in $\mathbf{T}$ as a fraction of the weight of all traces in L^*. Since absolute (coverage) numbers have meaning only if they are put in perspective of a maximum or average; we advocate that the relative coverage yields a good indication for the quality of a test suite.

Completeness of a test suite can easily be expressed in terms of coverage.

Definition 5.2.3. *Let $f : L^* \to \mathbb{R}^{\geq 0}$ be a WFM(L). A test suite $\mathbf{T} \subseteq \mathcal{P}(L^*)$ is* complete *with respect to f if*

$$\mathsf{relcov}(\mathbf{T}, f) \quad = \quad 1$$

The following proposition characterizes the complete test suites.

Proposition 5.2.4. *Let f be a WFM(L) and let $\mathbf{T} \subseteq \mathcal{P}(L^*)$ be a test suite. Then*

$$\mathbf{T} \textit{ is complete w.r.t. } f \quad \textit{if and only if} \quad \forall\ \sigma \in L^* : f(\sigma) > 0 : \exists\ t \in \mathbf{T} : \sigma \in t$$

Proof.
From Definition 5.2.3 we have that $\mathbf{T}$ is complete for f if and only if $\mathsf{relcov}(\mathbf{T}, f) = 1 = \frac{\mathsf{abscov}(\mathbf{T},f)}{\mathsf{totcov}(f)}$ if and only if $\sum_{\sigma \in \bigcup_{t\in\mathbf{T}} t} f(\sigma) = \sum_{\sigma \in L^*} f(\sigma)$ if and only if for all $\sigma \in L^*$ with $f(\sigma) > 0$ then $\sigma \in \bigcup_{t\in\mathbf{T}} t$ if and only if for all $\sigma \in L^*$ with $f(\sigma) > 0$ there exists a test $t \in \mathbf{T}$ such that $\sigma \in t$.

□

5.3 Test cases in labelled input-output transition systems

In this section we make use of all definitions from Chapter 2 about labelled input-output transition systems (LTS) and we only extend some of them because it prepares for the next section that treats an automaton-based formalism for specifying WFMs.

5.3.1 Labelled input-output transition systems

Definition 5.3.1. *Given $A = \langle Q, q^0, L, T\rangle$ a deterministic LTS, we define the input transition relation, T^I, and the output transition relation, T^O as*

$$\begin{array}{lcl} T^I & \triangleq & \textit{the restriction of } T \textit{ to } Q \times I \times Q \\ T^O & \triangleq & \textit{the restriction of } T \textit{ to } Q \times O \times Q \end{array}$$

Moreover, given $A = \langle Q, q^0, L, T\rangle$ a deterministic LTS, we write $T(q) = \{(l, q') \mid (q, l, q') \in T\}$ and similarly for $T^I(q)$ and $T^O(q)$. We denote by $\mathsf{outdeg}(q) = |T(q)|$ the out-degree of state q, meaning the number of outgoing transitions of q.

We required A to be deterministic only for technical simplicity. This is not a real restriction, since we can always determinize A. In case we incorporate quiescence by adding a self loop $q \xrightarrow{\delta} q$ labelled with a special label δ to each quiescent state q as in Chapter 2, we should note that: since quiescence is not preserved under determinization, we must first determinize and then add quiescence.

Example 5.3.2. *Figure 5.1 (a) presents a LTS of a cash machine: if the user ask for money, the machine should give money. We use this simple cash machine to make simple examples through all this chapter, later in Section 5.9 we present a more elaborated one. In Figure 5.1*

(b), we see the extension with quiescence. Since δ is not enabled in state q_1, we explicitly forbid the absence of outputs in q_1, i.e. the machine must give money. As in before chapters, the double circles represent the initial state.

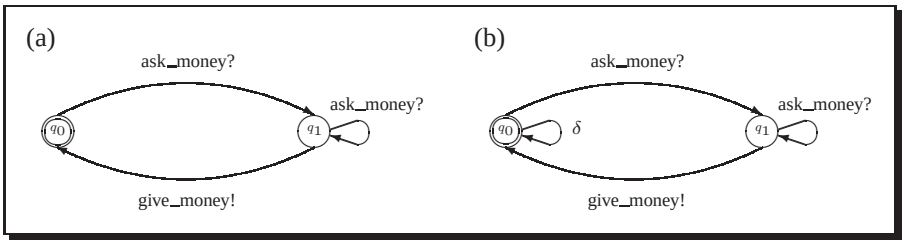


Figure 5.1: A LTS specification of a cash machine and its extension with quiescence

As follows we introduce some language theoretic concepts for LTSs.

Definition 5.3.3. *Let $A = \langle Q, q^0, L, T\rangle$ be a LTS, $\pi = q_0 l_1 q_1 l_2 \dots l_n q_n$ be a path in $\mathsf{paths}(A)$, and $\sigma \in L^*$ be any trace, not necessarily one from A, then*

$$
\begin{array}{lcl}
\mid \pi \mid & \triangleq & \mid \{q \mid \ q \in \pi\} \mid \\
\mathsf{last}(\pi) & \triangleq & q_n \\
\mathsf{trace}(\pi) & \triangleq & l_1 \cdot l_2 \dots l_n \\
\mathsf{reach}^k_A(\sigma) & \triangleq & \{q' \mid \ \exists\ \pi \in \mathsf{paths}(A) : |\sigma| = k \wedge \mathsf{trace}(\pi) = \sigma \wedge \mathsf{last}(\pi) = q'\} \\
\mathsf{reach}_A(\sigma) & \triangleq & \bigcup\limits_{k \in \mathbb{N}} \mathsf{reach}^k_A(\sigma)
\end{array}
$$

As before, we leave out the subscript A if it is clear from the context.

Then, $\mid \pi \mid$ denotes the number of states in the path π and last of a path denotes the last state of the path. With $\mathsf{trace}(\pi)$ we refer to the actions occurring in the path π and we write $\mathsf{traces}(A) = \{\mathsf{trace}(\pi) \mid \pi \in \mathsf{paths}(A)\}$ for the set of all traces in A. With $\mathsf{reach}^k_A(\sigma)$ we denote the set of states that can be reached in A in exactly k steps by following σ. Moreover, we write reach^k_A for the set of states that can be reached in k number of steps, by following any trace, $\mathsf{reach}^k_A = \bigcup\limits_{\sigma \in L^*} \mathsf{reach}^k_A(\sigma)$. Note that $\mathsf{reach}^k(\sigma)$ contain as most one state, since A is deterministic. Finally, $\mathsf{reach}_A(\sigma)$ is the set of states that can be reached through the trace σ in any number of steps and we write $\mathsf{reach}_A = \bigcup\limits_{\sigma \in L^*} \mathsf{reach}_A(\sigma)$ for the set of all reachable states in A. This definition is only a re-phrasing of the der set from Definition 2.2.5 (in Chapter 2, page 14).

Definition 5.3.4. *Let $A = \langle Q, q^0, L, T\rangle$ be a LTS and $q \in Q$ be a state in A, then by $A[q]$ we denote the LTS such that $A[q] = \langle Q, q, L, T\rangle$.*

Thus, $A[q]$ is the same as A, but with q as its initial state. This notation allows us to speak of paths, traces, in A starting from a state that is not the initial state. For instance, $\mathsf{paths}(A[q])$ denotes the set of paths starting from state q.

5.3.2 Test cases

As we already mentioned, the test cases for LTSs that we consider are based on the **ioco** testing theory [58]. As in TTCN, **ioco** test cases are adaptive. That is, the next action to be performed (observe the IUT, stimulate the IUT or stop the test) may depend on the test history, that is, the trace observed so far. If, after a trace σ, the tester decides to stimulate the IUT with an input $a?$, then the new test history becomes $\sigma \cdot a?$; in case of an observation, the test accounts for all possible continuations $\sigma \cdot b!$ with $b! \in O$ an output action. The **ioco** testing theory requires that tests fails fast, because they stop after the discovery of the first failure, and never fail immediately after an input. If $\sigma \in \mathsf{traces}(A)$, but $\sigma \cdot a? \notin \mathsf{traces}(A)$, then the behaviour after $\sigma \cdot a?$ is not specified, leaving room for implementation freedom. Formally, a test case consists of the set of all possible test histories obtained in this way.

For a detailed description of a test we refer the reader to Chapter 2. Here we only recall that given A a LTS a test is a finite, prefix-closed subset of L_A^* such that: if $\sigma \cdot a? \in t$, then $\sigma \cdot a'? \notin t$ for any $a'? \in L$ with $a \neq a'$; if $\sigma \cdot b! \in t$, then $\sigma \cdot b'! \in t$ for all $b'! \in O$; if $\sigma \notin \mathsf{traces}(A)$, then no proper suffix of σ is contained in t. Moreover, recall that we denote the set of all tests for A by $TESTS(A)$. The length of a test t, denoted $|\, t \,|$, is the length of the longest trace in t, $|\, t \,| = \max_{\sigma \in t}\{|\, \sigma \,|\}$. We denote by $TESTS^k(A)$ the set of all tests for A with length k.

Example 5.3.5. *Figure 5.2 shows two test cases for the cash machine from Figure 5.1 (b), represented as trees and augmented with verdicts pass and fail. The prefix closed trace set is obtained by taking all traces in these trees.*

Since each test of A is a set of traces, we can apply Definition 5.2.2 and speak of (absolute, total and relative) coverage of a test case (or a test suite) of A, with respect to f a WFM. However, not all WFMs are consistent with the interpretation that traces of A represent correct system behaviour, and that tests are "fail fast" and do not fail after an input.

Definition 5.3.6. *Let $A = \langle Q, q^0, L, T \rangle$ be a LTS and let $f : L^* \to \mathbb{R}^{\geq 0}$ be a WFM. Then f is* consistent with *A if $L = L_A$ and for all $\sigma \in L_A^*$ we have*

- *if $\sigma \in \mathsf{traces}(A)$, then $f(\sigma) = 0$ (correct traces have weight 0)*
- *$f(\sigma \cdot a?) = 0$ (no failure occurs after an input)*
- *if $f(\sigma) > 0$ then $f(\sigma \cdot \sigma') = 0$ for all $\sigma' \in L_A^+$ (at most one failure per trace)*

The following result states that the set containing all possible test cases has complete coverage.

Theorem 5.3.7. *Let $A = \langle Q, q^0, L, T \rangle$ be a LTS and f be a WFM consistent with A. Then, the set $TESTS(A)$ of all test cases for A is complete with respect to f.*

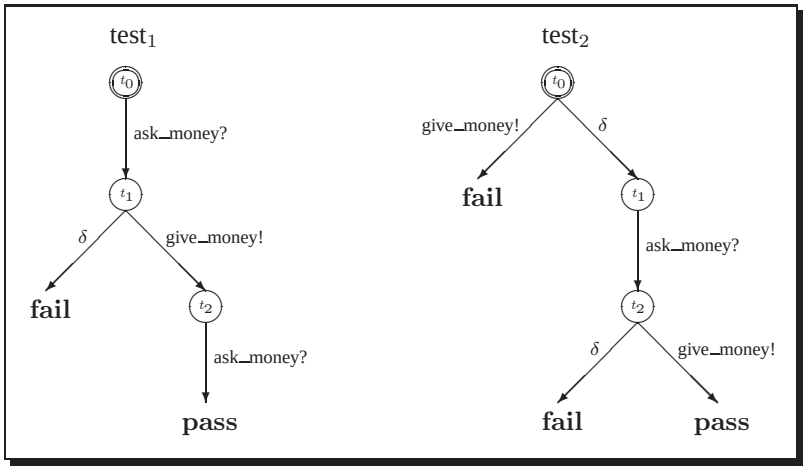


Figure 5.2: Two test cases for the LTS from Figure 5.1, augmented with verdicts **pass** and **fail**

Proof.
For all $\sigma \in L$ with $f(\sigma) > 0$, we build a test $t \in TESTS(A)$ with $\sigma \in t$. Let σ be $\sigma = l_1 \cdot l_2 \cdot \ldots \cdot l_n$. Then, for $1 \leq z \leq n$, we define a set X_z by

$$X_z \quad = \quad \begin{cases} \{l_1 \ldots l_z\} & l_z \in I \\ \{l_1 \ldots l_{z-1} b \mid b \in O\} & l_z \in O \end{cases}$$

The set t is defined as $t = \bigcup_{1 \leq z \leq n} X_z$. Since f is consistent with A, the set t is a test in $TESTS(A)$. Clearly, t contains σ. Now, Proposition 5.2.4 yields that $TESTS(A)$ is complete for f.

□

5.4 Fault automata

Weighted fault models are infinite semantic objects. This section introduces *fault automata*, which provide a syntactic format for specifying WFMs. A fault automaton A is a LTS augmented with a state weight function r. Then, A is a behavioural specification of the system; its traces represent the correct system behaviours. Hence, these traces will be assigned error weight 0; traces not in A are erroneous and get an error weight through r, as explained below.

Definition 5.4.1. *A* fault automaton *(FA) $\mathcal{F}$ is a pair $\langle A, r \rangle$, where $A = \langle Q, q^0, L, T \rangle$ is a LTS and $r : Q \times O \to \mathbb{R}^{\geq 0}$ is a function such that*

$$r(q, b!) > 0 \quad \triangleq \quad \nexists\, q' \in Q : (q, b!, q') \in T$$

We denote the components of $\mathcal{F}$ by $A_{\mathcal{F}}$ and $r_{\mathcal{F}}$ and leave out the subscripts $\mathcal{F}$ if it is clear from the context.

Then we require that if $r(q, b!) > 0$ then there is no $b!$-successor of q in $\mathcal{F}$. Moreover, we extend r to a function $r : Q \times L \to \mathbb{R}^{\geq 0}$ by putting $r(q, a?) = 0$ for $a? \in I$ and define $\overline{r} : Q \to \mathbb{R}^{\geq 0}$ as

$$\overline{r}(q) \quad \triangleq \quad \sum_{b \in O(q)} r(q, b)$$

Thus, $\overline{r}$ accumulates the weight of all the erroneous outputs in a state. We lift all concepts and notations (e.g. traces, paths, etc.) that have been defined for LTSs to FA.

Example 5.4.2. *Figure 5.3 presents an FA for our cash machine example from Figure 5.1 (b). We give error weight 10 if in state q_0 the cash machine gives us money with out asking; and weight 5 if in state q_1 the cash machine does not gives us money ones we did ask for it.*

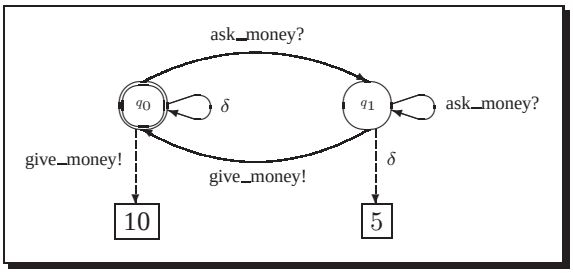


Figure 5.3: An FA for our cash machine extended with a value per error per state

We wish to construct f a WFM from the $\mathcal{F}$ an FA, using r to assign weights to traces not in A. If there is no outgoing $b!$-transition in state q, then the idea is that, for a trace σ ending in q, the (incorrect) trace $\sigma \cdot b!$ gets weight $r(q, b!)$. Doing so, however, could cause the total error weight $\mathsf{totcov}(f)$ to be infinite.

We consider two solutions to this problem. Firstly, in finite depth WFM(Section 5.5), for a given $k \in \mathbb{N}$, we only consider faults in traces of length k or smaller. Secondly, in discounted WFM(Section 5.6) we obtain finite total coverage through discounting, while considering error weight in all traces. The solutions presented here are only two potential solutions, there are many other ways to derive a WFM from a fault automaton.

5.5 Finite depth weighted fault models

As said before, the finite depth model derives a WFM from an FA $\mathcal{F}$, for a given $k \in \mathbb{N}$, by ignoring all traces of length longer than k, i.e. by putting their error weight to 0. For all other

traces, the weight is obtained through the function r. If σ is a trace of $\mathcal{F}$ ending in the state q, but $\sigma \cdot b!$ is not a trace in $\mathcal{F}$, then $\sigma \cdot b!$ gets weight $r(q, b!)$.

Definition 5.5.1. *Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$ and $r : Q \times O \to \mathbb{R}^{\geq 0}$. Then, let k be a number in $\mathbb{N}$, we define the function $f^k_{\mathcal{F}} : L^* \to \mathbb{R}^{\geq 0}$ by*

$$\begin{array}{lcll} f^k_{\mathcal{F}}(\varepsilon) & \triangleq & 0 & \\ f^k_{\mathcal{F}}(\sigma \cdot b) & \triangleq & \begin{cases} r(q,b) & q \in \mathsf{reach}^z_{\mathcal{F}}(\sigma) \wedge b \in O \wedge z \leq k \\ 0 & \textit{otherwise} \end{cases} & \end{array}$$

This function is uniquely defined because $\mathcal{F}$ is deterministic, so that there is at most one state q with $q \in \mathsf{reach}^k_{\mathcal{F}}(\sigma)$. Also, if $f^k_{\mathcal{F}}(\sigma \cdot b) = r(q, b) > 0$, then $\sigma \in \mathsf{traces}(\mathcal{F})$, but $\sigma \cdot b \notin \mathsf{traces}(\mathcal{F})$.

The following proposition states that $f^k_{\mathcal{F}}$ is a WFM consistent with $\mathcal{F}$, provided that $\mathcal{F}$ contains at most one state with a positive accumulated weight and that is reachable within k steps.

Proposition 5.5.2. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$ and $r : Q \times O \to \mathbb{R}^{\geq 0}$. Let k be a number in $\mathbb{N}$, then if there is a number $z \leq k$ and a state q in $\mathsf{reach}^z_{\mathcal{F}}$ with $\overline{r}(q) > 0$, then $f^k_{\mathcal{F}}$ is a WFM consistent with $\mathcal{F}$.*

Proof.
We have that $f^k_{\mathcal{F}} < \infty$ because it is finite and $f^k_{\mathcal{F}} > 0$ because $\exists\ z \leq k : \exists\ q \in \mathsf{reach}^z_{\mathcal{F}} : \overline{r}(q) > 0$. By construction $L = L_{\mathcal{F}}$ and for all $\sigma \in \mathsf{traces}(\mathcal{F}) : f^k_{\mathcal{F}}(\sigma) = 0$.

Now we only need to prove that if $f^k_{\mathcal{F}}(\sigma) > 0$ then for all $\sigma' \in L^+_{\mathcal{F}} : f^k_{\mathcal{F}}(\sigma \cdot \sigma') = 0$. So, because $f^k_{\mathcal{F}}(\sigma) > 0$ we know that there exists σ'' such that $\sigma = \sigma'' \cdot l$ and there exists a state q' such that $q^0 \xrightarrow{\sigma} q'$ with $r(q', l) > 0$, this means that then for all σ' we have that $f^k_{\mathcal{F}}(\sigma \cdot \sigma') = 0$.

□

Example 5.5.3. *Given $\mathcal{F}$ the FA from Figure 5.3 and $k = 3$, then Figure 5.4 shows the function $f^k_{\mathcal{F}}$. Using the tests test$_1$ and test$_2$ presented in Figure 5.2, we obtain* $\mathsf{abscov}(\textit{test}_1, f^k_{\mathcal{F}}) = 5$ *and* $\mathsf{abscov}(\textit{test}_2, f^k_{\mathcal{F}}) = 15$. *Moreover, if* $\mathbf{T} = \{\textit{test}_1, \textit{test}_2\}$ *then* $\mathsf{abscov}(\mathbf{T}, f^k_{\mathcal{F}}) = 20$.

5.6 Discounted weighted fault models

While finite depth WFMs achieve finite total coverage by considering finitely many traces, discounted WFMs take into account the error weights of all traces. To do so, only finitely many traces may have weight greater than ϵ, for any $\epsilon > 0$. One way to do this is by discounting: lowering the weight of a trace proportional to its length. The rationale behind this is that errors in the near future are worse than errors in the far future, and hence, the latter should have lower weights.

In its basic form, f a discounted WFM for $\mathcal{F}$ an FA sets the weight of a trace $\sigma \cdot b!$ to

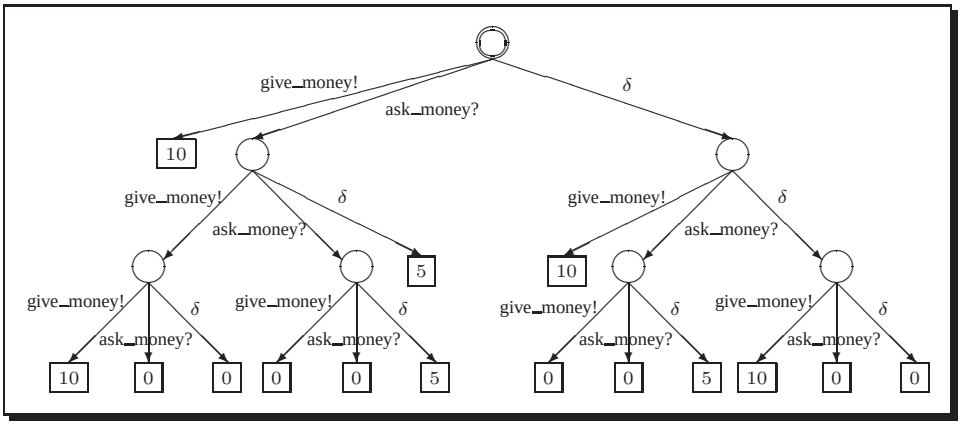


Figure 5.4: Function $f^k_{\mathcal{F}}$, with $k = 3$, consistent with $\mathcal{F}$ from Figure 5.3

$\alpha^{|\sigma|} \cdot r(q, b!)$, for some discount factor α in $(0, 1)$. If we take α small enough, then one can easily show that $\sum_{\sigma \in L^*} f(\sigma) < \infty$. To be precise, we take $\alpha < \frac{1}{d}$, where d is the branching degree of $\mathcal{F}$, i.e. $d = \max_{q \in Q} \{\mathsf{outdeg}(q)\}$. Indeed, let $\alpha \cdot d < 1$ and $M = \max_q \{\frac{r(q,b)}{\alpha}\}$. Then $f(\sigma) \leq \alpha^{|\sigma|} \cdot M$. Since there are at most d^k traces of length k in $\mathcal{F}$, it follows that:

$$\sum_{\sigma \in L^*} f(\sigma) \quad = \quad \sum_{k \in \mathbb{N}} \sum_{\sigma \in L^k} \alpha^k \cdot M \quad \leq \quad \sum_{k \in \mathbb{N}} d^k \cdot \alpha^k \cdot M \quad = \quad \frac{M}{1 - d\alpha} \quad < \quad \infty$$

To obtain more flexibility, we allow the discount to vary per transition. That is, we work with a discount function $\alpha : Q \times L \times Q \to \mathbb{R}^{\geq 0}$ that assigns a positive weight to each transition of $\mathcal{F}$. Then we discount the trace $l_1 \cdots l_k$ obtained from the path $q_0 l_1 q_1 \ldots q_k$ by $\alpha(q_0, l_1, q_1) \cdot \alpha(q_1, l_2, q_2) \cdots \alpha(q_{k-1}, l_k, q_k)$. The requirement that α is small enough now becomes

$$\sum_{l \in L, q' \in Q} \alpha(q, l, q') \quad < \quad 1$$

for each state q.

We can even be more flexible and, in the sum above, do not range over states in which all paths are finite, as in these states we have finite total coverage anyway. Thus, if $\mathsf{Inf}_{\mathcal{F}}$ is the set of all states in $\mathcal{F}$ with at least one outgoing infinite path, we require for all states q:

$$\sum_{l \in L, q' \in \mathsf{Inf}_{\mathcal{F}}} \alpha(q, l, q') \quad < \quad 1$$

Definition 5.6.1. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$ and $r : Q \times O \to \mathbb{R}^{\geq 0}$.*

Then the set $\mathit{Inf}_{\mathcal{F}} \subseteq Q$ *of states with at least one infinite path is defined as*

$$\mathit{Inf}_{\mathcal{F}} \quad \triangleq \quad \{q \in Q \mid \exists\ \pi \in \mathsf{paths}(\mathcal{F}[q]) : |\pi| > |Q|\}$$

The following proposition states that the set $\mathsf{Inf}_{\mathcal{F}}$ is closed under taking the predecessors of a state.

Proposition 5.6.2. *Let* $\mathcal{F} = \langle A, r\rangle$ *be an FA with* $A = \langle Q, q^0, L, T\rangle$ *and* $r : Q \times O \to \mathbb{R}^{\geq 0}$*, then*

$$\textit{if} \quad (q, l, q') \in T \wedge q' \in \mathit{Inf}_{\mathcal{F}} \quad \textit{then} \quad q \in \mathit{Inf}_{\mathcal{F}}$$

Proof.
Let (q, l, q') be in T and $q' \in \mathsf{Inf}_{\mathcal{F}}$ then there exists a path π in $\mathsf{paths}(\mathcal{F}[q])$ with $|\pi| > |Q_{\mathcal{F}}|$ then let $\pi' = qlq'\pi$ then $|\pi'| > |\pi| > |Q_{\mathcal{F}}| \wedge \pi \in \mathsf{paths}(\mathcal{F}[q])$, so by definition $q \in \mathsf{Inf}_{\mathcal{F}}$. □

Definition 5.6.3. *Let* $\mathcal{F} = \langle A, r\rangle$ *be an FA with* $A = \langle Q, q^0, L, T\rangle$ *and* $r : Q \times O \to \mathbb{R}^{\geq 0}$*. Then, a* discount function *for* $\mathcal{F}$ *is a function* $\alpha : Q \times L \times Q \to \mathbb{R}^{\geq 0}$ *such that*

- *for all* $q, q' \in Q$*, and* $l \in L$ *we have*
$$\alpha(q, l, q') = 0 \quad \triangleq \quad (q, l, q') \notin T$$
- *for all* $q \in Q$*, we have*
$$\sum_{l \in L, q' \in \mathit{Inf}_{\mathcal{F}}} \alpha(q, l, q') \quad < \quad 1$$

Definition 5.6.4. *Let* α *be a discount function for* $\mathcal{F} = \langle A, r\rangle$ *an FA with* $A = \langle Q, q^0, L, T\rangle$ *and* $r : Q \times O \to \mathbb{R}^{\geq 0}$*. Let* $\pi = q_0 l_1 \ldots q_n$ *be a path in* $\mathsf{paths}(\mathcal{F})$*, then*

$$\alpha(\pi) \quad \triangleq \quad \prod_{z=1}^{n} \alpha(q_{z-1}, l_z, q_z)$$

Definition 5.6.5. *Let* $\mathcal{F} = \langle A, r\rangle$ *be an FA with* $A = \langle Q, q^0, L, T\rangle$*,* $r : Q \times O \to \mathbb{R}^{\geq 0}$*,* $q \in Q$ *and* α *a discount function for* $\mathcal{F}$*. We define the function* $f^{\alpha}_{\mathcal{F}} : L^* \to \mathbb{R}^{\geq 0}$ *by*

$$\begin{array}{lcl} f^{\alpha}_{\mathcal{F}}(\varepsilon) & \triangleq & 0 \\ f^{\alpha}_{\mathcal{F}}(\sigma \cdot b) & \triangleq & \left\{ \begin{array}{ll} \alpha(\pi) \cdot r(q, b) & q \in \mathsf{reach}_{\mathcal{F}}(\sigma) \wedge b \in O \wedge \mathsf{trace}(\pi) = \sigma \\ 0 & \textit{otherwise} \end{array} \right. \end{array}$$

Since $\mathcal{F}$ *is deterministic, there is at most one* π *with* $\mathsf{trace}(\pi) = \sigma$ *and at most one* $q \in \mathsf{reach}(\sigma)$*. Hence, the function above is uniquely defined.*

The following proposition states that $f^{\alpha}_{\mathcal{F}}$ is a WFM consistent with $\mathcal{F}$, provided that $\mathcal{F}$ contains as most one reachable state with a positive accumulated weight.

Proposition 5.6.6. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α be a discount function for $\mathcal{F}$. If there is a state $q \in \mathsf{reach}_{\mathcal{F}}$ with $\overline{r}(q) > 0$, then $f^{\alpha}_{\mathcal{F}}$ is a WFM consistent with $\mathcal{F}$.*

Proof.
We have that $f^{\alpha}_{\mathcal{F}} < \infty$ because $\sum_{l \in L, q' \in \mathsf{Inf}_{\mathcal{F}}} \alpha(q, l, q') < 1$ and $f^{\alpha}_{\mathcal{F}} > 0$ because $\exists\ q \in \mathsf{reach}_{\mathcal{F}} : \overline{r}(q) > 0$. By construction $L = L_{\mathcal{F}}$. Now we need to show that for all $\sigma, q' \in \mathsf{traces}(\mathcal{F})$: $f^{\alpha}_{\mathcal{F}}(\sigma) = 0$. Let $\sigma = \sigma' \cdot l$ then exists a path π such that $\mathsf{trace}(\pi) = \sigma'$. So, there exists a state $q \in Q$ such that $q^0 \xrightarrow{\sigma'} q$, $q \xrightarrow{l} q'$ and $r(q, l) = 0$ then $f^{\alpha}_{\mathcal{F}}(\sigma) = 0$.

To prove that if $f^{\alpha}_{\mathcal{F}}(\sigma) > 0$ then for all $\sigma' \in L^{+}_{\mathcal{F}} : f^{\alpha}_{\mathcal{F}}(\sigma \cdot \sigma') = 0$. So, because $f^{\alpha}_{\mathcal{F}}(\sigma) > 0$ we know that there does not exists a path π with $\mathsf{trace}(\pi) = \sigma$ and σ in $\mathsf{traces}(\mathcal{F})$ then $f^{\alpha}_{\mathcal{F}}(\sigma \cdot \sigma') = 0$.

□

Example 5.6.7. *Figure 5.5 presents the function $f^{\alpha}_{\mathcal{F}}$ for $\mathcal{F}$ from Figure 5.3 with $\alpha(q, l, q') = \gamma$ for every transition $(q, l, q') \in T$. Using the tests t_1 and t_2 presented in Figure 5.2, we obtain* $\mathsf{abscov}(\mathit{test}_1, f^{\alpha}_{\mathcal{F}}) = \gamma 5$ *and* $\mathsf{abscov}(\mathit{test}_2, f^{\alpha}_{\mathcal{F}}) = 10 + \gamma^2 5$. *Moreover, if* $\mathbf{T} = \{\mathit{test}_1, \mathit{test}_2\}$ *then* $\mathsf{abscov}(\mathbf{T}, f^{\alpha}_{\mathcal{F}}) = 10 + \gamma 5 + \gamma^2 5$.

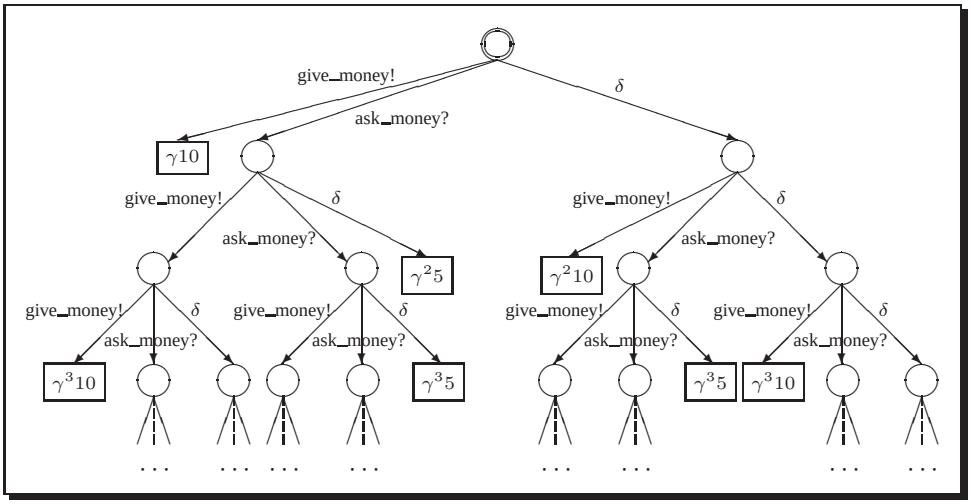


Figure 5.5: Function $f^{\alpha}_{\mathcal{F}}$ for $\mathcal{F}$ from Figure 5.3 with $\alpha(q, l, q') = \gamma$

5.7 Properties

5.7.1 Calibration of the discount function

Discounting weighs errors in short traces more than in long traces. Thus, if we discount too much, we may obtain very high test coverage just with a few short test cases. The calibration result (Theorem 5.7.2) presented in this section shows that, for any FA $\mathcal{F}$, any given length k and $\epsilon > 0$, there exists a discount function α such that the relative coverage of all test cases of length k or shorter is less than ϵ. This means that by choosing the right α we can always make the contribution of short tests (i.e. smaller than the given k) arbitrarily small (i.e. $< \epsilon$).

For technical reasons, the weight assignment function of an FA have to be fair, i.e. all states in Inf must be able to reach some state with a positive weight.

Definition 5.7.1. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$ and $r : Q \times O \to \mathbb{R}^{\geq 0}$, then $\mathcal{F}$ has a* fair weight assignment *if*

$$\forall\ q \in \mathit{Inf}_{\mathcal{F}} : \exists\ q' \in \mathsf{reach}_{\mathcal{F}[q]} : \overline{r}(q') > 0$$

Theorem 5.7.2. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and a fair weight assignment. Then there exists a family of discount functions $\{\alpha_\epsilon\}_{\epsilon\in(0,1)}$ for $\mathcal{F}$ such that for all $k \in \mathbb{N}$*

$$\lim_{\epsilon\to 0}\ \mathsf{relcov}\big(TESTS^k(f_{\mathcal{F}}^{\alpha_\epsilon}), f_{\mathcal{F}}^{\alpha_\epsilon}\big) \quad = \quad 0$$

The prove of this theorem follows from the next propositions and definitions.

Definition 5.7.3. *Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and a number $\epsilon \in (0, 1)$. We define a discount function $\alpha_\epsilon : Q \times L \times Q \to (0, 1)$ as*

$$\alpha_\epsilon(q, l, q') \quad \triangleq \quad \begin{cases} \frac{(1-\epsilon)}{|\mathsf{OutInf}(q)|} & (q, l, q') \in T \wedge q' \in \mathit{Inf}_{\mathcal{F}} \\ > 0 & (q, l, q') \in T \wedge q' \notin \mathit{Inf}_{\mathcal{F}} \\ 0 & \textit{otherwise} \end{cases}$$

Here $\mathsf{OutInf}(q) = \{(l, q') \in T(q) \mid q' \in \mathit{Inf}_{\mathcal{F}}\}$. We usually write A_ϵ to denote the A^{α_ϵ} matrix.

Definition 5.7.4. *Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$ and $r : Q \times O \to \mathbb{R}^{\geq 0}$, we define the vector $\mathbf{1}_{\mathit{Inf}}$ indexed by $q \in Q$ as*

$$\mathbf{1}_{\mathit{Inf}}(q) \quad \triangleq \quad \begin{cases} 1 & q \in \mathit{Inf}_{\mathcal{F}} \\ 0 & \textit{otherwise} \end{cases}$$

Proposition 5.7.5. *$\mathbf{1}_{\mathit{Inf}}$ is an eigenvector of A_ϵ with eigenvalue $1 - \epsilon$,*

$$A_\epsilon \cdot \mathbf{1}_{\mathit{Inf}} \quad = \quad (1-\epsilon)\cdot \mathbf{1}_{\mathit{Inf}}$$

Proof.
Firstly, we consider $q \in \mathsf{Inf}_{\mathcal{F}}$, then

$$\begin{aligned}
(A_\epsilon \cdot \mathbf{1}_{\mathsf{Inf}})_q &= \sum_{q' \in Q} (A_\epsilon)_{qq'} \cdot \mathbf{1}_{\mathsf{Inf}}(q') \\
&= \sum_{q' \in \mathsf{Inf}_{\mathcal{F}}} (A_\epsilon)_{qq'} \\
&= \sum_{q' \in \mathsf{Inf}_{\mathcal{F}}} \sum_{l \in L} \alpha_\epsilon(q,l,q') \\
&= \sum_{(l,q') \in \mathit{OutInf}\,(q)} \frac{(1-\epsilon)}{|\mathit{OutInf}\,(q)|} \\
&= |\,\mathit{OutInf}\,(q)\,| \cdot \frac{(1-\epsilon)}{|\mathit{OutInf}\,(q)|} \\
&= 1-\epsilon
\end{aligned}$$

For $q \in Q/\mathsf{Inf}_{\mathcal{F}}$ we get, using Proposition 5.6.2

$$\begin{aligned}
(A_\epsilon \cdot \mathbf{1}_{\mathsf{Inf}})_q &= \sum_{q' \in Q} (A_\epsilon)_{qq'} \cdot \mathbf{1}_{\mathsf{Inf}}(q') \\
&= \sum_{q' \in \mathsf{Inf}} (A_\epsilon)_{qq'} \\
&= \sum_{q' \in \mathsf{Inf}} \sum_{l \in L} \alpha_\epsilon(q,l,q') \\
&= \sum_{q' \in \mathsf{Inf}} \sum_{l \in L} 0 \\
&= 0
\end{aligned}$$

□

Corollary 5.7.6. $\big(A_\epsilon\big)^n \cdot \mathbf{1}_{\mathit{Inf}} \quad = \quad \big(1-\epsilon\big)^n \cdot \mathbf{1}_{\mathit{Inf}}$

Proof.
Direct using induction over n.

□

Proposition 5.7.7. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and a fair weight assignment r then for every $q \in \mathsf{Inf}_{\mathcal{F}}$*

$$\Big(\sum_{z=0}^{|Q|-1} A_\epsilon^z \cdot \overline{r}\Big)_q \quad > \quad 0$$

Proof.
Note that $(A_\epsilon^z)_{qq'} > 0$ implies that q' can be reached from q in z transitions. As $\mathcal{F}$ is based on an FA every state q is at most $|Q\,| - 1$ transitions removed any of the states q' that can be reached from it, so that there is an $z < |Q\,|$ with $(A_\epsilon^z)_{qq'} > 0$. Hence $\big(\sum_{z=0}^{|Q|-1} A_\epsilon^z\big)_{qq'} > 0$ for

any pair of such $q, q' \in Q$. By the definition of fair weight assignment all states $q \in \mathsf{Inf}_{\mathcal{F}}$ can reach an $q' \in Q$ with $\overline{r}(q') > 0$. Thus we get

$$\big(\sum_{z=0}^{|Q|-1} A_\epsilon^z \cdot \overline{r}\big)_q \quad = \quad \sum_{q' \in Q} \sum_{z=0}^{|Q|-1} (A_\epsilon^z)_{qq'} \cdot \overline{r}(q') \quad > \quad 0$$

□

Now we are ready to show that the family of discounted functions $\{\alpha_\epsilon\}_{\epsilon \in (0,1)}$ has the desired properties.

Proposition 5.7.8. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and a fair weight assignment. Then for every $q \in \mathsf{Inf}_{\mathcal{F}}$*

$$\lim_{\epsilon \to 0} \mathsf{relcov}\big(TESTS_k(f_{\mathcal{F}}^{\alpha_\epsilon}), f_{\mathcal{F}}^{\alpha_\epsilon}\big) \quad = \quad 0$$

Proof.
Recall that

$$\mathsf{relcov}\big(TESTS_k(f_{\mathcal{F}}^{\alpha_\epsilon}), f_{\mathcal{F}}^{\alpha_\epsilon}\big) \quad = \quad \frac{\mathsf{abscov}(TESTS_k(f_{\mathcal{F}}^{\alpha\epsilon}), f_{\mathcal{F}}^{\alpha\epsilon})}{\mathsf{totcov}(f_{\mathcal{F}}^{\alpha\epsilon})}$$

As $\mathsf{abscov}\big(TESTS_k(f_{\mathcal{F}}^{\alpha_\epsilon}), f_{\mathcal{F}}^{\alpha_\epsilon}\big)$ is always finite, it suffices to show that

$$\lim_{\epsilon \to 0} \mathsf{totcov}(f_{\mathcal{F}}^{\alpha_\epsilon}) \quad = \quad \infty$$

This can be shown as follows. Let

$$r_{\mathsf{min}} \quad = \quad \min_{q' \in \mathsf{Inf}} \big(\sum_{z=0}^{|Q|-1} A_\epsilon^z \cdot \overline{r}\big)_{q'}$$

Then Proposition 5.7.7 yields that $r_{\mathsf{min}} > 0$. Moreover, we have for all $q' \in Q$ that

$$\big(\sum_{z=0}^{|Q|-1} A_\epsilon^z \cdot \overline{r}\big)_{q'} \quad \geq \quad r_{\mathsf{min}} \cdot \mathbf{1}_{\mathsf{Inf}}(q')$$

Therefore,

$$\begin{aligned}
\mathsf{totcov}(f_{\mathcal{F}}^{\alpha_\epsilon}) \quad &= \quad \big(\sum_{z=0}^{\infty} A_\epsilon^z \cdot \overline{r}\big)_q \\
&= \quad \big(\sum_{j=0}^{\infty} A_\epsilon^{j|Q|} \cdot \sum_{z=0}^{|Q|-1} A_\epsilon^z \cdot \overline{r}\big)_q \\
&\geq \quad r_{\mathsf{min}} \cdot \big(\sum_{j=0}^{\infty} A_\epsilon^{j|Q|} \cdot \mathbf{1}_{\mathsf{Inf}}\big)_q \\
&= \quad r_{\mathsf{min}} \cdot \big(\sum_{j=0}^{\infty} (1-\epsilon)^{j|Q|} \cdot \mathbf{1}_{\mathsf{Inf}}\big)_q \\
&= \quad \frac{r_{\mathsf{min}}}{1-(1-\epsilon)^{|Q|}}
\end{aligned}$$

As $r_{\mathsf{min}} > 0$ and $1 - (1-\epsilon)^{|Q|}$ is of the order $\mathcal{O}(\epsilon)$, we get

$$\lim_{\epsilon \to 0} \left(\sum_{z=0}^{\infty} A_\epsilon^z \cdot \overline{r} \right)_q \quad = \quad \infty$$

□

5.7.2 Invariance under bisimilarities

It is not difficult to see that our coverage notions are truly semantic in that they are invariant under r-preserving bisimilarity, and α-preserving bisimilarity.

Definition 5.7.9. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and let $R \subseteq Q \times Q$ be an equivalence relation on the state space of $\mathcal{F}$. Then R is a* r-preserving bisimulation on $\mathcal{F}$ *if for all $(q, q') \in R$, $l \in L$, we have*

- *if $q \xrightarrow{l} q_1$ then there is a transition $q' \xrightarrow{l} q_1'$ with $(q_1, q_1') \in R$*
- *for all $b \in O : r(q, b) = r(q', b)$*

Theorem 5.7.10. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, R be a r-preserving bisimulation on $\mathcal{F}$ and $(q, q') \in R$, then for all k*

$$f^k_{\mathcal{F}[q]} \quad = \quad f^k_{\mathcal{F}[q']}$$

Proof.
We prove that $f^k_{\mathcal{F}[q]}(\sigma \cdot b) = f^k_{\mathcal{F}[q']}(\sigma \cdot b)$. Let $f^k_{\mathcal{F}[q]}(\sigma \cdot b) = r(q_1, b)$ then using Definition 5.5.1 we know that $q_1 \in \mathsf{reach}_{f^k_{\mathcal{F}[q]}}(\sigma) \ \wedge b \in O$ then because R is an r-preserving bisimulation on $\mathcal{F}$ then $\exists \ q_1' \in \mathsf{reach}_{f^k_{\mathcal{F}[q']}}(\sigma)$ such that $(q_1, q_1') \in R$ and moreover for $b \in O$ we have $f^k_{\mathcal{F}[q']}(\sigma \cdot b) = r(q_1', b)$. Now, again because $(q_1, q_1') \in R$ we know that $r(q_1, b) = r(q_1', b)$. So, $f^k_{\mathcal{F}[q]}(\sigma \cdot b) = f^k_{\mathcal{F}[q']}(\sigma \cdot b)$.

□

Definition 5.7.11. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and let $R \subseteq Q \times Q$ be an equivalence relation on the state space of $\mathcal{F}$. Then R is a* α-preserving bisimulation on $\mathcal{F}$ *if for all $(q, q') \in R$, $l \in L$, we have*

- *if $q \xrightarrow{l} q_1$ then there is a transition $q' \xrightarrow{l} q_1'$ with $(q_1, q_1') \in R$*
- *for all $l \in L : \alpha(q, l, q_1) = \alpha(q', l, q_1')$*

Theorem 5.7.12. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, R be a r-preserving bisimulation and α-preserving bisimulation on $\mathcal{F}$. Then,*

$$\textit{if} \quad (q, q') \in R \quad \textit{then} \quad f^\alpha_{\mathcal{F}[q]} = f^\alpha_{\mathcal{F}[q']}$$

Proof.
We prove that $f^\alpha_{\mathcal{F}[q]}(\sigma \cdot b) = f^\alpha_{\mathcal{F}[q']}(\sigma \cdot b)$. Let $f^\alpha_{\mathcal{F}[q]}(\sigma \cdot b) = \alpha(\pi) \cdot r(q_1, b)$ then using Definition 5.6.5 we know that $q_1 \in \mathsf{reach}_{f^\alpha_{\mathcal{F}[q]}}(\sigma) \wedge b \in O \wedge \mathsf{trace}(\pi) = \sigma$ then because R is an

r-preserving bisimulation on $\mathcal{F}$ then $\exists\ q_1' \in \mathsf{reach}_{f^{\alpha}_{\mathcal{F}[q']}}(\sigma) \wedge \exists\ \pi' : \mathsf{trace}(\pi) = \sigma'$ such that $(q_1, q_1') \in R$ and moreover for $b \in O$ we have $f^{\alpha}_{\mathcal{F}[q']}(\sigma \cdot b) = \alpha(\pi') \cdot r(q_1', b)$. Now, again because $(q_1, q_1') \in R$ using that R is α-preserving bisimulation on $\mathcal{F}$ then $\alpha(\pi) = \alpha(\pi')$. So, $f^{\alpha}_{\mathcal{F}[q]}(\sigma \cdot b) = f^{\alpha}_{\mathcal{F}[q']}(\sigma \cdot b)$.

□

5.7.3 More weighted fault models from fault automata

We like to stress that the finite depth and discounted models are just two examples for deriving WFMs from fault automata, but there are many more possibilities. For instance, one may combine the two and do not discount the weights of traces of length less than some k, and only discount traces longer than k. Alternatively, one may let the discount factor depend on the length of the trace. We claim that the methods and algorithms we present in this chapter can easily be adapted for WFMs with such variations.

5.8 Algorithms to compute and optimize coverage

This section presents various algorithms for computing and optimizing coverage for a given FA, interpreted under the finite depth or discounted weighted fault model. In particular, Section 5.8.1 presents algorithms to calculate the absolute coverage in a test suite of a given FA. In Section 5.8.2 we give algorithms that yield the total coverage in a weighted fault model derived from a FA. Section 5.8.4 provides three optimization algorithms for tests with length k. The first one finds a test case with maximal coverage; the second one finds the n test cases with maximal coverage; and the third one finds a test suite with n test cases with maximal coverage, i.e. the best n test cases with minimum overlap.

We use the following notation. Recall that $\mathcal{F}[q]$ denotes the FA that is the same as $\mathcal{F}$, but with q as initial state. When $\mathcal{F}$ is clear from the context, we write respectively f^{k}_{q} and f^{α}_{q} for the weighted fault models $f^{k}_{\mathcal{F}[q]}$ and $f^{\alpha}_{\mathcal{F}[q]}$ derived from $\mathcal{F}$. Moreover, given $\mathcal{F} = \langle A, r\rangle$ an FA, we write $A_{\mathcal{F}}$ for the multi-adjacency matrix of A. That is, $A_{\mathcal{F}}$ contains at position (q, q') the number of edges between q and q', then

$$(A_{\mathcal{F}})_{qq'} = \sum_{l:(q,l,q')\in T} 1$$

If α is a discount function for $\mathcal{F}$, then $A^{\alpha}_{\mathcal{F}}$ is a weighted version of $A_{\mathcal{F}}$, then

$$(A^{\alpha}_{\mathcal{F}})_{qq'} = \sum_{l\in L} \alpha(q, l, q')$$

We omit the subscript $\mathcal{F}$ if it is clear from the context.

5.8.1 Absolute coverage in a test suite

Given $\mathbf{T}$ a test suite, an $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and either a discounting function α for $\mathcal{F}$ or a number k, we desire to compute

$$\mathsf{abscov}(\mathbf{T}, f) \quad = \quad \mathsf{abscov}(\bigcup_{t\in\mathbf{T}} t, f)$$

where $f = f^k_{\mathcal{F}}$ or $f^\alpha_{\mathcal{F}}$.

Given two tests t and t' and an action l, we write $l{\cdot}t$ for $\{l{\cdot}\sigma \mid \sigma \in t\}$ and $t + t'$ for the union $t \cup t'$. We call a super-test the union of any number of tests.

Now, we can write each test as $t = \varepsilon$; or $t = l \cdot t_1$ in case a is an input; or $t = b_1 \cdot t_1 + \cdots + b_n \cdot t_n$ when $b_1, \cdots, b_n$ are all output actions of $\mathcal{F}$. Each super-test can be written as $a_1 \cdot t'_1 + \cdots + a_m \cdot t'_m + b_1 \cdot t''_1 + \cdots + b_n \cdot t''_n$ where a_z are inputs and b_j are all outputs and t'_z, t''_j are super-tests.

To compute the union $\bigcup_{t\in\mathbf{T}} t$, we recursively merge all tests in $\mathbf{T}$ into a super-test using the infix operator $\uplus$, then we add the error weights of all traces in $\bigcup_{t\in\mathbf{T}} t$ via the function ac defined below.

Tests merge Let $t' = a_1 \cdot t'_1 + \cdots + a_m \cdot t'_m + b_1 \cdot t''_1 + \cdots + b_n \cdot t''_n$ be a super-test and t be a test. Then $t = \varepsilon$ or $t = a \cdot t_1$ or $t = b_1 \cdot t'_1 + \cdots + b_n \cdot t'_n$. Then, we define

$$t' \uplus t = \begin{cases} a_1 t'_1 + \cdots + a_j(t'_j \uplus t_1) + \cdots + a_m t'_m + b_1 t''_1 + \cdots + b_n t''_n & t = a t_1 \wedge a = a_j \\ a_1 t'_1 + \cdots + a_m t'_m + b_1(t''_1 \uplus t_1) + \cdots + b_n(t''_n \uplus t_n) & t = b_1 t'_1 + \cdots + b_n t'_n \\ t' + t & \text{otherwise} \end{cases}$$

Absolute coverage in a super-test Given a super-test t of $\mathcal{F}$ and a state q on $\mathcal{F}$, then

$$\begin{aligned} \mathsf{ac}(\varepsilon, q) &= 0 \\ \mathsf{ac}(t, q) &= \sum_{z=1}^{n} aux(l_z \cdot t_z, q) \end{aligned}$$

$$\text{where} \quad aux(l_z \cdot t_z, q) \quad = \quad \begin{cases} \alpha(q, l_z, \delta(q, l_z)) \cdot \mathsf{ac}(t_z, \delta(q, l_z)) & l_z \in \delta(q) \\ r(l_z, q) & \text{otherwise} \end{cases}$$

The correctness of this algorithm is stated in the following theorem.

Theorem 5.8.1. *Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, a state $q \in Q$, a number $k \in \mathbb{N}$, a function $\alpha : Q \times L \times Q \to [0, 1]$ and $\mathbf{T}$ a test suite, then*

- *if α is a discount function for $\mathcal{F}$ then $\mathsf{abscov}(\mathbf{T}, f^\alpha_q) = \mathsf{ac}(\uplus\mathbf{T}, q)$*
- *if $k \geq \max_{t\in\mathbf{T}} |\,t\,|$ and $\alpha(q, l, q') = 1$ for all transitions (q, l, q') in $\mathcal{F}$, then* $\mathsf{abscov}(\mathbf{T}, f^k_q) = \mathsf{ac}(\uplus\mathbf{T}, q)$

Where we write $\uplus\{t_1, t_2, \ldots t_n\}$ meaning $t_1 \uplus t_2 \uplus \ldots t_n$.

5.8.2 Total coverage

Total coverage in discounted FA Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, a state $q \in Q$ and a discounting function α for $\mathcal{F}$, we desire to calculate

$$\mathsf{totcov}(f_q^\alpha) \quad = \quad \sum_{\sigma \in L^*} f_q^\alpha(\sigma)$$

We assume that from each state in $\mathcal{F}$ we can reach at least one error state, then

$$\forall\, q \in Q : \exists\, q' \in \mathsf{reach}_{\mathcal{F}[q]} : \overline{r}(q) > 0$$

In this way, f_q^α is a WFM for every state q.

The basic idea behind the computation method is that the function $\mathsf{tc} : Q \to [0,1]$ (for the total coverage) given by $q \mapsto \mathsf{totcov}(f_q^\alpha)$ satisfies the following set of equations:

$$\mathsf{tc}(q) \quad = \quad \overline{r}(q) + \sum_{l \in L, q' \in Q} \alpha(q,l,q') \cdot \mathsf{tc}(q') \quad = \quad \overline{r}(q) + \sum_{q' \in Q} A^\alpha_{qq'} \cdot \mathsf{tc}(q')$$

These equations express that the total coverage in state q equals the weight $\overline{r}(q)$ of all immediate errors in q, plus the weights in all successors q' in q, discounted by $\sum_{l \in L} \alpha(q,l,q')$.

The following proposition gives an alternative recursive characterization of $\mathcal{F}$.

Proposition 5.8.2. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α be a discount function for $\mathcal{F}$, then*

$$f_q^\alpha(l{\cdot}\sigma) \quad = \quad \begin{cases} \sum_{q' \in Q} \alpha(q,l,q') \cdot f_{q'}^\alpha(\sigma) & l \in T(q) \\ r(q,l) & l \notin T(q) \wedge \sigma = \varepsilon \\ 0 & \textit{otherwise} \end{cases}$$

where $l \in T(q)$ means that there exists q' such that $(l,q') \in T(q)$.

Proposition 5.8.3. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α be a discount function for $\mathcal{F}$. Then the function $\mathsf{tc} : Q \to [0,1], q \mapsto \mathsf{totcov}(f_q^\alpha)$ satisfies the following set of equations*

$$\mathsf{tc}(q) \quad = \quad \overline{r}(q) + \sum_{l \in L, q' \in Q} \alpha(q,l,q') \cdot \mathsf{tc}(q')$$

Proof.

$$\begin{aligned} \mathsf{tc}(q) \quad &= \quad \sum_{\sigma \in L^*} f_q^\alpha(\sigma) \\ &= \quad f_q^\alpha(\varepsilon) + \sum_{l \notin T(q)} f_q^\alpha(l) + \sum_{l \notin T(q), \sigma \in L^+} f_q^\alpha(l{\cdot}\sigma) + \sum_{l \in T(q), \sigma \in L^*} f_q^\alpha(l{\cdot}\sigma) \end{aligned}$$

$$
\begin{array}{lll}
 & \text{(Proposition 5.8.2)} & \\
= & 0 + \sum\limits_{l \notin T(q)} r(q,l) + 0 + \sum\limits_{l \in T(q), q' \in Q, \sigma \in L^*} \alpha(q,l,q') \cdot f^{\alpha}_{q'}(\sigma) & \\
= & \overline{r}(q) + \sum\limits_{l \in L, q' \in Q} \alpha(q,l,q') \cdot \mathsf{tc}(q') &
\end{array}
$$

□

Using matrix-vector notation, we obtain

$$\mathsf{tc} \quad = \quad \overline{r} + A^{\alpha} \cdot \mathsf{tc}$$

The next Proposition 5.8.4 states that the matrix $I - A^{\alpha}$ is invertible.

Proposition 5.8.4. *Let $\mathcal{F} = \langle A, r \rangle$ be an FA with $A = \langle Q, q^0, L, T \rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, such that for all states $q \in Q$ there is a state $q' \in \mathsf{reach}_{\mathcal{F}[q]}$ with $\overline{r}(q) > 0$. Let α be a discount function for $\mathcal{F}$. Then, the matrix $I - A^{\alpha}$ is invertible.*

Proof.
By reordering the states we can obtain $\mathsf{Inf}_{\mathcal{F}} = \{q_1, \ldots, q_{n_1}\}$ and $V_{\mathcal{F}} \backslash \mathsf{Inf}_{\mathcal{F}} = \{q_{n_1+1}, \ldots, q_{n_1+n_2}\}$ with $n_1 + n_2 = n = |Q_{\mathcal{F}}|$. Without loss of generality we may therefore assume that A^{α} is of the form

$$\begin{pmatrix} B & C \\ 0 & D \end{pmatrix}$$

with B the $n_1 \times n_1$ matrix that is the restriction of A^{α} to $\mathsf{Inf}_{\mathcal{F}}$, and D is the restriction of A^{α} to $V_{\mathcal{F}} \backslash \mathsf{Inf}_{\mathcal{F}}$. It follows that $I_{(n)} - A^{\alpha}$ is invertible if and only if $I_{(n_1)} - B$ and $I_{(n_2)} - D$ are invertible.

We first show that $\|B \cdot v\|_{\infty} < \|v\|_{\infty}$ for all $v \neq 0$, where $\|v\|_{\infty} = \max\limits_{z}\{v_z\}$ denotes the supremum norm of v.

Assume $v \neq 0$ and consider the z^{th} component $(B \cdot v)_z$ of the vector $B \cdot v$

$$
\begin{array}{lll}
(B \cdot v)_z & = & \sum\limits_{j \leqslant n_1} B_{zj} \cdot v_j \\
 & \leq & \sum\limits_{j \leqslant n_1} B_{zj} \cdot \|v\|_{\infty} \\
 & = & \|v\|_{\infty} \cdot \sum\limits_{(j,a) \in \mathit{OutInf(z)}} \alpha(z,a,j) \quad \text{(Definition 5.6.3)} \\
 & < & \|v\|_{\infty}
\end{array}
$$

Hence, $\|B \cdot v\|_{\infty} < \|v\|_{\infty}$. Therefore $B \cdot v \neq v$, so $(I - B) \cdot v \neq 0$ for $v \neq 0$, which yields that $I - B$ is invertible.

Without loss of generality we can also assume that the states have been numbered such that for $z, j \in Q_{\mathcal{F}} \backslash \mathsf{Inf}_{\mathcal{F}}$ $(z,l,j) \in \delta_{\mathcal{F}}$ implies $z < j$. It follows that $D_{zj} = 0$ for all

$1 < j \leqslant z < n_2$, and that $(I - D)_{zj} = 0$ for all $1 < j < z < n_2$ with $(I - D)_{zz} = 1$ for all $1 < z < n_2$. We can conclude that $\det(I - D) = 1 \neq 0$, and thus that $I - D$ is invertible. □

Theorem 5.8.5. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, such that for all $q \in Q$ there exists a state $q' \in \mathsf{reach}_{\mathcal{F}[q]}$ with $\overline{r}(q') > 0$, and let α be a discount function for $\mathcal{F}$, then*

$$\mathsf{tc} \quad = \quad (I - A^\alpha)^{-1} \cdot \overline{r}$$

The time complexity of the method above is dominated by matrix inversion, which can be computed in $O(|Q|^3)$ with Gaussian elimination, in $O(|Q|^{log_2 7})$ with Strassen's method.

Example 5.8.6. *Given $\mathcal{F}$ the FA from Figure 5.3 and a discount function $\alpha = \{(q_0, \delta, q_0, \frac{1}{5}), (q_0, \textit{ask\_money?}, q_1, \frac{1}{3}), (q_1, \textit{ask\_money?}, q_1, \frac{1}{4}), (q_1, \textit{give\_money!}, q_0, \frac{1}{2})\}$, then*

$$\begin{array}{lcll} \mathsf{tc}(q_0) & = & \overline{r}(q_0) + \alpha(q_0, \delta, q_0) \cdot \mathsf{tc}(q_0) & + \quad \alpha(q_0, \textit{ask\_money?}, q_1) \cdot \mathsf{tc}(q_1) \\ \mathsf{tc}(q_1) & = & \overline{r}(q_1) + \alpha(q_1, \textit{give\_money!}, q_0) \cdot \mathsf{tc}(q_0) & + \quad \alpha(q_1, \textit{ask\_money?}, q_1) \cdot \mathsf{tc}(q_1) \end{array}$$

In matrix notation:

$$\begin{array}{rcl} \mathsf{tc} & = & \overline{r} + (A^\alpha \cdot \mathsf{tc}) \\ \mathsf{tc} - (A^\alpha \cdot \mathsf{tc}) & = & \overline{r} \\ (I - A^\alpha) \cdot \mathsf{tc} & = & \overline{r} \\ \mathsf{tc} & = & (I - A^\alpha)^{-1} \cdot \overline{r} \end{array}$$

Then, with matrix A^α, matrix $I - A^\alpha$ and $(I - A^\alpha)^{-1}$ equal to

$$A^\alpha = \begin{bmatrix} \frac{1}{5} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{4} \end{bmatrix}, \quad I - A^\alpha = \begin{bmatrix} \frac{4}{5} & -\frac{1}{3} \\ -\frac{1}{2} & \frac{3}{4} \end{bmatrix}, \quad (I - A^\alpha)^{-1} = \begin{bmatrix} \frac{45}{26} & \frac{10}{13} \\ \frac{15}{26} & \frac{24}{13} \end{bmatrix}$$

and $\overline{r}$ as the one given in Figure 5.3: $\overline{r} = [10, 5]$, we obtain $\mathsf{tc} = 21.15384616$.

Total coverage in finite depth FA Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, a state $q \in Q$ and a depth $k \in \mathbb{N}$, we desire to compute

$$\mathsf{totcov}(f_q^k) \quad = \quad \sum_{\sigma \in L^*} f_q^k(\sigma)$$

We assume that from each state, there is at least one error reachable in k steps,

$$\forall\, q \in Q : \exists\, q' \in \mathsf{reach}^k_{\mathcal{F}[q]} : \overline{r}(q') > 0$$

This makes that f_q^k is a weighted fault model for any q.

The basic idea behind the computation method is that the function $\mathsf{tc}_k : Q \to [0,1]$ given by $q \mapsto \mathsf{totcov}(f_q^k)$ satisfies the following recursive equations:

$$\begin{array}{lcl} \mathsf{tc}_0(q) & = & 0 \\ \mathsf{tc}_{k+1}(q) & = & \overline{r}(q) + \sum\limits_{(a,q')\in T(q)} \mathsf{tc}_k(q') \\ & = & \overline{r}(q) + \sum\limits_{a\in L, q'\in Q} A_{q,q'} \cdot \mathsf{tc}_k(q') \end{array}$$

The following proposition gives an alternative recursive characterization of f_q^k; it is the analogon in the finite depth model from Proposition 5.8.2.

Proposition 5.8.7. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and $k \in \mathbb{N}$ be a number, then*

$$f_q^k(l\cdot\sigma) \quad = \quad \begin{cases} \sum\limits_{q':(l,q')\in T(q)} f_{q'}^{k-1}(\sigma) & l \in T(q) \wedge |\sigma| \leq k \\ r(q,l) & l \notin T(q) \wedge \sigma = \varepsilon \\ 0 & \textit{otherwise} \end{cases}$$

Proposition 5.8.8. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α a discount function for $\mathcal{F}$. Then the function $\mathsf{tc}_k : Q \to [0,1], q \mapsto \mathsf{totcov}(f_q^k)$ satisfies the following set of equations*

$$\mathsf{tc}_k(q) \quad = \quad \overline{r}(q) + \sum_{(l,q')\in T(q)} \mathsf{tc}_{k-1}(q')$$

Proof.
As the proof of Proposition 5.8.3.

□

In matrix-vector notation, we have

$$\begin{array}{lcl} \mathsf{tc}_0 & = & 0 \\ \mathsf{tc}_{k+1} & = & \overline{r} + A \cdot \mathsf{tc}_k \end{array}$$

Theorem 5.8.9. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and for all state $q \in Q$ exists $q' \in \mathsf{reach}^k_{\mathcal{F}[q]}$ such that $\overline{r}(q') > 0$. Let q be a state in Q and k be in $\mathbb{N}$, then*

$$\mathsf{tc}_k \quad = \quad \sum_{z=0}^{k-1} A^z \cdot \overline{r}$$

Note that for a state q in an arbitrary $\mathcal{F}$, there exists a state $q' \in \mathsf{reach}^k_{\mathcal{F}[q]}$ with $\overline{r}(q') > 0$ if

and only if $\left(\sum_{z=0}^{k-1} A^z \cdot \overline{r}\right)_q > 0$.

Using Theorem 5.8.9 with sparse matrix multiplication, or iterating the equations just above it, tc_k can be computed in time $O(k \cdot |T| + |Q|)$.

Example 5.8.10. *Given $\mathcal{F}$ the FA from Figure 5.3 and $k = 2$ the matrix A, becomes*

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \textit{and} \quad \mathsf{tc}_k = \begin{bmatrix} 10 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 5 \end{bmatrix}$$

Then, we calculate $\mathsf{tc}_k = 25$.

A similar method to the one above can be used to compute the weight of all tests of length k in the discounted weighted fault model, $\mathsf{abscov}(\mathbf{T}^k, f_q^\alpha)$, where $\mathbf{T}^k$ is the set of all tests of length k in $\mathcal{F}$.

Writing $tcd_k(q) = \mathsf{abscov}(\mathbf{T}^k, f_q^\alpha)$ (for the total coverage discounted), the recursive equations become

$$\begin{array}{lcl} tcd_0(q) & = & 0 \\ tcd_{k+1}(q) & = & \overline{r}(q) + \sum\limits_{l \in L, q' \in Q} tcd_k(q') \\ & = & \overline{r}(q) + \sum\limits_{l \in L, q' \in Q} A^\alpha_{qq'} \cdot tcd_k(q') \end{array}$$

and the analogon of Theorem 5.8.9 becomes:

$$\begin{array}{lcl} tcd_k & = & \sum\limits_{z=0}^{k-1} (A^\alpha)^z \cdot \overline{r} \\ & = & (I - A^\alpha)^{-1} \cdot (I - (A^\alpha)^k) \cdot \overline{r} \end{array}$$

The latter equality holds because $I - A^\alpha$ is invertible.

The computing of tcd_k requires one matrix inversion and, using the power method, $\log_2(k)$ matrix multiplications, we have time complexity equal to $O(|Q|^{\log_2 7} + |Q|^{\log_2(k)})$ with Strassen's method. If $(I - A^\alpha)$ can be put in diagonal form, the problem can be solved in $O(|Q|^3 + \log_2 n)$. These tricks cannot be applied in the finite depth model, because $I - A$ is not invertible; since A has row sum 1, we have for the vector $\mathbf{1}$ whose entries are all equal to 1 that $A \cdot \mathbf{1} = \mathbf{1}$. Hence, $\mathbf{1}$ is in the kernel of $I - A$, so $I - A$ is not invertible.

Example 5.8.11. *Given $\mathcal{F}$ our FA from Figure 5.3, $k = 2$, matrix A^α, matrix $I - A^\alpha$, $(I - A^\alpha)^{-1}$, and $(A^\alpha)^k$ are equal to*

$$A^\alpha = \begin{bmatrix} \frac{1}{5} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{4} \end{bmatrix} \qquad I - A^\alpha = \begin{bmatrix} \frac{4}{5} & -\frac{1}{3} \\ -\frac{1}{2} & \frac{3}{4} \end{bmatrix}$$

$$(I - A^\alpha)^{-1} = \begin{bmatrix} \frac{45}{26} & \frac{10}{13} \\ \frac{15}{26} & \frac{24}{13} \end{bmatrix} \qquad (A^\alpha)^k = \begin{bmatrix} \frac{31}{150} & \frac{3}{20} \\ \frac{9}{40} & \frac{11}{48} \end{bmatrix}$$

Here, we calculate $tcd_k = 13.66666667$.

5.8.3 Relative coverage

Combining the algorithms for computing total and absolute coverage from the previous sections, we can compute easily $\mathsf{relcov}(\mathbf{T}, f) = \frac{\mathsf{abscov}(\mathbf{T},f)}{\mathsf{totcov}(f)}$ for a test suite $\mathbf{T}$ and $f = f_q^k$ or $f = f_q^\alpha$.

5.8.4 Optimization

Optimal coverage in a test case Given $\mathcal{F} = \langle A, r\rangle$ an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and k a length, we compute the best test case with length k, i.e. the test with highest coverage. We treat the finite depth and discounted model at once by fixing, in the finite depth model

$$\alpha(q, l, q') = \begin{cases} 1 & (q, l, q') \in T \\ 0 & \text{otherwise} \end{cases}$$

We call this function α an *extended discount function* if it is a discount function or it is obtained from a finite depth model.

The optimization method is again based on recursive equations. We write

$$acopt_k(q) = \max_{t \in TESTS^k} \{\mathsf{abscov}(t, f_q^\alpha)\}$$

for the optimal absolute coverage. To understand the recursive characterization of $acopt_k$, we consider two situations. Firstly, we consider a test case of length $k+1$ that in state q applies an input $a?$ and in the successor state q' applies the optimal test of length k. The (absolute) coverage of this test case is $\alpha(q, a?, q') \cdot acopt_k(q')$. The best coverage that we can obtain by stimulating the IUT is given by

$$\max_{(a?,q') \in T^I(q)} \alpha(q, a?, q') \cdot acopt_k(q')$$

Secondly, we consider the test case of length $k+1$ that in state q observes the IUT and in each successor state q' applies the optimal test of length k. The coverage of this test case is

$$\overline{r}(q) + \sum_{(b!,q') \in T^O(q)} \alpha(q, b!, q') \cdot acopt_k(q')$$

Now, the optimal test $acopt(q)$ of length $k+1$ is obtained from $acopt_k$ by selecting from these options (i.e. inputing an action $a?$ or observing) the one with the highest coverage.

Proposition 5.8.12. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and let α be a discount function for $\mathcal{F}$ and $k \in \mathbb{N}$*

1. *Let q be a state, let $(a?, q') \in T^I(q)$, and let t' be a test case in states q'. We write t for the test case $t = \{a?\cdot\sigma \mid \sigma \in t'\}$. Then*

$$\begin{array}{lcll} \mathsf{abscov}(t, f_q^\alpha) & = & \alpha(q, a, q') \cdot \mathsf{abscov}(t', f_{q'}^\alpha) & \\ \mathsf{abscov}(t, f_q^k) & = & \mathsf{abscov}(t', f_{q'}^{k-1}) & \mid t \mid \leq k+1 \end{array}$$

2. *Let q be a state and $T^O(q) = \{(b_1!, q_1), (b_2!, q_2) \ldots (b_n!, q_n)\}$, where all $b_z!$ are different. Also, write $O\backslash\{b_1, \ldots b_n\} = \{c_1, c_2, \ldots c_m\}$. Let $t_1, t_2, \ldots t_n$ be test cases in states $q_1 \ldots q_n$ respectively. Write t for the test case $t = \{b_z!\cdot\sigma \mid \sigma \in t_z\} \cup \{c_1, c_2, \ldots c_m\}$. Then*

$$\begin{array}{lcll} \mathsf{abscov}(t, f_q^\alpha) & = & \overline{r}(q) + \sum_{z=1}^{n} \alpha(q, b_z, q_z) \cdot \mathsf{abscov}(t_z, f_{q_z}^\alpha) & \\ \mathsf{abscov}(t, f_q^k) & = & \overline{r}(q) + \sum_{z=1}^{n} \mathsf{abscov}(t_z, f_{q_z}^k) & \mid t \mid \leq k+1 \end{array}$$

Proof.
We give the proof for f_q^α; the one for f_q^k is similar.

1. $$\begin{array}{lcll} \mathsf{abscov}(t, f_q^\alpha) & = & \sum_{\sigma \in t} f_q^\alpha(\sigma) & \\ & = & \sum_{\sigma' \in t'} f_q^\alpha(a\cdot\sigma') & \text{(Proposition 5.8.2)} \\ & = & \sum_{\sigma' \in t'} \alpha(q, a, q') \cdot f_{q'}^\alpha(\sigma') & \\ & = & \alpha(q, a, q') \cdot \mathsf{abscov}(t', f_{q'}^\alpha) & \end{array}$$

2. $$\begin{array}{lcll} \mathsf{abscov}(t, f_q^\alpha) & = & \sum_{\sigma \in t} f_q^\alpha(\sigma) & \\ & = & \sum_{c \notin T(q)} f_q^\alpha(c) + \sum_{z=1}^{n} \sum_{\sigma' \in t_z} f_q^\alpha(b_z\cdot\sigma') & \text{(Proposition 5.8.2)} \\ & = & \overline{r}(q) + \sum_{z=1}^{n} \sum_{\sigma' \in t_z} \alpha(q, b_z, q_z) \cdot f_{q_z}^\alpha(\sigma') & \\ & = & \overline{r}(q) + \sum_{z=1}^{n} \alpha(q, b_z, q_z) \cdot \mathsf{abscov}(t_z, f_{q_z}^\alpha) & \end{array}$$

□

Thus, we obtain the following result, which follows from previous Proposition 5.8.12.

Theorem 5.8.13. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, α be an extended discount function, and $k \in \mathbb{N}$ be a test length. Then $acopt_k$ satisfies the following recursive equations*

$$\begin{aligned} acopt_0(q) &= 0 \\ acopt_{k+1}(q) &= \max\Big(\overline{r}(q) + \sum_{(b!,q')\in T^O(q)} \alpha(q, b!, q') \cdot acopt_k(q'), \\ &\qquad \max_{(a?,q')\in T^I(q)} \alpha(q, a?, q') \cdot acopt_k(q')\Big) \end{aligned}$$

Based on Theorem 5.8.13, we can compute $acopt_k$ in time $O(k(|Q| + |T|))$.

Shortest test case with high coverage We can use the above method not only to compute the test case of a fixed length k with optimal coverage, but also to derive the shortest test case with coverage higher than a given bound κ. We iterate the equations in Theorem 5.8.13 and stop as soon as we achieve coverage higher than κ, i.e. at the first n with $acopt_n(q) > \kappa$.

We have to take care that the bound κ is not too high, i.e. higher than what is achievable with a single test case. In the finite depth model, this is easy: if the test length is the same as κ then we can stop, since this is the longest test we can have. In the discounted model, however, we have to ensure that κ is strictly smaller than the supremum of the coverage of all tests in single test case.

Let $\mathsf{mw}(q) = \underset{t\in TESTS}{supp} \{\mathsf{abscov}(t, q)\}$, i.e. the maximal absolute weight of a single test case. Then mw is again characterized by a set of equations.

Theorem 5.8.14. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α be a discount function for $\mathcal{F}$. Then* mw *is the unique solution of the following set of equations*

$$\mathsf{mw}(q) = \max\left(\overline{r}(q) + \sum_{(b!,q')\in T^O(q)} \alpha(q, b!, q') \cdot \mathsf{mw}(q'), \max_{(a?,q')\in T^I(q)} \alpha(q, a?, q') \cdot \mathsf{mw}(q')\right)$$

The solution of these equations can be found by linear programming (LP).

Theorem 5.8.15. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, and α be a discount function. Then* mw *is the optimal solution of the following LP problem : minimize $\sum_{q\in Q} \mathsf{mw}(q)$ subject to*

$$\begin{aligned} &\mathsf{mw}(q) \geq \alpha(q, a?, q') \cdot \mathsf{mw}(q') && (a?, q') \in T^I(q) \\ &\mathsf{mw}(q) \geq r(q) + \sum_{(b!,q')\in T^O(q)} \alpha(q, b!, q') \cdot \mathsf{mw}(q') && q \in Q \end{aligned}$$

The above LP problem contains $|Q|$ variables and $|Q| + |T^I|$ inequalities. Thus, solving

this problem is polynomial in $|Q|$, $|Q| + |T^I|$ and the length of the binary encoding of the coefficients [57].

Optimal coverage in n test cases The first algorithm in this section for computing the best test case of length k can be extended to a method for computing the best n test cases with optimal coverage: the previous algorithm picks the best test case with length k. To pick the second best test case, we apply the same procedure, except that we exclude the first choice from all possible options; for the third best choice, we exclude the previous two, and so on.

Optimal coverage in a test suite (with n test cases) Differently from the previous algorithm where the n chosen tests may overlap, we now present an algorithm to compute the best coverage in a test suite with n tests. In this test suite, we avoid test overlapping. Avoiding overlapping after we combined the tests in a super-test we obtain the test suite with optimal coverage. The idea is the following, we write

$$\mathsf{acopt}^n_k(q) \quad = \quad \underset{t_1,t_2,\ldots t_n \in TESTS^k}{\mathsf{maxn}} \{\mathsf{abscov}(TESTS, q)\}$$

for the ordered list $[l_1, l_2, \ldots l_n]$, where l_z is the coverage of the z^{th} best test of length k. We characterize acopt^n_k recursively. To do this we start dividing our reasoning in two sets: a test suite for inputs and a test suite for outputs, and later we combine them.

Assume the input actions are $a_1, a_2, \ldots a_m$. Let $\mathbf{T}$ be a test suite started in state q such that $t = \{a_1 \cdot \mathbf{T}_1, \cdots, a_m \cdot \mathbf{T}_m\}$. To compute $\mathsf{acopt}^n_k(q)$ of the test suite $\mathbf{T}$ we assume that there exists the set of n best test cases that start on $q_z : \mathsf{acopt}^n_{k-1}(q_z)$ for all $0 \leq z \leq m$ (where q_z is the state reachable from s after the input action a_z). Then, let $\mathsf{acopt}^n_{k-1}(q_z)$ be equal to $[l'_1, l'_2, \ldots l'_n]$ where l'_1 is the optimal coverage of a test started in q_z, l'_2 is the second optimal coverage of a test started from q_z, and so on.

$$\begin{array}{rcl} \mathsf{acopt}^n_{k-1}(q_z) & = & [l'_1, l'_2, \ldots l'_n] \\ & = & [(\mathsf{acopt}^n_{k-1}(q_z))_1, \cdots, (\mathsf{acopt}^n_{k-1}(q_z))_n] \end{array}$$

$$\begin{array}{rcl} \mathsf{acopt}^n_k(q) & = & \mathsf{maxn}\,[\alpha(q, a_1, q_1)(\mathsf{acopt}^n_{k-1}(q_1))_1, \cdots, \alpha(q, a_1, q_1)(\mathsf{acopt}^n_{k-1}(q_1))_n, \cdots \\ & & \alpha(q, a_m, q_m)(\mathsf{acopt}^n_{k-1}(q_m))_1, \cdots, \alpha(q, a_m, q_m)(\mathsf{acopt}^n_{k-1}(q_m))_n] \\ & = & \mathsf{maxn}\,[\alpha(q, a, q') \cdot l \mid (a?, q') \in T^I(q) \wedge l \leftarrow (\mathsf{acopt}^n_{k-1}(q'))_j \\ & & \wedge\, 0 \leq j \leq n] \end{array}$$

For the case of outputs. Assume the outputs actions are $b_1, b_2, \ldots b_p$. Let $\mathbf{T}$ be a test suite started in state q such that $\mathbf{T} = \{b_1 \cdot \mathbf{T}_1, \cdots, b_p \cdot \mathbf{T}_p\}$. To compute $\mathsf{acopt}^n_k(q)$ of the test suite $\mathbf{T}$, again, we assume that there exists the set of n best test cases that start on q_z: $\mathsf{acopt}^n_{k-1}(q_z)$ for all $0 \leq z \leq m$ (where q_z is the state reachable from q after an output action b_z). Also, let $\mathsf{acopt}^n_{k-1}(q_z)$ be equal to $[l'_1, l'_2, \ldots l'_n]$ where l'_1 is the optimal coverage of a test started in q_z, and so on.

$$\mathsf{acopt}^n_{k-1}(q_z) \quad = \quad [l'_1, l'_2, \ldots, l'_n]$$

$$= [(\mathsf{acopt}^n_{k-1}(q_z))_1, \cdots, (\mathsf{acopt}^n_{k-1}(q_z))_n]$$

$$\begin{array}{rcl}
\mathsf{acopt}^n_k(q) & = & [\overline{r}(q)+\alpha(q,b_1,q_1)(\mathsf{acopt}^n_{k-1}(q_1))_1+\cdots+\alpha(q,b_p,q_p)(\mathsf{acopt}^n_{k-1}(q_n))_n, \cdots \\
& & \overline{r}(q)+\alpha(q,b_1,q_1)(\mathsf{acopt}^n_{k-1}(q_1))_n+\cdots+\alpha(q,b_p,q_p)(\mathsf{acopt}^n_{k-1}(q_p))_n] \\
& = & \overline{r}(q) \oplus [\sum_{(b!,q')\in T^O(q)} \alpha(q,b,q')\cdot l \mid l \leftarrow (\mathsf{acopt}^n_{k-1}(q'))_j \wedge 0 \leq j \leq n]
\end{array}$$

Where, $x \oplus l$ adds the number $x \in \mathbb{R}^{\geq 0}$ to each element of the list l (i.e., $x \oplus [e_1, e_2, \ldots e_n] = [x+e_1, x+e_2, \ldots x+e_n]$). Moreover, maxn yields the n maximal elements in a list. By keeping the lists sorted (largest element first) we can efficiently implement the algorithm. To do so, it suffices that maxn returns a sorted list.

Theorem 5.8.16. *Let $\mathcal{F} = \langle A, r\rangle$ be an FA with $A = \langle Q, q^0, L, T\rangle$, $r : Q \times O \to \mathbb{R}^{\geq 0}$, α a discount function for $\mathcal{F}$, $k \in \mathbb{N}$ be a test length and $n \in \mathbb{N}$ be a number. Then tc_k satisfies the following equations*

$$\begin{array}{rcl}
v_0(q) & = & [0,0,\ldots,0] \\
v_{k+1}(q) & = & \mathsf{maxn}\{ [\alpha(q,a,q')\cdot v \mid (a?,q') \in T^I(q), v \leftarrow (v_k(q'))_j \wedge 0 \leq j \leq n] \\
& & ++\, r(q) \oplus [\sum_{(b!,q')\in T^O(q)} \alpha(q,b,q')\cdot l \mid l \leftarrow (v_k(q'))_j \wedge 0 \leq j \leq n]\}
\end{array}$$

Example 5.8.17. *This example shows the difference between the last two algorithms: optimal coverage in n test cases and optimal coverage in a test suite (with n test cases).*

In Figure 5.6 upper part we see four tests with length 3. Test t_1 is the test with optimal coverage in length 3, it value is 30 (for any f^k_q with $k > 3$). Any of the next three tests (t_2, t_3 and t_4) can be the second test with optimal coverage, given that all of them have value 20. Then, using the optimal coverage in n test cases algorithm, we can choose t_1 as the optimal test case and t_2 as the second optimal test case. But, as we appreciate in the button part in Figure 5.6 the test suite $\mathbf{T}$ with $\mathbf{T} = \{t_1, t_2\}$ is not the test suite with optimal coverage (it has value 40). The test suite with optimal coverage is $\mathbf{T} = \{t_1, t_4\}$ (with value 50). This is because to compute the value of a test suite we first combined the tests and then we compute the value. Thus, we obtain a tests suite with optimal coverage if we choose test that have less overlap between them, as is the case with $\mathbf{T} = \{t_1, t_4\}$.

5.9 Application: a chat protocol

This section applies our theory to a practical example, namely a chat protocol, also known as a conference protocol [10]. This protocol provides a multi-cast service to users engaged in a chat session. A chat is a group of users that can exchange messages. Each user can send messages to and receive messages from all other partners participating in the same chat session. The chat participants are dynamic, as the chat service allows them to join and leave the chat at any moment in time. Different chats can exist at the same time, but each user can only participate in at most one chat at a time.

The protocol specifies the data units, the underlying service and the chat service. The protocol data units describes the format of the data units that are used by the protocol entities

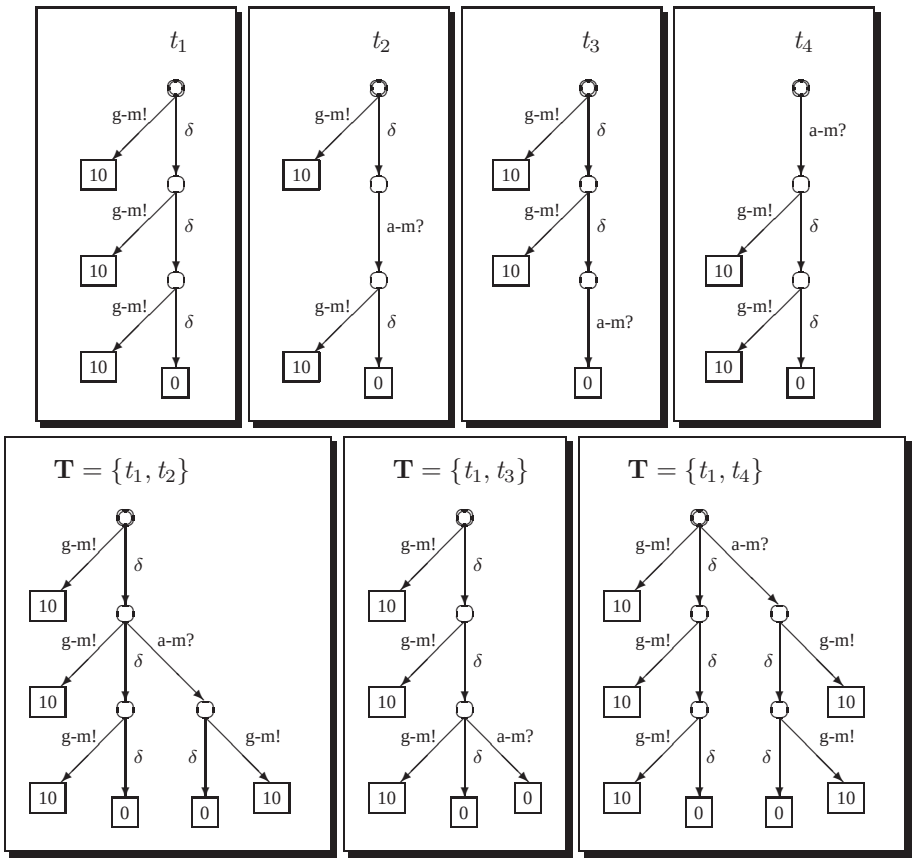


Figure 5.6: We use a_m? for ask_money? and g_m! for give_money!

to communicate with peer entities, the underlying service describes the service of the underlying communication medium through which these data units have to be communicated between peer entities and the behaviour of the protocol entities. Details of all these services can be found in [10]. The chat service is explained as follows. Each chat session has a name. The chat service has the following service primitives (called CSPs), which can be performed at the chat service access points (CSAPs):

- *join*: a user joins a named chat and defines its user title in this session; the user title identifies a user in a chat
- *datareq*: a user sends a message to all other users participating in its session
- *dataind*: a user receives a message from another user participating in its session
- *leave*: a user leaves the chat; since a user can only participate in one chat at a time, there is no need to identify the chat in this primitive.

The service primitives join and leave are used for chat control. The service primitives datareq and dataind are used for data transfer. Initially, a user is only allowed to perform a

join to a chat. After joining, the user is allowed to send messages, by performing datareq's, or to receive messages, by performing dataind's. In order to stop its participation in the chat, a user can a leave at any time after it has done a join.

Data transfer is multi-cast, which means that each datareq causes corresponding dataind's in all other participants in the chat. Data transfer in the chat service is not reliable: messages may get lost, but they never get corrupted; corrupted messages are discarded. Also, the sequence delivery of messages is not guaranteed.

Figure 5.7 displays a LTS model of the chat protocol. This model considers two chat sessions and three users (A, B and C). We consider different weights values per error, depending on the gravity of the error, in Figure 5.8 we present the transition weight function r. Basically, we consider absence of required answers as the worse errors with weight 10; inappropriate answers as less serious with weight 7 and inappropriate joins or leaves as the least severe with weight 3.

We interpret $\mathcal{F}$ as a discounted WSM under different discount functions, α_1, α_2 and α_3. If $\theta = (q, l, q')$ is a transition in $\mathcal{F}$ leaving from state q with out-degree d, we use $\alpha_1(\theta) = \frac{1}{8}$; $\alpha_2(\theta) = \frac{1}{d} - \frac{1}{100}$; and $\alpha_3(\theta) = \frac{1}{d} - \frac{1}{10000}$.

States error values (i.e. $\overline{r}$), out-degree (i.e. d), and the different values of discount function (i.e. $\alpha_1, \alpha_2, \alpha_3$) can be found in Figure 5.9. Figure 5.10 gives the total coverage in $\mathcal{F}$, again, for $\alpha_1, \alpha_2, \alpha_3$. Also, in Figure 5.10 are the absolute and relative coverage of the test suites containing all tests of length k, for $k = 2, 4, 50$ and $\alpha_1, \alpha_2, \alpha_3$. We used Maple 9.5 to resolve the matrix equations in these algorithms.

Figure 5.11 displays the relative coverage for test suites that have been generated automatically with TORX, using discount functions α_2. For tests with lengths $k = 30, 35, 40, 45, 50$, TORX has generated a test suite $\mathbf{T}^k$, consisting of 10 tests $t_1^k, \ldots t_{10}^k$ of length k. Finally, Figure 5.12 lists the coverage for the same test suites $\mathbf{T}^k$ generated by TORX, also using α_2.

The running times of all computations were very small, in the order of a few seconds. Notice the influence of the discount factor and the test length on the coverage numbers.

5.10 Related work

There is a vast literature on syntactic test coverage criteria [8]. Test coverage and optimization are well studied for (extended) finite state machines [61, 42]. Most works consider syntactic coverage measures and optimize preset tests, i.e. find the shortest sequence of inputs to the IUT that achieves a certain coverage.

Test optimization in the adaptive setting is also considered in [49]. Their specification models are Markov Decision Processes, i.e. the tester chooses an input to the IUT and the IUT makes a probabilistic choice among all possible outputs, and assigns a cost to each transition to be executed. This paper provides optimization techniques for deriving test suites with maximal expected coverage for (final) states and transitions at minimal expected cost. Thus, their coverage criteria are syntactic.

The work [13] optimize the order in which a test suite is executed, such that the impact (i.e. the probability that a certain error occurs times its weight) is maximized against total duration, cost and produced quality.

5.11 Conclusions

Semantic notions of test coverage have long been overdue, while they are much needed in the selection, generation and optimization of test suites. In this chapter, we presented semantic coverage notions based on WFMs. We introduced fault automata, FA, to represent syntactically (a subset of) WFMs and provided algorithms to compute and optimize test coverage. This approach is purely semantic since replacing an FA with a semantically equivalent one (i.e. r-preserving bisimilar and α-preserving bisimilar) leaves the coverage unchanged. Our experiments with the chat protocol indicate that our approach is feasible for small protocols. Larger case studies should evaluate the applicability of this framework for more complex systems. In order to do this we are implementing our theory in the SECO tool, standing for Semantic Coverage. This new tool is being developed in Java under Eclipse.

Our weighted fault models are based on (adaptive) **ioco** test theory. We expect to be easy to adapt our approach to different settings, such as FSM testing or on-the-fly testing. Furthermore, our optimization techniques use test length as an optimality criterion. To accommodate more complex resource constraints (e.g time, costs, risks/probability) occurring in practice, it is relevant to extend our techniques with these attributes. Since these fit naturally within our model and optimization problems subject to costs, time and probability are well-studied, we expect that such extensions are both feasible and useful.

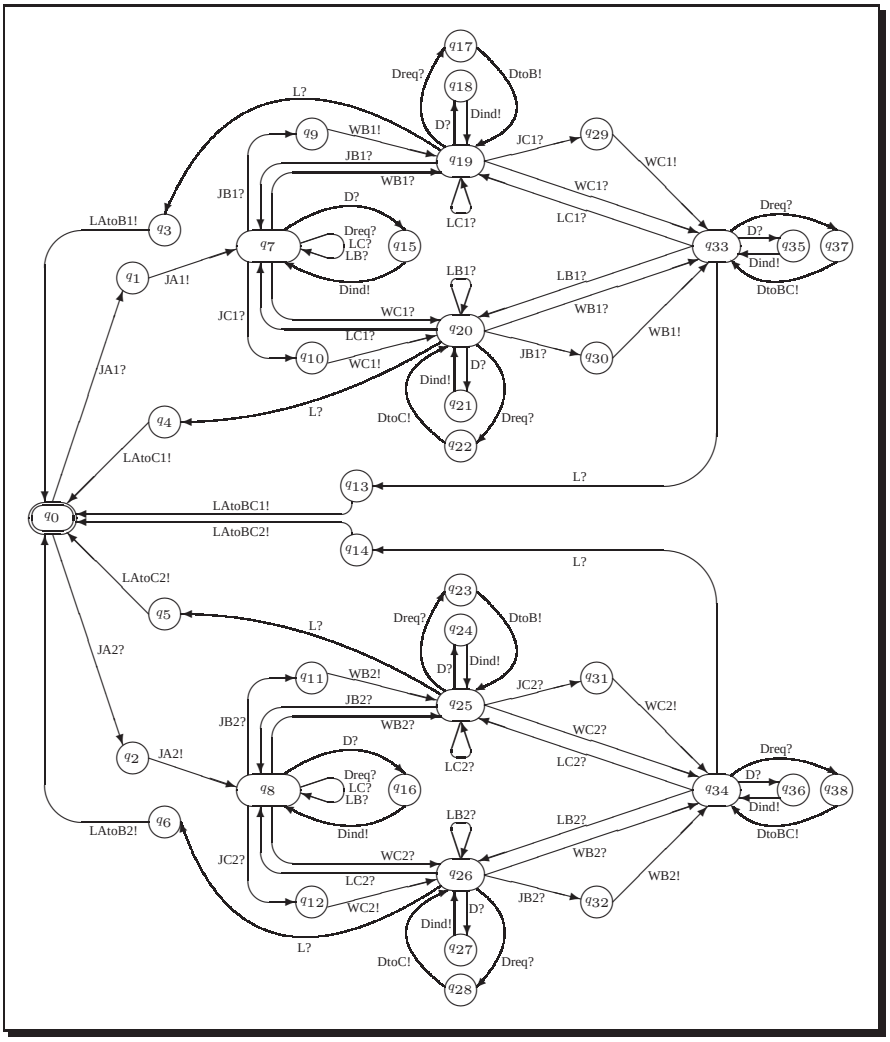


Figure 5.7: Chat protocol with two chats. L = leave, J = join, D = data and W = answer

name of error	value	name of error	value
join.A.1.PDU!(JA1!)	3	leave.A.to.C.2.PDU!(LAtoC2!)	3
join.A.2.PDU!(JA2!)	3	leave.A.to.BC.1.PDU!(LAtoBC1!)	3
answer.B.1!(WB1!)	7	leave.A.to.BC.2.PDU!(LAtoBC2!)	3
answer.B.2!(WB2!)	7	dataind!(Dind!)	3
answer.C.1!(WC1!)	7	data.to.B.PDU!(DtoB!)	3
answer.C.2!(WC2!)	7	data.to.C.PDU!(DtoC!)	3
leave.A.to.B.1.PDU!(LAtoB1!)	3	data.to.BC.PDU!(DtoBC!)	3
leave.A.to.B.2.PDU!(LAtoB2!)	3	quiescent!	10
leave.A.to.C.1.PDU!(LAtoC1!)	3		

Figure 5.8: Error names and error values

state	r	d	α_1	α_2	α_3
q_0	64	3	1/8	0.323	0.333
q_1	71	1	1/8	0.990	0.999
q_2	71	1	1/8	0.990	0.999
q_3	71	1	1/8	0.990	0.999
q_4	71	1	1/8	0.990	0.999
q_5	71	1	1/8	0.990	0.999
q_6	71	1	1/8	0.990	0.999
q_7	64	9	1/8	0.101	0.111
q_8	64	9	1/8	0.101	0.111
q_9	67	1	1/8	0.990	0.999
q_{10}	67	1	1/8	0.990	0.999
q_{11}	67	1	1/8	0.990	0.999
q_{12}	67	1	1/8	0.990	0.999
q_{13}	71	1	1/8	0.990	0.999
q_{14}	71	1	1/8	0.990	0.999
q_{15}	71	1	1/8	0.990	0.999
q_{16}	71	1	1/8	0.990	0.999
q_{17}	71	1	1/8	0.990	0.999
q_{18}	71	1	1/8	0.990	0.999
q_{19}	64	8	1/8	0.115	0.124

state	r	d	α_1	α_2	α_3
q_{20}	64	8	1/8	0.115	0.124
q_{21}	71	1	1/8	0.990	0.999
q_{22}	71	1	1/8	0.990	0.999
q_{23}	71	1	1/8	0.990	0.999
q_{24}	71	1	1/8	0.990	0.999
q_{25}	64	8	1/8	0.115	0.124
q_{26}	64	8	1/8	0.115	0.124
q_{27}	71	1	1/8	0.990	0.999
q_{28}	71	1	1/8	0.990	0.999
q_{29}	67	1	1/8	0.990	0.999
q_{30}	71	1	1/8	0.990	0.999
q_{31}	67	1	1/8	0.990	0.999
q_{32}	67	1	1/8	0.990	0.999
q_{33}	64	6	1/8	0.156	0.166
q_{34}	64	6	1/8	0.156	0.166
q_{35}	71	1	1/8	0.990	0.999
q_{36}	71	1	1/8	0.990	0.999
q_{37}	67	1	1/8	0.990	0.999
q_{38}	71	1	1/8	0.990	0.999

Figure 5.9: The accumulated weights of erroneous outputs ($\overline{r}$), the out-degree (d) and the different discounts α per state

	tc
α_1	99.134
α_2	511.369
α_3	743.432

ack	$k=2$	$k=4$	$k=50$
α_1	89.750	97.171	99.134
α_2	130.607	239.025	510.768
α_3	132.652	249.320	733.540

rck	$k=2$	$k=4$	$k=50$
α_1	91%	98%	100%
α_2	25%	47%	100%
α_3	18%	34%	99%

Figure 5.10: Total coverage (tc) for different discount functions, absolute (ack) and relative coverage (rck) of the test suite containing all tests of length k

	test t_1^k	test t_2^k	test t_3^k	test t_4^k	test t_5^k
$k=30$	15.3%	4.6%	14.0%	5.3%	15.3%
$k=35$	14.1%	15.3%	15.3%	8.5%	8.6%
$k=40$	5.3%	14.0%	14.2%	15.3%	5.3%
$k=45$	5.0%	8.5%	14.0%	5.0%	8.5%
$k=50$	5.3%	14.2%	5.3%	4.9%	14.0%

	test t_6^k	test t_7^k	test t_8^k	test t_9^k	test t_{10}^k
$k=30$	4.6%	14.2%	8.5%	15.3%	4.9%
$k=35$	5.3%	15.3%	8.5%	8.5%	4.9%
$k=40$	14.1%	15.3%	5.3%	14.0%	15.3%
$k=45$	15.3%	4.9%	15.3%	4.5%	14.2%
$k=50$	5.3%	14.2%	5.3%	14.0%	15.3%

Figure 5.11: Relative coverage, as a percentage, of tests generated by TORX, with α_2

	test suite $\mathbf{T}^k$
$k=30$	63.1%
$k=35$	69.1%
$k=40$	72.8%
$k=45$	47.2%
$k=50$	54.2%

Figure 5.12: Relative coverage, as a percentage, of tests generated by TORX, with α_2

CHAPTER 6

Concluding remarks

In this thesis we study a formal approach to software testing. More specifically, we provide a mathematical foundation for conformance testing of implementations with respect to specifications formally described. Our developments and models are shaped by the following assumptions:

- We assume that the implementation is completely opaque, that is it is black box (i.e., we do not know its internal structure and can only observe its behaviour).
- We assume that the real implementation can be modelled by a formal model. This is known as the testing assumption.
- We assume the implementation to be input-enabled. (This is always the case except for Chapter 4, where we are able to relax slightly this assumption.)
- We assume that we are always testing reactive systems, which allow us to interact by applying inputs to them and observing their outputs.
- We assume that the interaction between the implementation and a test is fully synchronous (i.e., there is an unison coordination between inputs from the test and inputs on the implementation, and outputs from the test and outputs on the implementation).
- We assume that the given specification reflects precisely the intended behaviour of the system (i.e. specifications are considered correct).

We start our study in Chapter 1 by giving a formal interpretation to the various concepts used in model based conformance testing. Next, in Chapter 2, we present the testing formalism applied specifically to non-deterministic labelled input-output transition systems; in particular, we present the **ioco** testing theory. There, the notion of quiescence plays a central role. Quiescence characterizes systems that do not, and never will, produce an output without prior stimulation with an input. By treating quiescence as a special kind of system output, the notion of behavioural traces can be generalized to include quiescent observations. This allows to distinguish systems that are not distinguishable otherwise. A test derivation algorithm is presented and it is proved that the set of generated test is sound and exhaustive with respect to the **ioco** testing relation. These two chapters review existing formal approaches for conformance testing, and they provide the base-line to which these thesis' contributions build upon.

In Chapter 3 we present our first extension to **ioco**. The extension considers time explicitly, and this constitutes a crucial addition to the models considered in this thesis. The concepts defined in this chapter to deal with real-time propagate throughout the remainder of the chapter. In particular, the notion of quiescence becomes parameterized by the amount

of time that is need to recognize quiescent states on the implementation. As a consequence, our testing relation (**tioco**$_M$) is parameterized as well, with quiescence recognition as a parameter (M). This bound M represents the time it takes to infer that the implementation is in a quiescent state. We define a non-deterministic test generation framework, parameterized by the bound M, and show that the set of test generated is sound and exhaustive with respect to **tioco**$_M$. Moreover, we explore the relation between our proposed timed extension, **tioco**$_M$, and the non-timed approach, **ioco**.

Subsequently, in Chapter 4, we present our second extension consisting of both real-time and the addition of channels, which are partition on the inputs and output actions. The combination of channels and real-time are interestingly integrated together. A system may have a channel enabled or disabled depending on its particular execution stage, thus to model the fact that actions are enabled or disabled, it is sufficient to enable or disable the communication channels in which the actions occur. The resulting model is the timed labelled multi input-output transition systems (TLMTS), and it allows us to consider input enabledness and quiescence properties on a per channel basis, thus relaxing global system assumptions. We replace the input enabling requirement of a system by the following weaker requirement: for each input channel, either all inputs are allowed, or they are all blocked. Also, we replace the global bound M from the **tioco**$_M$ testing relation by a vector of bounds $\mathcal{M} = \langle M_1, \cdots, M_m \rangle$; where M_j represents the bound on the output channel j. Relaxing the global bound M for a vector of bounds means that we do not have to wait for the slowest response time to conclude the quiescence on a faster channel. The resulting testing relation is parameterized by a set of bounds which detect quiescence per each output channel, the **mtioco**$_{\mathcal{M}}$ conformance implementation relation. We develop a test derivation procedure for **mtioco**$_{\mathcal{M}}$, which is shown to be sound and exhaustive with respect to the **mtioco**$_{\mathcal{M}}$ implementation relation. Moreover, we elaborate on the relation between our proposed timed extension with channels, **mtioco**$_{\mathcal{M}}$, and the timed approach, **tioco**$_M$.

Finally, in Chapter 5, we present our semantic approach for test coverage. In this chapter, we provide a semantic point of view which allows us to study coverage formally and precisely. Our point of departure is a weighted fault model (WFM) that assigns a weight in the specification to each potential error in an implementation. We define our coverage measures relative to these WFMs. Since WFMs are augmented specifications, our coverage framework qualifies as black-box. Moreover, because WFMs are infinite semantic objects, we provide a finite representation as fault automata. Our theory allows to assign equal coverage measures to semantically equivalent specifications, taking into account that certain failures are more severe than others. Moreover, we provide algorithms that calculate and optimize test coverage. In particular, we compute the (total, absolute and relative) coverage of a test suite with respect to a WFM. Also, given a test length k, we present an algorithm that finds the test of length k with maximal coverage and an algorithm that finds the shortest test with coverage exceeding a given coverage bound.

Our extensions, based (mostly) on the **ioco** testing theory, are directed to answer the research questions proposed in Section 1.3, where we search for new theories that are able to test better, effectively and efficiently.

In this thesis we believe that we achieve our goal successfully. We support this conclusion by describing our contributions as follows:

- We present new models and formalisms that are usable for implementations and specifications taking into account real-time and real-time with channels.

- We develop new testing relations between our models.
- We compare and elaborate on the relation between the original relation we consider and the new ones proposed.
- We develop an algorithm to derive test for our timed testing relations.
- We prove that the proposed algorithms are sound and exhaustive with respect to their corresponding testing relation.
- We propose a new approach for test coverage in a semantic style.
- We prove that the proposed coverage approach is authentically semantic under preserving bisimilarities.
- We develop several algorithms to calculate test with optimal coverage.

More precisely, some of the main contributions for each chapter are shown in Figure 6.1.

	Contribution	
Chapter 3	timed output set	Definition 3.3.15
	$\mathsf{nttraces}^{\Delta}_{M}$	Definition 3.4.2
	$\mathbf{tioco}_{M}$	Definition 3.4.5
	M-quiescent implementations	Definition 3.5.1
	test generation procedure (TGP)	Section 3.6.1
	soundness of TGP	Theorem 3.7.4
	exhaustiveness of TGP	Theorem 3.7.6
	relation between **ioco** and $\mathbf{tioco}_{M}$	Theorem 3.8.2
Chapter 4	O_j-quiescent	Definition 4.3.3
	$\mathcal{M}$-quiescent implementations	Definition 4.3.6
	$\mathsf{nttraces}^{\Delta}_{\mathcal{M}}$	Definition 4.4.5
	channels output set	Definition 4.4.7
	$\mathbf{mtioco}_{\mathcal{M}}$	Definition 4.4.10
	multi test generation procedure (MTGP)	Section 4.5.1
	soundness of MTGP	Theorem 4.6.3
	exhaustiveness of MTGP	Theorem 4.6.5
	relation between $\mathbf{tioco}_{M}$ and $\mathbf{mtioco}_{\mathcal{M}}$	Theorem 4.7.3
Chapter 5	weighted fault models (WFM)	Definition 5.2.1
	coverage measures	Definition 5.2.2
	fault automaton	Definition 5.4.1
	finite depth WFM	Definition 5.5.1
	discounted WFM	Definition 5.6.5
	calibration	Theorem 5.7.2
	invariance under bisimilarities	Section 5.7.2
	absolute coverage algorithms	Section 5.8.1
	total coverage algorithms	Section 5.8.2
	relative coverage algorithm	Section 5.8.3
	optimization algorithms	Section 5.8.4

Figure 6.1: Principal contributions of this thesis per chapter

As possible future work, it would certainly be interesting to apply the **tioco**$_M$ testing relation to more realistic examples. We would also like to conclude the implementation tool of the **mtioco**$_{\mathcal{M}}$ testing relation, and apply the semantic coverage algorithms (that is being implementing in the SECO tool) to existing examples, thus allowing us to make more realistic judgments.

From theory to practice Our first proposed extension **tioco**$_M$, is already implemented for timed automata on the TORX tool. The second proposed extension **mtioco**$_{\mathcal{M}}$ is being implemented as part of current work. The third proposed approach for semantic coverage is also, as we mention above, currently being implemented in a new tool called SECO.

Moreover, most of the results presented in this thesis are published in refereed conferences (i.e. ATVA, ICFEM, FATES), so we are confident that these theories have been "tested" on the research community.

Formal methods, in general, are not yet smoothly integrated with practical, day-to-day development tools. We believe that theories should be easily applicable, in a way that practical communities (e.g., the industry) can appreciate the advantage that formal methods can offer. The work in this thesis attempts to achieve this effect. Although we focus on formal approaches, indeed we carefully develop them in a way that they can be built in practical and useful tools. By having this in mind, the tools which implement our theories can be directly applied to real life examples.

Bibliography

Author references

[1] L. Brandán Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In Jens Grabowski and Brian Nielsen, editors, *Formal Approaches to Software Testing, FATES*, pages 64–78, Linz, Austria, Sep 2004. Springer-Verlag GmbH. Also an Extended Version in Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands. Technical report, TR-CTIT-04-40, 20 pp., Oct. 2004.

[2] L. Brandán Briones and E. Brinksma. Testing multi input-output real-time systems. In *ICFEM 2005 Seventh International Conference on Formal Engineering Methods.*, pages 264–279, Manchester, UK, Nov 2005. Springer-Verlag GmbH. Also as Extended Version in Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands. Technical report, TR-CTIT-05-40, 20 pp., Sep. 2005.

[3] L. Brandán Briones, E. Brinksma, and M.I.A. Stoëlinga. A semantic framework for test coverage. In S. Graf and W. Zhang, editors, *Proceedings of the fourth international symposium on Automated Technology for Verification and Analysis (ATVA'06)*, LNCS 4218, pages 399–414. Springer, 2006. Also an Extended Version in Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands. Technical report, TR-CTIT-06-24, 31 pp., 2006.

[4] L. Brandán Briones and M. Röhl. Test derivation from timed automata. In M. Broy, B. Jonsson, and et al. J-P Katoen, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, pages 201–232. Springer-Verlag GmbH, Sep 2005.

Other references

[5] R. Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the 11th Internation Conference on Computer Aided Verification (CAV 1999)*, volume 1633 of *LNCS*, pages 8–22. Springer-Verlag, 1999.

[6] R. Alur and D. Dill. Automata for modeling real-time systems. In *In Proceedings, Seventeenth International Colloquium on Automata Languages and Programming*, volume 443, pages 322–335. Springer-Verlag, 1990.

[7] R. Alur and D. Dill. A theory of timed automata. In *Journal of Theoretical Computer Science*, volume 126(2), pages 183–235, 1994.

[8] T. Ball. A theory of predicate-complete test coverage and generation. In *Proceedings of FMCO*, pages 1–22, 2004.

[9] A. Belinfante, J. Feenstra, L. Heerink, and R.de Vries. Specification based formal testing: The easylink case study. In *Progress 2001-2nd Workshop on embedded Systems*, pages 73–82. STW Technology Foundation, Utrecht, 2001.

[10] A. Belinfante, J. Feenstra, R.de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In *International Workshop on Testing of Communicating Systems 12*, pages 179–196. Kluwer, 1999.

[11] G. Bernot. Testing against formal specifications: A theoretical view. In *TAPSAFT'91*, volume 2, pages 99–119. Springer-Verlag, 1991.

[12] H. Bohnenkamp and A. Belinfante. Timed testing with TORX. In *FM 2005: Formal Methods*, volume 3582/2005, pages 173–188, 2005.

[13] R. Boumen, I. de Jong, J. Vermunt, J. van de Mortel-Fronczak, and J. Rooda. Test sequencing in a complex manufacturing system. In *Xootic Magazine, 11(2)*, pages 9–16, 2005.

[14] E. Brinksma, R. Alderden, R. Langerak, J. Lagemaat, and J. Tretmans. A formal approach to conformance testing. In *Second Int. Workshop on Protocol Test Systems*, pages 349–363. North-Holland, 1990.

[15] Ed Brinksma. On the coverage of partial validations. In *AMAST '93: Proceedings of the Third International Conference on Methodology and Software Technology*, pages 245–252, London, UK, 1994. Springer-Verlag.

[16] R. Cardell-Oliver. Conformance test experiments for distributed real-time systems. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 159–163. ACM Press, 2002.

[17] P. Christian. Specification based testing with IDL and formal methods: A case study in test automation. In *Master's Thesis*. University of Twente, Enschede, The Netherlands, 2001.

[18] D. Clarke and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *IEEE Real Time Technology and Applications Symposium*, pages 112–124, 1997.

[19] M. Clatin. Manuel d'utilisation de TVEDA V3. In *User Manual LAA/EIA/EVP/109*. France Télécom CNET, Lannion, France, 1996.

[20] R. Cleaveland, I. Lee, P. Lewis, and S. Smolka. A theory of testing for soft real-time processes, 1996.

[21] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proceedings of Workshop on Verification and Control of Hybrid Systems III*, pages 208–219. Springer-Verlag, October 1995.

[22] M.van der Bijl and F. Peureux. I/O-automata based testing. In *Model-Based Testing of Reactive Systems: Advanced Lectures*, pages 173–200. Springer Berlin, 2005.

[23] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 197–212. Springer-Verlag NY, 1990.

[24] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed test cases generation based on state characterization technique. In *19th IEEE Real-Time Systems Symposium*, pages 220–229, 1998.

[25] ETSI. Es 201 873-6 v1.1.1 (2003-02). Methods for Testing and Specification (MTS). In *The Testing and Test Control Notation version 3: TTCN-3 Control Interface (TCI). ETSI Standard*, 2003.

[26] J-C. Fernandez, C. Jard, T. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *Computer Aided Verification, 8th International Conference, CAV '96*, volume 1102 of *LNCS*, pages 348–359. Springer, 1996.

[27] R.J.van Glabbeek. The linear time-branching time spectrum II (the semantics of sequential systems with silent moves). In *CONCUR'93*, volume 715, pages 66–81. E.Best, 1993.

[28] L. Heerink. Ins and outs in refusal testing. In *PhD thesis*, 1998.

[29] L. Heerink and J. Tretmans. Refusal testing for classes of transition systems with inputs and outputs. In *Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE X*, volume 107 of *IFIP Conference Proceedings*, pages 23–38. Chapman & Hall, 1997.

[30] M. Hennessy. Algebraic theory of processes. In *Foundations of Computing. Series*. MIT Press, 1988.

[31] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the 7th Symposium of Logics in Computer Science*, LICS, pages 394–406. IEEE Computer Scienty Press, 1992.

[32] T. Higashino, A. Nakata, K. Taniguchi, and R. Cavalli. Generating test cases for a timed I/O automaton model. In *IWTCS 1999*, pages 197–214, 1999.

[33] C. Hoare. In *Communicating Sequential Processes*. Prentice-Hall, 1985.

[34] G. Holzmann. In *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[35] ISO8807. *Information processing systems, Open Systems Interconnection, LOTOS, A formal description technique based on the temporal ordering of observational behaviour*. International Organization for Standardization, 1989.

[36] P. Kars. The application of PROMELA and SPIN in the BOS project. In *Proceedings Second SPIN Workshop*, volume 32, 1996.

[37] A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from SDL specifications. In *SDL '97 Time for Testing, MSC and Trends - 8th International SDL Forum*, pages 135–152. Elsevier, 1999.

[38] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *SPIN 2004*, pages 109–126. Springer-Verlag, 2004.

[39] K.G. Larsen, M. Mikučionis, and B. Nielsen. Real-time system testing on-the-fly. In *The 15th Nordic Workshop on Programming Theory (NWPT)*, Åbo Akademi University, Turku, Finland, 2003. Extended abstract.

[40] K.G. Larsen, M. Mikučionis, and B. Nielsen. Online testing of real-time system using UPPAAL. In *Formal Approaches to Software Testing*, Linz, Austria, 2004.

[41] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL in a nutshell. In *Journal on Software Tools for Technology Transfer*, 1997.

[42] 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.

[43] Z. Li, J. Wu, and X. Yin. Testing multi input/output transition system with all-observer. In *TestCom*, pages 95–111, 2004.

[44] Z. Li, X. Yin, and J. Wu. Distributed testing of multi input/output transition system. In *2nd International Conference on Software Engineering and Formal Methods (SEFM)*, pages 271–280. IEEE Computer Society, 2004.

[45] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real-time systems from logic specifications. *TOCS*, 13(4):365–398, 1995.

[46] R. Milner. In *A Calculus of Communicating Systems*, volume LNCS. Springer-Verlag, 1980.

[47] G. Myers. *The Art of Software Testing*. Wiley & Sons, 1979.

[48] G. Myers, C. Sandler, T. Badgett, and T. Thomas. *The Art of Software Testing*. Wiley & Sons, 2004.

[49] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. In *International Symposium on Software Testing and Analysis*, pages 55–64. ACM Press, 2004.

[50] R. De Nicola. Extensional equivalences for transition systems. In *Acta Informatica*, volume 24, pages 211–237.

[51] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. In *Theoretical Computer Science*, volume 34, pages 83–133, 1984.

[52] B. Nielsen and A. Skou. *Automated Test Generation from Timed Automata*. TACAS 2001, 2001.

[53] M. Phalippou. Relations d'implantation et hypothèses de test sur des automates à entrèes et sorties. In France L'Université de Bordeaux I, editor, *PhD thesis*, 1994.

[54] R.S. Pressman. Software engineering: A practitioner's approach. pages 1–22. McGraw Hill, 1992.

[55] H. Schaefer. Risk based testing, strategies for prioritizing tests against deadlines. Winter 2005 issue of Methods & Tools, 2005.

[56] J. Springintveld, F. Vaandrager, and P. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1–2):225–257, 2001.

[57] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[58] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. In *Software-Concepts and Tools, TACAS*, pages 127–146, 1996. Also: Technical Report N0. 96-26, Center for Telematics and Information Technology, University of Twente, The Netherlands.

[59] J. Tretmans. Testing techniques. In *Formal Methods & Tools Group. Faculty of Computer Science*. University of Twente. The Netherlands, 2002.

[60] J. Tretmans and E. Brinksma. TORX: Automated model-based testing. In *First European Conference on Model-Driven Software Engineering*. A.Hartmann and K.Dussa-Ziegler, 2003.

[61] H. Ural. Formal methods for test sequence generation. *Computer Communications Journal*, 15(5):311–325, 1992.

[62] R.J. van Glabbeek. The linear time-branching time spectrum I: The semantics of concrete, sequential processes. In *Handbook of Process Algebra*, pages 3–99, 2001.

[63] R.de Vries, A. Belinfante, and J. Feenstra. Automated testing in practice: The highway tolling system. In H. König, I. Schieferdecker, and A. Wolisz, editors, *TestCom*. Kluwer Academic, 2002.

[64] ITU-T SG 10/Q.8 ISO/IEC JTC1/SC21 WG7. *Information Retrieval, Transfer and Management for OSI; Famework:Formal Methods in Conformance Testing. Committee Draft CD 13245-1, ITU-T proposed recommendation Z.500.ISO-ITU-T*. International Organization for Standardization, 1996.

Nomenclature

LTS

A	label input-output transition system
Q	set of states
Q'	subset of states
L	set of labels
L'	subset of labels
L'_τ	$L' \cup \{\tau\}$
L'_δ	$L' \cup \{\delta\}$
$L'_{\delta\tau}$	$L' \cup \{\tau\} \cup \{\delta\}$
L^*	finite label sequences
L^+	$L^* \setminus \{\varepsilon\}$
L^ω	infinite label sequences
I	input labels
O	output labels
T	set of transitions

TLTS

$\mathcal{A}$	timed label input-output transition system
$\mathcal{Q}$	set of states
$\mathcal{Q}'$	subset of states
$\mathcal{L}$	set of timed labels
$\mathcal{L}'$	subset of timed labels
$\mathcal{L}'_\tau$	$\mathcal{L}' \cup \{\tau\}$
$\mathcal{L}'_\delta$	$\mathcal{L}' \cup \{\delta\}$
$\mathcal{L}'_{\delta\tau}$	$\mathcal{L}' \cup \{\tau\} \cup \{\delta\}$
$\mathcal{L}'_{\delta\gamma}$	$\mathcal{L}' \cup \{\delta_1, \cdots, \delta_n\} \cup \{\gamma_1, \cdots, \gamma_m\}$
$\mathcal{D}$	set of time-passage actions
$\mathcal{L}^*$	finite timed label sequences
$\mathcal{L}^\omega$	infinite timed label sequences
$\mathcal{T}$	set of timed transitions

TMLTS

$\mathcal{M} = \langle M_1, \cdots, M_m \rangle$	ordered set of bounds
$\mathcal{I} = \{I_1, \cdots, I_n\}$	input set partition
$\mathcal{O} = \{O_1, \cdots, O_m\}$	output set partition

Variables and constants

q	states
q^0	initial state
σ	trace (timed or untimed)
l	action (timed or untimed)
τ	internal action
β	action different than τ
a	input action or action and time
b	output action
δ	action that denotes quiescence
δ_j	action that denotes quiescence in channel j
γ_k	action that denotes blocking in channel k
π	path
ε	empty sequence
d, e	time variables
c	clock
C	set of clocks
κ	constant
u	variable over $[1, \cdots, n]$
v	variable over $[1, \cdots, n']$
j	variable for output channels over $[1, \cdots, m]$
k	variable for input channels over $[1, \cdots, n]$
z	extract constant
M, M_j	quiescence observation bounds

Tests

i, i_{IUT}	implementations
S	specification
t	tests
$\mathbf{T}$	test suite
k	test length
$TESTS$	set of tests for LTS(I, O)
$\mathcal{TEST}$	set of tests for TLTS(I, O)
$\mathcal{MTEST}$	set of tests for TLMTS$(\mathcal{I}, \mathcal{O})$

Coverage

r	error weighted function
$\overline{r}$	error weighted function per state
α	discounted function
T^I	set of input transitions
T^O	set of output transitions
$\mathcal{F}$	fault automaton
$f^k_{\mathcal{F}}$	finite depth weighted fault model
$f^\alpha_{\mathcal{F}}$	discounted weighted fault model

Summary

In the last years, increasingly complex systems are being put in charge of critical tasks. When these complex systems, are drive by sophisticated software, they need to attain a high degree of reliability. Unfortunately, developing correct systems is difficult, and in the past there have been several complex systems that went wrong because they lacked serious analysis of their potential behaviour. In this thesis, we study an effective way of obtaining confidence on the correctness of a system, known as testing. Testing is the systematic process of finding errors in a system by means of extensively experimenting with it.

In order to successfully test a system, it is crucially needed to count with both effective test cases and feasible strategies to execute them. Fortunately, work in formal methods helps us achieving this task in a precise and rigorous manner. A particularly successful formal theory of testing is the **ioco** theory, devised by Tretmans to work on labelled input-output transition systems. The theory smoothly covers issues like nondeterminism and quiescence (that is, the notion representing the absence of outputs). The **ioco** testing theory is clean and precise, and is the basis used in successful testing tools, like the TORX tool and the TGV tool. In this thesis we extend the **ioco** testing theory in three important directions, as follows.

Our first extension concerns the addition of real-time, which is crucial to the analysis of several systems (e.g., systems where actions are required to occur in a precise moment). New models and formalisms that take into account real-time are introduced. Furthermore, we develop a new testing relation between these real-time models, and a sound and exhaustive algorithm to derive tests for that relation.

Our second extension arises when we consider the input and output actions as being subdivided in communication channels. We explore how these channels interact with real-time. Interestingly, this new setting is more flexible since it allows us to relax some standard assumptions. We develop a testing relation between models with real-time and channels, and a sound and exhaustive algorithm to derive tests for this new richer setting.

Our third, final extension is concerned with the common problem that complete test suites usually cannot be covered in finite time for most interesting cases. Test coverage measures the proportion of the implementation exercised by a test suite. Existing coverage criteria are usually defined in terms of syntactic characteristics, having the disadvantage that behaviorally equivalent, although syntactically different systems have different measures. Moreover, these metrics do not take into account risks (i.e., values which represent that certain failures are more severe than others). We propose a novel approach for test coverage in a semantic style, where bisimilar processes measure equally. Moreover, we develop several algorithms to calculate tests with optimal coverage.

The results presented in this thesis enrich the formal theory of testing. They provide a solid basis for make the process of testing more applicable, complete, and effective, helping today's and tomorrow's complex systems to be more reliable.

Samenvatting

De laatste jaren worden steeds complexere systemen ingezet voor kritieke taken. Als deze complexe systemen bestuurd worden door geavanceerde software dienen ze aan een hoge graad van betrouwbaarheid te voldoen. Helaas is het ontwikkelen van correcte systemen moeilijk, en er zijn in het verleden dan ook complexe systemen geweest die de fout ingingen als gevolg van het ontbreken van een serieuze analyse van hun mogelijke gedrag. Dit proefschrift gaat over een effectieve manier om vertrouwen in de correctheid van een systeem te krijgen: testen. Testen is het systematisch zoeken naar fouten in een systeem door het uitgebreid aan experimenten bloot te stellen.

Om een systeem succesvol te kunnen testen is het van cruciaal belang om te kunnen beschikken over effectieve tests en over een werkbare strategie om die uit te voeren. Gelukkig is er werk gedaan op het gebied van formele methoden dat ons kan helpen om dit accuraat en zorgvuldig te doen. Een bijzonder succesvolle formele testtheorie is de **ioco** theorie, ontwikkeld door Tretmans, voor gelabelde transitiesystemen waarin onderscheid wordt gemaakt tussen invoer en uitvoer. Zaken als non-determinisme en quiescence (dat wil zeggen, de afwezigheid van uitvoer) zijn voor deze theorie geen probleem. De **ioco** testtheorie is ongekunsteld en precies, en vormt de basis voor sucessvolle test tools zoals TORX en TGV. In dit proefschrift breiden we de **ioco** theorie als volgt uit op drie belangrijke gebieden.

Onze eerste uibreiding bestaat uit het toevoegen van het begrip tijd, wat van cruciaal belang is voor de analyse van bijvoorbeeld systemen waarin activiteiten op een exact moment plaats moeten vinden. We introduceren nieuwe modellen en formalismen waarin het begrip tijd meegenomen kan worden. Verder ontwikkelen we een nieuwe formele testrelatie tussen deze modellen met tijd en een algoritme, dat correct en volledig is voor het afleiden van tests voor deze relatie.

Onze tweede uitbreiding komt tot stand wanneer we in- en uitvoeracties aan communicatiekanalen koppelen. We onderzoeken het effect van het koppelen aan kanalen in combinatie met het begrip tijd. Het blijkt dat deze combinatie het mogelijk maakt om een aantal standaard aannamen af te zwakken. We ontwikkelen een formele testrelatie tussen modellen met tijd en kanalen, en een algoritme, dat correct en volledig is voor het afleiden van tests voor deze uitgebreidere relatie.

Onze derde en laatste uitbreiding betreft het veelvoorkomende probleem dat volledige verzamelingen van tests over het algemeen niet volledig kunnen worden afgedekt in eindige tijd, in ieder geval voor de interessantere gevallen. De dekkingsgraad geeft aan hoeveel van het gedrag van het te testen systeem wordt geraakt door een verzameling tests. Bestaande dekkingsgraadcriteria zijn over het algemeen gedefinieerd aan de hand van de syntactische vorm van een systeembeschrijving. Dit heeft als nadeel heeft dat syntactisch verschillende beschrijvingen van het zelfde systeemgedrag verschillende dekkingsgraadwaarden (kunnen) hebben. Daarenboven kunnen in bestaande dekkingsgraadbenaderingen geen risico's uitgedrukt worden (waarmee we getalswaarden bedoelen die aangeven dat bepaalde fouten ernstiger zijn dan andere). We stellen een nieuwe aanpak voor die gebruikt kan worden om de dekkingsgraad op een semantische manier te definiëren, zodanig dat beschrijvingen die bisimulatie-gelijk aan elkaar zijn dezelfde dekkingsgraadwaarde krijgen. Tevens ontwikke-

len we een aantal algoritmes voor het berekenen van testverzamelingen met optimale dekkingsgraad.

De in dit proefschrift gepresenteerde resultaten vormen een verrijking van de formele theorie op het gebied van testen. Ze vormen een solide basis om het testproces beter toepasbaar, vollediger en effectiever te maken, waarmee ze ertoe bijdragen dat de systemen van vandaag en morgen betrouwbaarder worden.

Resumen

En los últimos años, cada vez más, sistemas complejos son puestos a cargo de tareas críticas. Cuando estos sistemas son controlados por un software sofisticado, necesitan un alto grado de confiabilidad. Desafortunadamente, desarrollar sistemas correctos es difícil, y en el pasado varios de estos sistemas han fallado debido a que su comportamiento potencial carecía de un análisis serio. En esta tesis, estudiamos una manera eficaz de obtener confianza en que un sistema sea correcto, conocida como testing. El testing consiste en un proceso sistemático desarrollado para encontrar errores en un sistema por medio de su experimentación extensiva.

Para testear con éxito un sistema, es crucialmente necesario contar con tests eficaces y estrategias factibles para ejecutarlos. Afortunadamente, el trabajo realizado en métodos formales nos ayuda a realizar estas tareas de manera exacta y rigurosa. Una de las mas famosas teorías de testing formal es la teoría de **ioco**, ideada en el trabajo de Tretmans sobre sistemas etiquetados de transición con entrada-salida (LTS). Dicha teoría cubre adecuadamente temas como el no-determinismo y la quietud (es decir, la noción que representa la ausencia de salidas). La teoría de testing **ioco** es rigurosa y exacta, y es la base usada por varias herramientas de testing, como TORX y TGV. En esta tesis extendemos dicha teoría en tres direcciones importantes, como presentamos a continuación.

Nuestra primera extensión se refiere a la adición de tiempo real, lo que es crucial para el análisis de varios sistemas (por ejemplo, sistemas donde las acciones deben ocurrir en un momento preciso). Introducimos nuevos modelos y formalismos que consideran tiempo real. Desarrollamos una nueva relación de testing entre dichos modelos con tiempo real, y un algoritmo que es consistente y exhaustivo para derivar tests confirmando dicha relación.

Nuestra segunda extensión considera las acciones de entrada y salida como subdivididas en canales de comunicaciones, donde exploramos cómo estos canales interactúan con tiempo real. Interesantemente, este nuevo contexto es más flexible debido a que permite relajar algunas suposiciones que eran estándar. También desarrollamos una relación de testing entre los modelos con canales y tiempo real, y un algoritmo consistente y exhaustivo que deriva tests para dicho nuevo contexto.

Nuestra tercera y última extensión se refiere al problema de que el testing, en general, no puede ser completado en tiempo finito, al menos para la mayoría de los casos más interesantes. La cobertura de un test, o un conjunto de tests, mide qué partición de una implementación es experimentada por dicho test, o dicho conjunto de tests. Los criterios de cubrimientos existentes, en general, son definidos en términos de características semánticas. Esto tiene la desventaja de que sistemas con comportamientos equivalentes, pero expresados sintácticamente en forma diferente, tienen medidas diferentes. Por otra parte, dichas métricas no tienen en cuenta riesgos, es decir, valores representando que ciertas fallas son más severas que otras. En esta tesis proponemos un nuevo método para medir el cubrimiento de un test, o de un conjunto de tests, de manera semántica. Mas aún, desarrollamos varios algoritmos para calcular tests con cubrimiento óptimo.

Los resultados presentados en esta tesis enriquecen la teoría formal del testing. Proporcionan una base sólida que hace el proceso de testing más aplicable, completo y efectivo, ayudando a que los sistemas complejos de hoy y mañana sean más confiables.

Titles in the IPA dissertation series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*.

Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*.

Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Fac-

ulty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05