THESIS ON INFORMATICS AND SYSTEM ENGINEERING C79

A Symbolic Approach to Model-based Online Testing

MARKO KÄÄRAMEES

TUT PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Science

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer Science on November 5, 2012

Supervisors: Prof. Jüri Vain, PhD
Chair of General Informatics, Dept. of Computer Science
Tallinn University of Technology, Tallinn, Estonia

Michael Reichhardt Hansen, PhD
Department of Informatics and Mathematical Modeling
Technical University of Denmark, Lyngby, Denmark

Opponents: Prof. Keijo Heljanko, D. Sc.
Department of Information and Computer Science
School of Science
Aalto University, Espoo, Finland

Margus Veanes, PhD
Research in Software Engineering (RiSE) Group
Microsoft Research, Redmond, USA

Defence of the thesis: November 28, 2012

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/Marko Kääramees/

European Union
European Social Fund

Investing in your future

Copyright: Marko Kääramees, 2012
ISSN 1406-4731
ISBN 978-9949-23-386-1 (publication)
ISBN 978-9949-23-387-8 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C79

Mudelipõhine *online*-testimine kasutades sümbolarvutust

MARKO KÄÄRAMEES

TTÜ
KIRJASTUS

ABSTRACT

Testing and test development is a significant part of the software development process. Model-Based Testing (MBT) provides a means for systematic and formal description of various aspects of an Implementation Under Test (IUT), enabling automated test generation and thus facilitation of test suite management and accommodation of changes to the IUT or the requirements. As it is infeasible to attempt to cover all possible behaviours and aspects of the IUT with tests, certain significant test purposes are selected by the test engineer.

Non-deterministic control structures and data components provide a powerful means of abstraction for high level modelling of complex systems, at the expense of making automated test generation more challenging. Non-determinism in the model does not allow pre-computation of a set of fixed test cases that are guaranteed to achieve the test purpose in the general case. Online model-based testing where test inputs are computed from the model and outputs fed back to the tester at the time of testing provides an approach where testing non-deterministic systems is possible. One of the restrictions to more widespread use of online model-based testing is the relatively high computational overhead at runtime.

In this thesis we develop a fully-fledged approach that addresses the computational overhead issue of online testing by pre-computation of test strategies based on the model and a formally specified test purpose. The proposed method allows the model of the IUT to be formalised as an Extended Finite State Machine over different first-order background theories. Both reachability and coverage oriented test purposes can be expressed using constraints attributed to edges of the model, called traps. We show how a testing strategy can be represented symbolically by a set of constraints and generated from the model and test purpose offline using symbolic backwards reachability analysis. The strategy can be used in online testing for efficient test input generation that guides the IUT towards fulfilment of the test purpose. The method is supported by the latest achievements of Satisfiability Module Theories (SMT) solver technology.

Finally we demonstrate the feasibility of the method on three case-studies. A case-study originating from an industrial software project performs reasonably in online test generation.

KOKKUVÕTE

Testimine ja testide väljatöötamine moodustab märkimisväärse osa tarkvaraarenduse protsessist. Mudelipõhine testimine võimaldab kirjeldada süstemaatiliselt ja formaalselt testitava süsteemi erinevaid aspekte, automatiseerida testide genereerimist ja hõlbustada testilugude haldamist ning kohandamist nõuete või testitava süsteemi muutumisel. Kuna süsteemi kõikvõimalike käitumiste testimine ei ole praktikas enamasti võimalik, siis valib testiinsener mõned olulisemad testi eesmärgid, mida püütakse testimise käigus saavutada.

Mittedeterministlike andmekomponentidega mudeleid kasutades saab keerulisi süsteeme modelleerida abstraktsel ja üldisel tasemel, kuid reeglina vaid testide genereerimise keerukuse hinnaga. Mittedeterminismi esinemine mudelis ei võimalda üldjuhul leida fikseeritud hulka testi stsenaariume, mis garanteerivad testi eesmärgi täitmise. Mudelipõhine *online*-testimine, kus testi sisendid leitakse mudeli ja testimise ajal süsteemilt testrile saadetava väljundi alusel, võimaldab ka mittedeterministlike süsteemide testimist. Üheks oluliseks takistuseks *online*-testimise vahendite laiemalt kasutusele võtmisel on senini olnud nende suur arvutusjõudluse vajadus testimise ajal.

Käesolevas väitekirjas arendame välja meetodi *online*-testimiseks, kus suure arvutusmahu probleemi lahendamiseks leiame enne testimist mudeli ja testi eesmärgi alusel testimisstrateegia. Meetod võimaldab modelleerida testitavaid süsteeme laiendatud lõpliku olekuautomaadiga üle erinevate esimest järku taustateooriate. Mudeli üleminekutele lisatavate kitsendustega on võimalik formaliseerida nii katvuse kui saavutatavuse tüüpi testi eesmärke. Näitame, kuidas on võimalik esitada testimisstrateegiat kitsenduste süsteemina ja seda genereerida sümbolarvutusel põhineva tagurpidi saavutatavusanalüüsiga. Selline strateegia on kasutatav efektiivseks testide sisendi genereerimiseks *online*-testimise ajal, nii et süsteemi juhitakse testi eesmärkide täitmise suunas. Väljapakutud testide genereerimise meetod tugineb *Satisfiability Module Theories* (SMT) lahendajate teooria uutele tulemustele ja tehnilistele realisatsioonidele.

Väljapakutud meetodi kasutatavust demonstreerime kolme eksperimendi näitel. Tööstuspartnerilt pärineva eksperimendi tulemused näitavad piisavat jõudlust testide *online*-genereerimisel.

ACKNOWLEDGEMENTS

I am grateful to everybody who has encouraged, advised, supported, and helped me in the process of completing this thesis. This would not have been possible without you. You have helped me to learn a lot during the process about myself, all of you, and how things are set in the world.

I would like to sincerely thank my supervisor Prof. Jüri Vain for providing the inspiring topic, insightful comments, and being supportive in every possible way during all the years we have worked together.

The core of this thesis was written while visiting the Embedded Systems Engineering section at Technical University of Denmark in 2011. I am particularly thankful to my second supervisor Michael R. Hansen for his guidance in formulating the ideas and explaining these in an understandable way. His humble but apposite questions have caused me on several occasions to put aside the work on which I have spent several days and dig into important problems for weeks. His support at the right moments has been crucial in completing the thesis. I am also thankful to Aske Brekling for a good company and help in conducting experiments. It was a privilege to be a part of the inspiring research community at DTU.

I am thankful to my colleagues at Tallinn University of Technology and in the projects where I have participated. The discussions with Juhan Ernits, Kullo Raiend, Andres Kull, Danel Ahman, and Maili Markvardt have been very important in developing the ideas presented in this thesis. I am especially thankful to James Chapman, Prof. Tarmo Uustalu, and Prof. Kuldar Taveter for their comments and support.

I am grateful to my opponents, Prof. Keijo Heljanko and Margus Veanes for the time and hard work they invested in reading my thesis, their insightful comments and constructive criticism.

I would like to acknowledge the organisations and projects that have supported my research and the development of the ideas presented here: Tallinn University of Technology, ELIKO Competence Centre, the project “Synthesis of model based reactive planners for nondeterministic and distributed systems” supported by Estonian Science Foundation, the project “Formal Methods in Distributed systems” funded by the Estonian Ministry of Education and Research, and the ITEA project "Deployment of Model-Based Technologies to Industrial Testing" (D-MINT) supported by Enterprise Estonia.

There have been many friends standing by my side and taking over some of my duties. Thank you Edda, Lea, Kalle, Taavi, Johannes, Vilver, Erki, Valjo, and Brigitta. Finally, I am most thankful to my dear wife Karmen for endless support and beloved kids Arpad, Marten, and Adeele for understanding and for the time they could not but deserved to spend with me. Above all, I am thankful to God, the ultimate source of everything.

CONTENTS

ACRONYMS

API	Application Programming Interface
EFSM	Extended Finite State Machine
ERPT	Extended Reactive Planning Tester
FSM	Finite State Machine
GLAS	Guarded Labelled Assignment System
IOCO	Input Output Conformance
IOTS	Input Output Transition System
IUT	Implementation Under Test
I/O-EFSM	Input/Output Extended Finite State Machine
LTS	Labelled Transition System
MBT	Model-Based Testing
MDE	Model-Driven Engineering
RPT	Reactive Planning Tester
SAT	Boolean Satisfiability Problem
SMT	Satisfiability Modulo Theories
STS	Symbolic Transition System

LIST OF SYMBOLS

Symbol	Description	Page
$\mathcal{A}$	arity function	37
α	assignment function	39,44
$\mathcal{C}$	set of models fo first order language	40
$\mathcal{D}$	domain of a model	39
$\mathcal{E}_{l \to S}$	promising outgoing edges	74,87
ϵ	missing (refused) input or output	42
$\mathcal{F}, f$	set of functions, a function	37
$\mathcal{I}$	interpretation function	39
$\mathcal{M}$	model of first order language	39
$\mathcal{P}, p$	set of predicates, a predicate	37
Σ	signature of first order language	37
$\mathcal{T}_\Sigma$	theory over signature Σ	40
τ, t	set of terms, a term	38
Φ, ϕ, φ	set of formulas and a formula	38
$\wedge, \vee, \to, \neg$	propositional connectives	37
$\exists, \forall$	quantifiers	37
$[\![]\!]^{M,\alpha}$	valuation function	39
$\models$	satisfiability relation	39
A	set of interactions	44
C	constraint	63
$C^{+}_{l \to S}$	weakest reachability constraint	67
$C^{0}_{l \to S}$	shortest run reachability constraint	67
$C^{g}_{e \to S}$	guarding constraint	67
D	type (domain) predicate	41
E	set of edges of automaton	41
g	a guard of an edge	41

Symbol	Description	Page
I, i	set of input labels, an input label	41,44
L, l	set of locations of automaton, a location	41
M	automaton	41,59
O, o	set of output labels, an output label	41,44
Q	queue	74
$\mathcal{S}, s$	set of states, a state	44
S,(l, C)	symbolic state	63
T	set of transitions	44
Tr, tr, (e, C)	set of traps, a trap	58
U	a list of updates of an edge	41
v	actual parameter (value)	44
X, x, y	a set of variables, a variable	37, 41
X_a, X_i, X_o, X_s	sets of auxiliary, input, output, state variables	41
$iLabel$	input label	66
$Pre()$	pre-image	64
$wp()$	weakest pre-condition	65

Part I

THESIS

1 INTRODUCTION

Software is increasingly pervasive and the extent to which we depend on software in our everyday lives continues to grow. A modern car includes more than 100 microprocessors for controlling all kind of functions from fuel injection to playback of MP3s. It is difficult to find a new washing machine or refrigerator without an embedded computer-based controller. Many of us are carrying devices with us that have computing power comparable to that a supercomputer had 30 years ago. Supermarkets, banks and gas stations are not able to operate when their IT systems fail. The IT systems we are surrounded by can make some aspects of the life much more efficient (e.g. submitting a pre-filled tax return online), cheaper (e.g. free international video calls) or safe (e.g. traction control in cars), but it also means that finding faults in software becomes more important. Faults in software can be mere irritations such as temporarily being unable to access a service, but they can also cause material damage or even endanger lives. Space agencies have lost their missions due to software bugs (Ariane 5, Mars Climate Orbiter). A radiation therapy device (Therac-25) has given a lethal dose of radiation to many patients. A soviet duty officer avoided nuclear war by acting against protocol when the faulty alert system gave a false alert of approaching missiles in 1983. Although some aspects and components of software can be formally and rigorously verified, industry relies largely on testing to assure the reliability of the hardware and software. It is not uncommon that half of the software development budget is spent on testing.

1.1 SOFTWARE TESTING

An apposite definition of testing is given in [1]:

> Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems.

> Software testing consists of the *dynamic* verification of the behaviour of a program on a *finite* set of test cases, suitably *selected* from the usually infinite executions domain, against the *expected* behaviour.

Four keywords are emphasized in the definition above:

- *Dynamic*: The real execution of the implementation is involved. Dynamic verification is in contrast to and is complemented by static methods like code inspection and static analysis. The outcome of the execution may not be known beforehand, because of non-determinism being involved.
- *Finite*: Only a finite number of finite tests can be executed. Testing is not exhaustive in contrast to formal verification. The finite set of test cases cannot guarantee that there are no flaws in the system under test.
- *Selected*: The finite set of tests should be selected from the potentially infinite set according to some criteria, purpose or requirements to support the goal of the testing in the best way.
- *Expected*: The tester must be able to conclude whether the observed output is expected or not. This is a question of having a test *oracle*.

White-box and black-box testing

There is no universal approach or method for testing. Different methods of testing are applied at different phases of development and have different objectives. The testing methods and uses are typically divided in to white-box and black-box testing methods. *White-box testing* means that the internals (source code) of the system is known and the tests are developed using that knowledge. The white-box approach can be applied on different levels known as unit testing, integration testing, and system testing. These levels are for testing modules separately, testing modules together, and testing the whole system respectively. In the case of *black-box testing* no knowledge of the implementation or the internals of the system is assumed. The functionality of the system as seen at the interface is tested for conformance to requirements, specification or reference implementation. We focussing on black-box testing in this thesis.

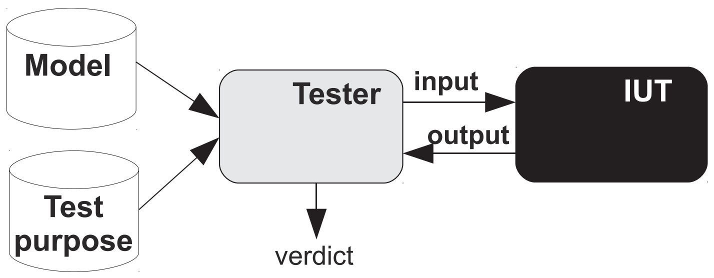


Figure 1.1: Model-based testing

Test objective

Testing may have different objectives like regression testing, performance testing, stress testing, security testing, functional testing and others [1]. Regression testing is used for checking that the behaviour and functionality of the unchanged aspects of the system did not change during the modifications. Performance testing is used to check if the system meets the performance requirements. Stress testing is used to examine the behaviour of the system when the maximum intended load is exceeded. These objectives are orthogonal to functional or conformance testing, which is used to assure that the system does what is expected by the specification. We are focussing on conformance testing in this thesis.

1.2 MODEL-BASED TESTING

Model-Based Testing (MBT) is a method where the tests and test oracle are derived from a formal model of the Implementation Under Test (IUT) and test purpose. The main application field of MBT is black-box functional testing [62], but it has been used in white-box [40], regression [14] and other types of testing. In model-based testing one can automate the test or tester generation that usually needs a lot of manual work. A good example of the benefits of model-based testing can be found in [30]. The general view of model-based testing is illustrated in Figure 1.1.

Model of the IUT

A model of the IUT is a formal finite representation of the possibly infinite set of acceptable input-output behaviours of the IUT. The model may originate from the specification, abstraction or other sources and may have different internal structure than the IUT. There are a wide variety of formal specification formalisms. A good overview of the classes of formalisms is provided in [47]. We concentrate more on automata and transition based modelling formalisms. Finite State Machine (FSM) are widely used for modelling the control flow of the systems that communicate to the environment. An FSM consists of locations (states) and edges (transitions) that have an associated input and output symbols. In order to model a system which has both control and data parts (e.g., communication protocols), an extension is needed. Such systems are represented using an Extended Finite State Machine (EFSM) model. An EFSM has internal state variables in addition to locations and the transitions depend and change the value of the variables. EFSMs provide a means of representing very large or infinite state spaces and sets of possible behaviours in a compact form.

An EFSM is *deterministic* if the state and input determine the next transition and state unambiguously and it is *non-deterministic* otherwise. Non-determinism of the model of the IUT can arise from several sources. It may be that the IUT has some non-deterministic aspects due to, for example, a multi-threaded or distributed implementation. The model can also represent the requirements given in the specifications that allow freedom of different implementations and such freedom is expressed as non-determinism. Non-determinism may also emerge from abstraction of some aspects of the system. For the model builder non-determinism can be a useful tool but it places much greater demands on the test generation tool. Many of the available model-based testing tools assume that the model is deterministic. The main problem lies in the fact that it is not possible to compute a fixed set of test cases that achieve the test purpose based on the non-deterministic model. It is necessary to decide the inputs and parameters given to the IUT during testing.

Test purpose and goals

The model specifies a very large or infinite set of possible behaviours. It is not feasible to run tests on the real IUT for all possible behaviours. It means that a *test purpose* is needed in addition to the model to select which kinds of behaviours should be tested.

The test purpose may originate from the requirements in the informal specification. The test purpose may also originate from the general approach of finding as many faults as possible and be expressed as some kind of coverage criteria [27]. The coverage criteria are usually given in MBT using the elements of the model, e.g. transition, state, or border value coverage.

Fulfilling the purpose of test may involve many tests or subtasks. We call such subtasks *test goals*. A test purpose may consist of many test goals. For instance, a test goal may correspond to a test case and a test purpose to a test suite in the testing of deterministic systems. Test goals are formally specified in MBT as test scenarios, states, or transitions to be reached in the model or elements of the model that have to be covered.

Test verdict

The testing process needs some way to evaluate if the observed behaviour of the IUT is correct. This is called the *test oracle* problem. Some predefined criteria or even human expertise is used in the manual testing for that purpose. In the case of MBT, the test oracle can be derived from the model. It is formalised as conformance relation between the model and the IUT. A test fails whenever a non-conforming behaviour is observed. The formal definition and discussion about conformance is given in Section 4.2.

Tester

In the context of MBT we mean under the term tester some testing engine that forms an artificial environment for the IUT. It uses a test suite in some representation (e.g., tests, test script, or test strategy) and is able to supply inputs to the IUT, observe outputs, evaluate the conformance of the behaviour, give verdicts, and fulfil the test purpose in this way. One common approach is to generate a test suite of test scripts. Every script is a sequence of inputs and expected outputs with the ability to give a verdict depending on the actual run of the test. This is called *offline* test generation and is suitable for deterministic systems. Such scripts can be generated from deterministic models.

The things are different when the model is non-deterministic. It even does not matter if the IUT is actually deterministic and the non-determinism in the model is caused by abstraction or loose specification. It is neither possible nor feasible to generate such linear test scripts in this case, because the reaction of the IUT is not known beforehand and the tester's behaviour for every reaction should be included in the script. This may cause infeasible or infinite branching. It may be possible to express the behaviour of the tester as an automaton. This approach of tester generation has close relations to controller synthesis. Another possibility is a game-theoretic approach, where the testing is viewed as a game between the tester and the IUT. The tester needs a strategy to achieve its goal and tester generation boils down to strategy generation. In any case, the linear test cases cannot be pre-computed from a non-deterministic model and the actual inputs sent to the IUT should be generated on-the-fly depending on the reactions of the IUT. This is also known as *online testing*. In the following we treat *online testing* and *on-the-fly testing* to be equivalent terms.

1.3 OBJECTIVE OF THE THESIS

The main objective of the thesis is to develop a model based testing approach for efficient online testing based on the (possibly non-deterministic) model of the IUT and a test purpose. More specifically:

- A formalism suitable for modelling the IUT that supports test generation.
- A method to specify test purposes that are rich enough to represent both scenarios and structural coverage criteria.
- A method for offline analysis of the model to generate a testing strategy that can be used for efficient online testing.
- Empirical evaluation of the feasibility of the method in case-studies.

1.4 RELATED WORK

Along with spreading the ideas of Model-Driven Engineering (MDE) the MBT as constituent of MDE has become one of the dominant approaches to constructive test design. Since the focus of thesis is model-based online testing of non-deterministic systems we do not endeavour to comprehensive overview of MBT in general, but refer to surveys [63, 3, 9] and books [62, 11, 73] written on the subject. We focus on earlier results specifically related to model-based test generation for non-deterministic FSM/EFSM and related models.

The EFSM model has been widely studied and many methods for test data generation are proposed [46, 51, 56, 43]. The complication with EFSM models is that paths can be infeasible due to the variable interdependencies among the actions and conditions. There is no input data that can cause an infeasible path to be traversed. While the feasibility of paths is undecidable in general, there are techniques that handle them in certain special cases [28, 23, 31, 13]. Also the methods of heuristic search are used for discovering feasible paths [43, 20, 72]. Many important theoretical results of MBT are based on EFSM related modelling formalisms and semantic frameworks like Labelled Transition System (LTS) [59], Symbolic Transition System (STS) [26] and Guarded Labelled Assignment System (GLAS) [67]. For instance, conformance between the model and the IUT is defined as a Input Output Conformance (IOCO) relation and other related relations in [59] lifted to symbolic setting of STS in [26] and shown to be closely related to alternating simulation relation in [67]. We base our notation of conformance on alternating simulation relation.

An important aspect to be dealt with test generation for non-deterministic systems is the proportion between online and offline generation activities. Due to the non-deterministic nature of the model of the IUT test sequences cannot be synthesized fully in advance (offline). The test generation procedure can derive the test inputs including the values of data parameters only one by one online depending on the current state of the IUT and the target state set by the test goal. Thus, in online testing it is not required to explore the whole state space of the model of the IUT any time the test stimulus is generated. Instead, the decisions about the next actions are made by observing the current output of the IUT locally [60]. However, online test execution requires more run-time resources for interpreting the model and choosing the most relevant test stimulus. The taxonomy of online testing methods is based on how the test purpose is defined, how the test stimuli are selected on-the-fly, and what is the planning effort behind each choice.

The test purpose can be stated in a very abstract way when applying the IOCO [59, 10] relation. Usually the conformance relation is tested using either a completely random or heuristic driven state space exploration algorithm. A test stimulus at a given state is selected randomly from the set of stimuli having uniform distribution of preference to trigger a next transitions of the IUT model. Random choice has been used in the early TorX tool [7], Uppaal-Tron [35, 48], and also in the on-the-fly testing mode of SpecExplorer [54, 70]. In [24] the transition probabilities directed input selection method is introduced to TorX. More restrictive are the test goal-directed exploration algorithms that reduce the total number of states to be explored in a model. The goal-directed approach is stronger than random exploration in the sense of providing guidance towards a certain set of the IUT execution sequences that cover so called test goal items (e.g., states or transitions in the IUT model). The goal-directed approach was introduced in [25, 44], used in testing tool elaborated in [18] and later used in TorX [61] and TGV [41]. Uppaal Cover [35] specifies observer automata for specifying test goals.

Further advancement of test goal specification has been introduced in NModel [69, 68] where the IUT model presented as a model program can be composed with test scenario models to restrict the sets of test sequences. An "anti-ant" [52] based algorithm of reinforcement learning [70] is used to cover specified test sequences in the model program, dynamic approach of DART system [29], inserted assertions [44], path fitness [20], etc. are applied for guiding the exploration of the IUT state space. Game-theoretic approach is used for generating winning strategies for achieving the test goal in non-deterministic models [54] and implemented in AsmL tester tool. The generated strategies do not take the infeasibility of the paths and data-communication into account. An efficient method for generating a winning strategy for achieving a goal in timed automaton is presented in [12]. Similar method could be used for test strategy generation for a single reachability goal, but not for coverage based testing purposes. The extreme of guiding the selection of test stimuli is exhaustive planning by solving at each test execution step a full constraint system set by the test purpose and test planning strategy. For instance, the witness trace generated by model checking provides a selection of the next test stimulus. The critical issue in the case of explicit state model checking algorithms is the size and complexity of the model leading to the explosion of the state space, specially in cases such as "combination lock" or deep nested loops in the model [31]. Therefore, model checking based approaches are used mostly in offline test generation.

The Reactive Planning Tester (RPT) synthesis method of [66] is online testing approach that combines structural test coverage criteria with reactive online planning strategy. The test purpose is defined using simple boolean trap variable assignments associated the IUT model transitions. The reactive planning strategy is synthesised and partially evaluated offline. At each step of concrete test run the state vector is assumed to be fully observable that allows evaluation of strategy constraints and calculate the explicit value of gain function that determines locally optimal test inputs. The main limitation of RPT approach is that it is applicable for output-observable non-deterministic EFSM models only with the assumption that all transition paths are feasible and variables have finite domains.

Novelty of the contribution

The main contribution of this thesis is a MBT approach for efficient online testing based on a test purpose and an EFSM model with limited non-determinism. Novelty of the contribution lies in the following:

- Model-based test generation for EFSM (Section 3.2) models with infeasible paths and non-determinism, that is constrained by the condition of output-observability that is a key prerequisite for efficient model-based online testing
- Test purpose specification using trap predicates that are a generalization of boolean trap variables introduced in [31] (Section 4.3). Trap predicates and auxiliary variables enable to specify both scenario and structural coverage type of test purposes, both on control structures and data components.
- Test generation is divided to offline testing strategy generation and online explicit test data generation phases as in [66, 65], but the novelty proposed in the thesis is that the strategy generation is carried out and represented completely symbolically (Chapter 5) avoiding the need of constraining the data components to finite domains and converting the model to FSM representation.
- The proposed online procedure for test data generation is based on the symbolic test strategy that avoids expensive analysis of the model during the test run and achieves model coverage or feasible path to the test goals with close to optimal length (Chapter 6). The test generation is optimized to covering all goals in case the test purpose consists of a set of goals, e.g. in the case of achieving some coverage.
- Demonstration of the feasibility of symbolic reachability analysis using algebraic simplifications and quantifier elimination procedures that are supported by contemporary Satisfiability Modulo Theories (SMT) solvers (Subsection 5.4.3, Section 7.2).

1.5 OVERVIEW OF THE THESIS

The rest of the thesis is organized as follows:

CHAPTER 2: We give a quick overview of the MBT approach proposed in the thesis by an example of a vending machine.

CHAPTER 3: We define a formal framework for modelling the IUT, the syntax and semantics of the Input/Output Extended Finite State Machine and the first order logic it is based on. In addition, we explore the similarities and differences of Input/Output Extended Finite State Machine (I/O-EFSM) to related formalisms.

CHAPTER 4: We elaborate on testing, conformance, test purpose, goals and coverage. Traps are defined for formal specification of the test goals. A simple method is presented that can be used for generating tests from deterministic models using traps.

CHAPTER 5: We show how the testing strategy can be represented by constraints and distance measures. The symbolic representation of the sets of states is defined and reachability analysis explained.

CHAPTER 6: The testing procedure using the symbolic test strategy is explained in details.

CHAPTER 7: We describe the overall workflow and tools used in the implementation of the method.

CHAPTER 8: The feasibility of the method is demonstrated on three case-studies from different backgrounds. One of the case studies is an academic example to demonstrate the properties of the proposed method. The second case study is a communication protocol used also in other approaches of MBT and the third is a billing application from an industrial project.

CHAPTER 9: The chapter summarises the results of the work presented in this thesis and discusses directions for future research.

2

OVERVIEW OF THE METHOD

In this chapter we give an overview of the method proposed in this thesis. The sections of the overview correspond roughly to the following chapters. We illustrate the method on the simple example of a latte vending machine without going to the details presented in the following chapters.

We start from an informal set of specifications and develop a model and corresponding test goals. Then we show the generated result of the model exploration and how the data is used during online testing.

The test generation is divided into two main phases – computationally expensive model analysis yielding a test strategy and computationally efficient test input generation based on the strategy. The division is motivated mainly by the need to be able to test the systems with a non-deterministic model of the IUT. Test inputs cannot be pre-computed for non-deterministic systems and must be generated on-the-fly during testing. They depend on the actual state of the system. The test strategy generated during the model analysis enables to find a suitable input based on local data, deeper model exploration is not needed. The method is motivated by, but not limited to, the non-deterministic systems and can be applied for the deterministic systems and models as well.

2.1 IUT AND ITS MODEL

The tests are generated from the model of the IUT and the set of test goals. The model gives a black-box view of the IUT at the testing interface. The goals of the test are specified as a set of reachability tasks on the model. The formal definition of modelling and test goal specification formalism can be found in Chapter 4. We illustrate the method on a simple example of testing a controller of a latte vending machine.

The following specification and model are simplified and do not reflect many important aspects of a realistic vending machine. It can also be viewed as an abstraction of a vending machine with only some interesting

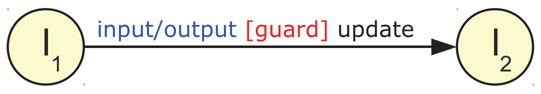


Figure 2.1: Extended Finite State Machine

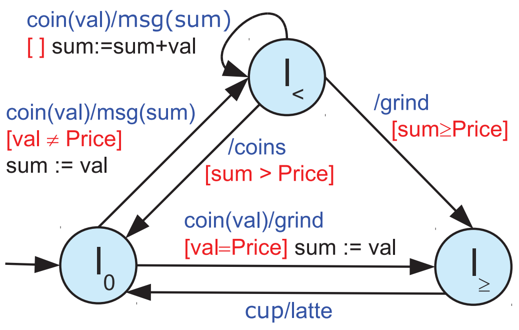


Figure 2.2: Model of latte vending machine

aspects modelled. The informal specification consists of some requirements:

1. The vending machine should be able to collect different coins.
2. It should deliver latte when the amount of money equal to the predefined price is given. The controller should start grinding after receiving the money and brewing after that.
3. Brewing should not start if there is no cup present.
4. If an amount of money exceeding the price is given, then it can either return all the coins or give latte. The choice is left to the implementation.
5. No change is given if latte is brewed for an amount of money exceeding the price to simplify the logical and mechanical design of the vending machine.

The model is given as an EFSM. The elements of EFSM are presented in Figure 2.1 and defined in Section 3.2. Here we explain the modelling formalism based on the specification example of the vending machine depicted in Figure 2.2. The model consists of state variables (*e.g.*, sum for the amount of money inserted), locations attributed by labels ($l_0, l_<, l_\geq$), and directed edges connecting the locations. The edges are labelled with:

input to be received to make a transition. It may be without parameters (*e.g.*, cup) or with parameter (*e.g.*, coin(val)), where the parameter val represents the value of the coin *guard* condition (*e.g.*, $\mathsf{sum} > \mathsf{Price}$) that must evaluate to true for being able to take a transition over the edge. The constant Price denotes some predefined value here.

update (*e.g.*, $\mathsf{sum} := \mathsf{sum} + \mathsf{val}$) that defines how the values of the state variables change because of the transition

output (*e.g.*, grind,msg(sum)) sent by the transition.

The behaviour of the automaton can be understood so that it starts from location l_0 and waits for a coin. The next transition is deterministic and depends on the value of the coin. If the value of the coin is equal to the price grinding starts. Then from the location $\mathsf{l}_{\geq}$ latte is served if the cup is present. If the value of the coin received in the initial location does not equal to the price, the automaton moves to the location $\mathsf{l}_{<}$ where more coins can be collected until the sum is greater or equal to the price. There is a non-deterministic reaction possible in case $sum > Price$. The automaton can make a transition to l_0 or $\mathsf{l}_{\geq}$. Although it is not possible to control the choice, the outcome determines which choice was taken. This is called *output observability* and we assume the models have such property.

There are some more concrete restrictions on the model. For the vending machine model we assume that all the variables are of type *integer*, *sum* is non-negative, *Price* is 20, and only coins with values 1, 5 and 20 are available.

2.2 TEST REQUIREMENTS

Our aim is to generate tests that cover the whole model in some sense or at least visit some crucial elements of the model. To achieve this we model the goals of the test by traps. A trap is a pair of a transition and a predicate on state variables and input parameters. The test goal modelled by a trap is satisfied when the transition is taken with the associated predicate satisfied as a pre-condition. The discussion about the specification of the goal of the testing is in Chapter 4, traps are formally defined in Section 4.3.

We have three test goals in our example expressed by three traps:

1. Test that the machine is able to brew latte
2. Test that it can take several smaller coins before brewing latte
3. Test if it is possible to get latte if coins with total value more than the price (20) are given

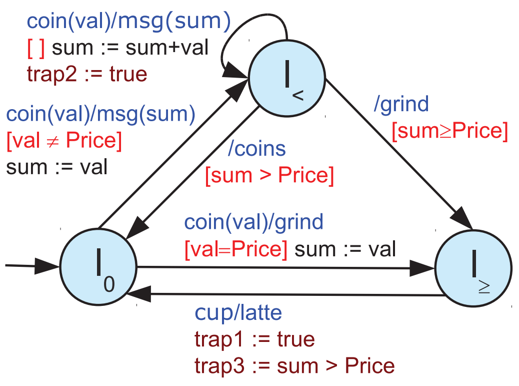


Figure 2.3: Model of vending machine with traps

The traps can be attributed to the edges as assignments of the trap predicates to special Boolean valued *trap* variables. Finding a test can also be seen as finding a run on the automaton that evaluates the trap variable to *true*. The model augmented with traps is depicted in Figure 2.3.

2.3 OFFLINE MODEL ANALYSIS

Our aim is to analyse the model augmented with traps and generate a test strategy that can guide the testing process towards the uncovered traps. The generated strategy consists of constraints and distance measures attributed to the edges and locations. The components of the strategy and the method to find these is provided in Chapter 5. The strategy can be viewed as signs that help us to choose a right direction with appropriate input parameters. The simplified strategy for reaching trap2 and trap3 is depicted in Figure 2.4

The constraints are generated by backwards breath-first symbolic analysis of the model starting from the condition associated with the trap. The process ends when a fixpoint or the allowed depth of the exploration is reached. The constraints are simplified during the process to keep the size of the constraints under control. Simplification is the most computationally demanding and critical part of the work, taking more than 95% of the resources usually. The detailed description of the model exploration procedure is given in Chapter 5.

2.4 ONLINE TESTING OF NON-DETERMINISTIC SYSTEMS

The testing process simulates the model in parallel to driving the IUT. It checks if the behaviour of the IUT conforms to the model during the process and drives the process using the strategy so that all the goals

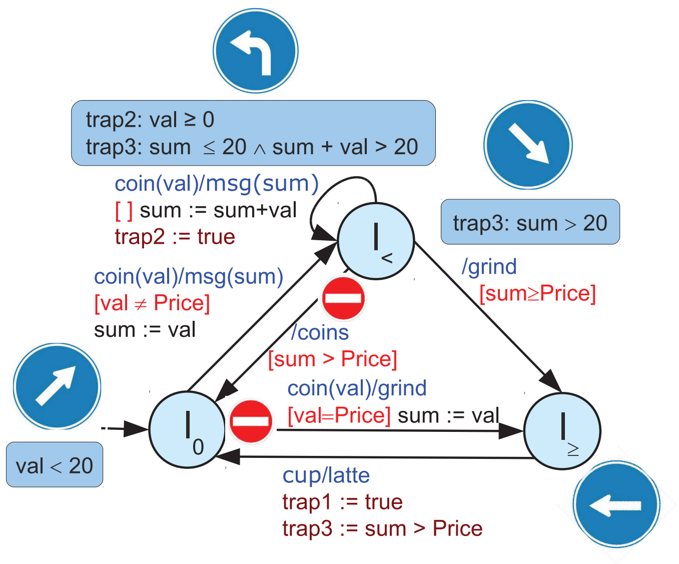


Figure 2.4: Test strategy for reaching *trap3*

Table 2.1: online input generation for vending machine example

Step	Goal	Input	Output	State	Covered
1	trap2	coin(5)	msg(5)	$(l_<, \text{sum} = 5)$	
2	trap2	coin(1)	msg(6)	$(l_<, \text{sum} = 6)$	trap2
3	trap3	coin(20)	msg(26)	$(l_<, \text{sum} = 26)$	
4	trap3		coins	$(l_0, \text{sum} = 26)$	
5	trap3	coin(1)	msg(1)	$(l_<, \text{sum} = 1)$	
6	trap3	coin(20)	msg(21)	$(l_<, \text{sum} = 21)$	
7	trap3		grind	$(l_\geq, \text{sum} = 21)$	
8	trap3	cup	latte	$(l_0, \text{sum} = 21)$	trap1, trap3

(traps) are covered one after another. The aim is to minimize the length of the test for covering all the goals. The verdict "failure" with trace is given when a non-conforming behaviour is found and "success" if all the goals are covered with conforming behaviour. A formal definition and discussion about conformance is in Section 4.2. The detailed description of the testing process and subtasks involved is in Chapter 6.

The test procedure for a non-deterministic model selects the next test goal and input on-the-fly depending on the state of the system. The next test goals and inputs are selected by solving the constraints for input parameters using a constraint solver.

A simplified view of the steps of the online testing process are shown in Table 2.1. The column Goal describes which trap is selected as the next goal. An optimal approach would be to cover trap2 first and then trap3 together with trap1. The Input and Output columns describe interactions between the tester and the IUT. The State column describes the state of the model after the interaction and the last columns shows what traps are covered.

Every row of the table describes one interaction with the IUT. There are no non-conforming interactions present and we can conclude that the test was successful and all the goals were covered. At first trap2 is taken as a goal. A value is selected for input parameter val from the set of possible values $\{1, 5, 20\}$, such that the guarding constraint $\mathsf{val} < 20$ is satisfied. Value 5 is selected and communicated to the IUT as the parameter of input $\mathsf{coin}(5)$ and output $\mathsf{msg}(5)$ is expected back from the IUT. Any other response would be regarded as non-conforming behaviour. The only possible result of the simulation of the model with the input and output shown is the location $\mathsf{l}_<$ and state variable sum having value 5.

The second step results in covering trap2. Any possible value is suitable for the constraint $\mathsf{val} \geq 20$ and lets assume that the value 1 was selected. The goal changes to trap3 after that step. The only enabled guard is still the self-loop of the location $\mathsf{l}_<$. The guarding constraint $\mathsf{sum} \leq 20 \wedge \mathsf{sum} + \mathsf{val} > 20$ for reaching trap3 is a little more complex. Solving it in the state $\mathsf{sum} = 6$ means that a substituted constraint $6 \leq 20 \wedge 6 + \mathsf{val} > 20$ should be solved and the only possible solution is $\mathsf{val} = 20$.

Non-deterministic behaviour is possible in the step 4. The state $(\mathsf{l}_<, \mathsf{sum} = 26)$ allows both returning to l_0 and progressing to $\mathsf{l}_\geq$. Lets assume that the IUT chooses the first option and returns all the coins. The tester should be able to cope with such a deviation from the planned path and the strategy starts to guide the process again from the location l_0. The steps 5 and 6 are similar to steps 1 and 3, except a different solution is found and a coin with value 1 is inserted. There is again a non-deterministic choice made by the IUT for step 7. Lets assume that the desirable choice is done this time and the brewing process starts with grinding. The last step covers both traps trap1 and trap3. This happened, because the trap with stronger condition was chosen as a goal. Otherwise, if a latte had been served for amount 20, then trap3 have been left uncovered.

2.5 BACKGROUND THEORIES

The example is used on models expressed by EFSM with linear constraints and updates, but the method is not limited to this. It can be applied to different background theories as well. The theoretical limitation is the ability to calculate the weakest preconditions of the updates and to check satisfiability and solve the constraints in the test strategy. This constrains the theories to those where the SMT problem is decidable. The ability to eliminate existential quantification and simplify the constraints is important for the feasibility of the satisfiability check and model generation, but no normal forms or other strict requirements are placed on the simplification.

The set of possible background theories is limited to the theories supported by Boolean Satisfiability Problem (SAT) and SMT solvers from the practical point of view. This includes FSM models and Binary Decision Diagram (BDD) based symbolic representation and EFSM models with integer or real arithmetic, linear or non-linear functions, arrays, bit-vectors, and many algebraic data types. Using higher level theories complicates the elementary steps of satisfiability check and simplification. But it helps to express the same properties in a more concise and natural way on the other hand. Also the applicability of theory-specific higher level simplification may help to cope with much larger state spaces. The typical example is to use integers with inequalities and arithmetical operations instead of bit vectors. The feasibility of the test generation for a concrete model depends on the structure of the model and tools available to support the decision procedures in the chosen background theory.

3

A FORMAL FRAMEWORK

We define a formal framework for modelling an IUT in this chapter. In model based testing approach the tests are generated from a formal model of an IUT and test goals. In Section 3.2 we propose a formalism of parametrised *Input/Output Extended Finite State Machine* (I/O-EFSM). We extend the formalism of EFSM used for test generation in [65] by data communication in form of input/output parameters, infinite domains of state variables and relaxing condition of path feasibility. The approach is inspired by Statecharts [33, 16] and other similar formalisms of Mealy style input/output and extended finite state machines [56, 43] where the communication takes place in interactions consisting of input and output. This kind of approach is common in game theory [5, 71] and the generation of testing strategy can be viewed also as finding a winning strategy of a two-player game. We also define a property of *output-observability* of I/O-EFSM that can be viewed as limited non-determinism and is a useful property for more efficient test generation.

Parts of the model, test goals and results are expressed in the logical framework of a first-order theory which will be described in the following section. For a more detailed description we refer to [39]. The procedures of test generation introduced in Chapters 5 and 6 are based on the concepts of the logical framework developed in this chapter.

3.1 LOGICAL FRAMEWORK

We use a *first-order language* over a set of variables X and a signature $\Sigma = (\mathcal{P}, \mathcal{F}, \mathcal{A})$, where $\mathcal{P}$ is a set of predicate symbols, $\mathcal{F}$ is a set of function symbols, and $\mathcal{A}$ is the arity function that assigns a natural number to every element of $\mathcal{P}$ and $\mathcal{F}$.

3.1.1 *Syntax of the first-order language*

The language includes *propositional connectives* $\wedge, \vee, \rightarrow, \neg$, *quantifiers* $\exists, \forall$, *terms* τ, and *formulas* Φ.

Definition 3.1. The set of *well-formed terms* τ is defined inductively as follows:

- a variable is a term: $x \in \tau$, for all $x \in X$,
- an application of a function to a sequence of terms is a term: $f(t_1, \ldots, t_n) \in \tau$, where $f \in \mathcal{F}$, $t_i \in \tau$, $i \in [1, n]$, and $\mathcal{A}(f) = n$

Definition 3.2. The set of *well-formed formulas* ϕ is defined inductively as follows:

- **true** and **false** are formulas
- an application of a predicate to a sequence of terms is a formula: $p(t_1, \ldots, t_n) \in \Phi$, where $p \in \mathcal{P}$, $t_i \in \tau$, and $\mathcal{A}(p) = n$;
- a Boolean combination of formulas is a formula: $\phi \wedge \varphi, \phi \vee \varphi, \phi \rightarrow \varphi, \neg\phi \in \Phi$, where $\phi, \varphi \in \Phi$
- formula with a variable bound by a quantifier is a formula: $\exists x : \phi, \forall x : \phi \in \Phi$, where $\phi \in \Phi, x \in X$

Moreover:

constant is the function with arity 0

ground term is the term, which does not include variables; a *ground formula* is the formula including only ground terms

quantifier free formula is the formula, which does not include quantifiers

free variable is the variable in the formula, that is not bound by any quantifier

sentence is a formula with no free variables

3.1.2 *Semantics of the first-order language*

The semantics of the first-order language is defined using a model $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ and variable assignment α. The domain $\mathcal{D}$ is a non-empty set. The interpretation $\mathcal{I}$ assigns a function $f^{\mathcal{M}} : \mathcal{D}^n \mapsto \mathcal{D}$ to each function $f \in \mathcal{F}$ with arity $\mathcal{A}(f) = n$ and a relation $p^{\mathcal{M}} \subseteq \mathcal{D}^n$ to each predicate $p \in \mathcal{P}$ with arity $\mathcal{A}(p) = n$. An assignment function α assigns an element $x^{\alpha} \in \mathcal{D}$ to every variable $x \in X$. An *assignment update* $\alpha[x \mapsto d]$ denotes an assignment α', where $\alpha'(x) = d$ and $\alpha'(y) = \alpha(y)$ for all variables $y \in X$, such that $y \neq x$, i.e. α' is an assignment which is the same as α except that x is mapped to d.

Definition 3.3. The interpretation of terms is given as a valuation function $[\![]\!]^{\mathcal{M},\alpha} : \mathcal{D}^n \mapsto \mathcal{D}$ defined recursively as follows:

- $[\![x]\!]^{\mathcal{M},\alpha} = \alpha(x)$ for all $x \in X$
- $[\![f(t_1, \ldots, t_n)]\!]^{\mathcal{M},\alpha} = f^{\mathcal{M}}([\![t_1]\!]^{\mathcal{M},\alpha}, \ldots, [\![t_n]\!]^{\mathcal{M},\alpha})$ for all $f \in \mathcal{F}$ with $\mathcal{A}(f) = n$ and $t_i \in \tau$

Definition 3.4. The interpretation of the formulas is given as a *satisfiability* relation $\mathcal{M}, \alpha \models \phi$ (a *formula* ϕ is satisfied or true in the model $\mathcal{M}$ under the assignment α) and defined recursively as follows:

- $\mathcal{M}, \alpha \models$ **true**
- $\mathcal{M}, \alpha \not\models$ **false**
- $\mathcal{M}, \alpha \models p(t_1, \ldots, t_n)$ iff $([\![t_1]\!]^{\mathcal{M},\alpha}, \ldots, [\![t_n]\!]^{\mathcal{M},\alpha}) \in p^{\mathcal{M}}$ for all $p \in \mathcal{P}$ with $\mathcal{A}(p) = n$ and $t_i \in \tau$
- $\mathcal{M}, \alpha \models \neg\phi$ iff $\mathcal{M}, \alpha \not\models \phi$
- $\mathcal{M}, \alpha \models \phi \vee \varphi$ iff $\mathcal{M}, \alpha \models \phi$ or $\mathcal{M}, \alpha \models \varphi$
- $\mathcal{M}, \alpha \models \phi \wedge \varphi$ iff $\mathcal{M}, \alpha \models \phi$ and $\mathcal{M}, \alpha \models \varphi$
- $\mathcal{M}, \alpha \models \exists x : \phi$ iff $\mathcal{M}, \alpha[x \mapsto d] \models \phi$ for some $d \in \mathcal{D}$
- $\mathcal{M}, \alpha \models \forall x : \phi$ iff $\mathcal{M}, \alpha[x \mapsto d] \models \phi$ for all $d \in \mathcal{D}$

Definition 3.5. An assignment ν to the variables $\overline{x}$ is a partial satisfying assignment for φ if

$$\nu \models \exists \overline{y} : \varphi$$

where $\overline{y}$ is the free variables of φ that are not in $\overline{x}$.

I.e., a partial assignment for φ is a assignment that is extensible to a satisfying assignment for φ.

A formula ϕ is *satisfiable* if there is a model $\mathcal{M}$ and assignment α, such that $\mathcal{M}, \alpha \vDash \phi$. Otherwise the formula is *unsatisfiable*. If the formula ϕ is satisfiable, so is its *existential closure* $\exists \vec{x} : \phi$, where $\vec{x}$ is the set of free variables in ϕ.

A formula ϕ is *valid* (written as $\vDash \phi$), if for all models $\mathcal{M}$ and assignments α it holds that $\mathcal{M}, \alpha \vDash \phi$. The *universal closure* $\forall \vec{x} : \phi$ of a formula ϕ is valid iff its negation is not satisfiable $\nvDash \neg\phi$.

3.1.3 *Theories*

A *first-order theory* $\mathcal{T}$ over signature Σ is a set of first-order sentences over Σ. The theory $\mathcal{T}$ fixes a class of models $\mathcal{C}$ (sometimes called *intended models*) in which the sentences are true: $\mathcal{M} \in \mathcal{C}$ iff $\mathcal{M}, \alpha \models \psi$ for all $\psi \in \mathcal{T}$ and α. A formula ϕ is said to be satisfiable modulo $\mathcal{T}$ if it is satisfiable in this class of models, i.e., if there is a model $\mathcal{M} \in \mathcal{C}$ and assignment α such that $\mathcal{M}, \alpha \models \phi$. We write also $\alpha \vDash \phi$ to denote that the formula ϕ is satisfied under a (partial) assignment α when we can assume that the background theory is known and the theory determines a single model $\mathcal{M}$.

Example 3.1. The theory of Presburger arithmetic (also called linear integer arithmetic) is a first-order theory over signature

$$(\{\leq\}, \mathbb{N} \cup \{+, -\}, \{\langle \leq \mapsto 2\rangle, \langle n \mapsto 0\rangle, \langle + \mapsto 2\rangle, \langle - \mapsto 2\rangle\}),$$

where $n \in \mathbb{N}$. It consists of all true sentences in the model given by $\mathbb{N}$ as the carrier and the usual arithmetic interpretation of $\leq, +, -$ and numerical constants. In the assignment $\alpha = [x \mapsto 5][y \mapsto 7]$ the term $x + y$ is interpreted as $12 \in \mathcal{D}$ as follows

$$[\![x + y]\!]^{\mathcal{M},\alpha} = [\![x]\!]^{\mathcal{M},\alpha} +^{\mathcal{M}} [\![y]\!]^{\mathcal{M},\alpha} = \alpha(x) + \alpha(y) = 5 + 7 = 12$$

The sentence $5 \leq 7 \wedge \forall x : \exists y : x + 1 \leq y$ is true in the intended model and hence belongs to the theory, because there exists a number $y = x + 1 \in \mathbb{N}$ for any other natural number $x \in \mathbb{N}$. But the sentences $\forall x : \exists y : \neg x \leq y$ and $\exists y : \forall x : x \leq y$ are false and do not belong to the theory, because all the natural numbers in $\mathbb{N}$ are greater or equal to 0 and none is bigger than any other natural number in the interpretation of $\leq$.

In the context of the current work we consider the theories for which the SMT (Satisfiability Modulo Theories) problem is decidable and also an efficient decision procedures and solvers are available. The other important feasibility aspect is the availability of the efficient existential quantifier elimination and algebraic simplification procedures.

3.2 MODEL OF THE IUT

Test generation needs the IUT to be represented by a formal model. We propose a formalism of *Input/Output Extended Finite State Machine* (I/O-EFSM) for that purpose.

3.2.1 *Syntax of I/O-EFSM*

We define a model of an IUT using a parametrized *Input/Output Extended Finite State Machine* I/O-EFSM($\mathcal{T}_\Sigma$) over the first-order theory $\mathcal{T}_\Sigma$.

Definition 3.6. *Input/Output Extended Finite State Machine* (I/O-EFSM($\mathcal{T}_\Sigma$)) M (also called *automaton*) over a first order theory $\mathcal{T}_\Sigma$ is a tuple (L, l_0, X, D, I, O, E), where

- L is a finite set of *locations*, $l_0 \in L$ is an initial location;
- X is a set of *variables*, consisting of the disjoint sets of *state variables* X_s, *input variables* X_i, *output variables* X_o, and *auxiliary variables* X_a;
- $D \in \mathcal{T}_\Sigma(X)$ is a well-formed formula of the theory that specifies the *domain* of the variables. The predicate D is needed to constrain the value domain of variables, because the automaton is defined over single-sorted language;
- I is a set of *input labels* including symbol ϵ for a missing input. Every input label $i \in I$ may have an associated n-tuple of input variables $\overline{x}_i \in X_i^n$ called *formal parameters* $par(i) = \overline{x}_i$. $|par(i)| = n$ is a natural number denoting the number of parameters of the label.
  $|par(\epsilon)| = 0$;
- O is a set of *outputs* including symbol ϵ for a missing output. Every output label $o \in O$ may have an associated n-tuple of output variables $\overline{x}_o \in (X_o \cup X_s)^n$ called *formal parameters* $par(o) = \overline{x}_o$. The state variables can be used as formal parameters of output also. $|par(o)| = n$ is a natural number denoting the number of parameters of the label. $|par(\epsilon)| = 0$;
- E is a finite set of *edges*. An edge $e \in E$ is a tuple $(l, i(\overline{x}_i), o(\overline{x}_o), g, U, l')$, written also as $l \xrightarrow{i(\overline{x}_i)/o(\overline{x}_o)[g]U} l'$ where

- $l \in L$ is the source location
- $l' \in L$ is the target location
- $i(\overline{x}_i)$ is an *input port* where $i \in I$ is an input label and $\overline{x}_i = par(i)$ are its formal parameters
- $o(\overline{x}_o)$is an *output port* where $o \in O$ is an output label and $\overline{x}_o = par(o)$ are its formal parameters
- g is a well-formed formula of the theory $\mathcal{T}_\Sigma(X_s \cup X_i)$ over the state and input variables named *guard.*
- U is a list of *updates* in the form $x_1 := expr_1, \ldots, x_n := expr_n$, where $x_i \in X_s \cup X_o \cup X_a$ and $expr_i \in \tau$ is a well-formed term of $\mathcal{T}_\Sigma(X_s \cup X_i)$ or $x_i \in X_a$ and $expr_i \in \tau$ is a well-formed term of $\mathcal{T}_\Sigma(X_s \cup X_i \cup X_a)$.

We use *locations* and *edges* on the syntactic level of the automaton, whereas corresponding terms *state* and *transition* are used on the semantic level and defined in the next section. This may differ from some other approaches.

The theory $\mathcal{T}_\Sigma$ determines the allowed types of the variables, operators and functions in the guards and updates. The theory may restrict variables to Booleans, reals, integers, arrays or other algebraic data types and operations to linear arithmetic, finite structures and so on. The different theories differ in expressive power, complexity of the solver procedures and tool support. We present the method of test generation independently of the theory chosen, but we require that the SMT (Satisfiability Modulo Theories) problem is decidable for the used formulas and chosen background theories. In the mixed theories, where the variables can belong to different types we assume that the predicates defining the type (e.g., $Int(x), Set(x)$) are part of the theory and can be used in D to constrain the possible values.

The symbol ϵ refers to a missing or refused action (input or output). It is considered as an intentional behaviour from the party (tester or IUT) and understood as such by the other party. It can model a predefined timeout in case the decision is made based on the action or a mere fact that the other party made an internal unobservable action in case of deterministic behaviour. The actual interpretation depends on the application and state. Thus ϵ can be taken as a legal action symbol, even if the real communication is not taking place. An ϵ output is considered to be received when no output is received in the process where some output is expected. The ϵ action assumes that it is possible to model and interface an IUT such that the possible race conditions, because of the communication delays, buffering, system load and scheduling do not interfere with the testing process.

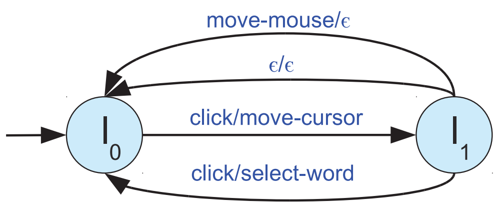


Figure 3.1: Mouse double-click

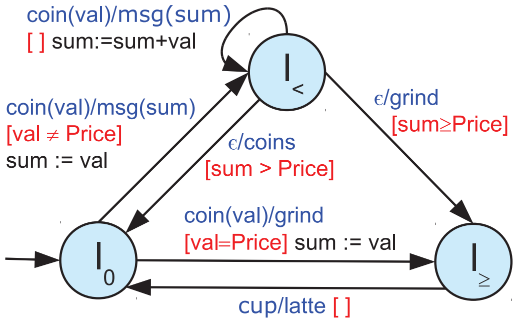


Figure 3.2: I/O-EFSM model of vending machine

A testing theory with buffered input-output is considered in [57]. An ϵ input is useful for modelling double-click like behaviour as in Figure 3.1.

Example 3.2. The formal definition of the vending machine introduced in Chapter 2 and depicted in Figure 3.2 is a follows:

$$
\begin{array}{rcl}
L & = & \{l_0, l_<, l_\geq\} \\
l_0 & = & l_0 \\
X_s & = & \{sum\} \\
X_i & = & \{val\} \\
X_o & = & \emptyset \\
X_a & = & \emptyset \\
D & = & Int(sum) \wedge Int(val) \wedge sum \geq 0 \wedge val \in \{1,5,20\} \wedge Price = 20 \\
I & = & \{coin, cup, \epsilon\} \\
O & = & \{msg, coins, grind, latte, \epsilon\} \\
par(k) & = & \{\langle coin \mapsto 1\rangle, \langle cup \mapsto 0\rangle, \langle msg \mapsto 1\rangle, \langle coins \mapsto 0\rangle,) \\
 & & \langle grind \mapsto 0\rangle, \langle latte \mapsto 0\rangle, \langle \epsilon \mapsto 0\rangle\} \\
E & = & \{(l_0, coin(val), msg(sum), val \neq Price, sum := val, l_<), \} \\
 & & (l_0, coin(val), grind, val = Price, sum := val, l_\geq), \\
 & & (l_<, coin(val), msg(sum), true, sum := sum + val, l_<), \\
 & & (l_<, \epsilon, coins, sum > Price, , l_0), (l_<, \epsilon, grind, sum \geq Price, , l_\geq), \\
 & & (l_\geq, cup, latte, true, , l_0)
\end{array}
$$

Please note that if we restrict the theory $\mathcal{T}_\Sigma$ to pure Presburger arithmetic (Example 3.1), then $\leq$ is the only comparison predicate defined and we should rewrite all comparisons that use some other predicate in terms of $\leq$. E.g. the correct domain predicate D would be following in that case:

$$D = (val \leq 1 \wedge 1 \leq val) \vee (val \leq 5 \wedge 5 \leq val) \vee (val \leq 20 \wedge 20 \leq val)$$

and 20 substituted for *Price* in all other formulas.

The input variables X_i and output variables X_o are not considered as part of the state of an automaton, but an implicit environment. The update functions are not allowed to assign values to input variables and the behaviour of the automaton does not depend on the value of output and auxiliary variables. The auxiliary variables are not allowed to occur in the guards and right hand of the assignments to state variables and do not affect the behaviour of the automaton because of that. They may be regarded as dependent or history variables and used for specifying test goals. The difference of auxiliary and output variables is that the output variables do not belong to the state and have a value only during the transition. The main function of the output variables is to give a possibility to calculate a value based on the state variables that is revealed on the interface. But the state can be revealed directly also. For instance in the example above the variable *sum* is a state and not output variable although it is used as a parameter to output port *msg*.

Furthermore, the following functions are defined on edges $e = (l, i(\overline{x}_i), o(\overline{x}_o), g, U, l')$:

$source(e) = l$ is the source location of the edge

$target(e) = l'$ is the target location of the edge

$guard(e) = g$ is the guard of the edge

$in(l) \subseteq E$ is the set of incoming edges $\{e' | e' \in E \text{ and } target(e') = l\}$

$out(l) \subseteq E$ is the set of outgoing edges $\{e' | e' \in E \text{ and } source(e') = l\}$

3.2.2 *Interaction transition system*

We define an *interaction transition system* ITS($\mathcal{T}_\Sigma$) for describing the semantics of an I/O-EFSM($\mathcal{T}_\Sigma$) (L, l_0, X, D, I, O, E).

Definition 3.7. An interaction transition system ITS($\mathcal{T}_\Sigma$) is a tuple $(\mathcal{S}, A, T)$ where:

- $\mathcal{S}$ is a set of states. A state $s \in \mathcal{S}$ is a pair (l, α) of a location $l \in L$ and assignment α of the variables in $X_s \cup X_a$. The assignment satisfies the domain of the variables $\alpha \vDash D$.
- A is a set of *interactions*. An interaction $a \in A$ is a pair $(i(\overline{v}_i), o(\overline{v}_o))$ of an input and output action. An *input action* $i(\overline{v}_i)$ consists of an input label $i \in I$ and a tuple of actual parameters (values) $(\overline{v}_i) = (v_1, \dots, v_n)$, when $n = |par(i)| > 0$. An output action is defined in the same way. (ϵ, ϵ) denotes a transition without interaction.
- $T \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a set of transitions. We use a notation $(l, \alpha) \xrightarrow{(i(\overline{v}_i), o(\overline{v}_o))} (l', \alpha')$ or $s \xrightarrow{a} s'$ for $(s, a, s') \in T$. A *transition* $(l, \alpha) \xrightarrow{(i(\overline{v}_i), o(\overline{v}_o))} (l', \alpha') \in T$ if there is an edge $l \xrightarrow{i(\overline{x}_i)/o(\overline{x}_o)[g]U} l'$ and
  - if $i \neq \epsilon$, an input action $i(\overline{v}_i)$ is received.
  - the transition is *enabled*, meaning that the conjunction of the guard and domain is satisfiable $\alpha'' \models g \wedge D$ in an assignment α'' where the input variable assignments are updated by the input parameters $\alpha'' = \alpha[x_1 \mapsto v_1]...[x_n \mapsto v_n]$ for each $x_i \in par(i)$.
  - the assignment α' in the target state is the result of applying the updates $U = [x_1 := t_1, \dots, x_n := t_n]$ to the assignment α'' , such that $\alpha_1 = \alpha''$, $\alpha_{i+1} = \alpha_i[x_i \mapsto [\![t_i]\!]^{M,\alpha_i}]$ for $i = 1 \dots n$, and α' is α_{n+1} restricted to $X_s \cup X_a$
  - if $o \neq \epsilon$, an output action $o(\overline{v}_o)$ is sent, where the parameters $v_o = (v_1, \dots, v_n)$ correspond to the assignment of output variables $v_o = [\![x_o]\!]^{M,\alpha_{n+1}}$ for $x_o \in par(o)$.

By *input action* we mean a pair of input label and assignment of input parameters. The *output action* is a pair of output label and assignment of output variables.

We consider a transition to be atomic although it contains several internal sub-states for receiving an input, updating state variables one after another and sending an output. The semantics is defined in this way to enable modelling of atomic interactions. Modelling one-way interaction where only input or output is present and transitions without interaction are possible also by using ϵ as a symbol of implicit action.

The input and output variables have an assigned value only during transitions and can be regarded as local to transition. A state is defined by the partial assignment α to state and auxiliary variables.

Definition 3.8. A *run* of the automaton is a sequence of states $\langle s_1, \dots, s_k \rangle$, so that the states are related by the transition relation $s_i \xrightarrow{a} s_{i+1}$ for all the states $i = 1 \dots k-1$. A *feasible run* starts from the initial state (l_0, α_0), where the initial assignment α_0 of the variables is defined by their type, e.g., all numerical variables are initialized to 0, Booleans to $false$, sets to $\emptyset$ and so on. No special construction for initialization is defined, but it is always possible to model initialization by a single edge leaving from the initial location. The *length of a run* $|\langle s_1, \dots, s_k \rangle| = k - 1$ is the number of transitions in it. An automaton is not well-defined if it has a feasible run with infinite number or consequent (ϵ, ϵ) actions.

Definition 3.9. An *observable behaviour* of an automaton is a finite sequence $\langle a_1, \dots, a_k \rangle$ of interactions of some feasible run, from where unobservable (ϵ, ϵ) transitions are removed. The (ϵ, ϵ) transition is unobservable, when it follows to $(*, \epsilon)$ or precedes to $(\epsilon, *)$ interaction, i.e. it is observable only when it follows to the real output and precedes to the real input. A sequence of (ϵ, ϵ) transitions is observable as a single (ϵ, ϵ) interaction.

$\mathcal{O}(M)$ is the set of all observable behaviours of automaton M. $\mathcal{O}(M)$ has a *prefix inclusion* property:

$$\langle a_1, \dots, a_k, a_{k+1} \rangle \in \mathcal{O}(M) \implies \langle a_1, \dots, a_k \rangle \in \mathcal{O}(M), \text{ for all } k \geqslant 1$$

Example 3.3. Two runs of the automaton depicted in Figure 3.2 are

$$\langle (l_0, sum \mapsto 0), (l_<, sum \mapsto 5), (l_<, sum \mapsto 25), (l_0, sum \mapsto 25) \rangle$$
$$\langle (l_0, sum \mapsto 0), (l_<, sum \mapsto 5), (l_<, sum \mapsto 25), (l_\geq, sum \mapsto 25) \rangle$$

with the following corresponding observable behaviours

$$\langle (coin(5), msg(5)), (coin(20), msg(25)), (\epsilon, coins) \rangle$$
$$\langle (coin(5), msg(5)), (coin(20), msg(25)), (\epsilon, grind) \rangle$$

The runs differ only by the location of the last state and different output in the last interaction.

3.2.3 *Output-observability*

The output-observability or observability is a well-know term in the control theory, but also used in connection to using EFSMs and FSMs in the testing [56, 71]. The intuitive meaning is that the internal state of the automaton is detectable by its external behaviour, i.e. by knowing the initial state and the sequence of input and output actions. The output observability property gives one to one correspondence between the run and observable behaviour of the automaton. It does not mean that the internal state of the IUT is detectable, but that the relevant part of the state of the IUT what determines the observed behaviour is modelled by an unique state of the model. We require in the following that the model of the IUT is given as an output observable I/O-EFSM.

Definition 3.10. An ITS and I/O-EFSM is *output-observable* iff for all states (l, α), such that $l \in L, \alpha \models D$ and input action $i(\overline{v}_i)$, such that $i \in I$, $\overline{v}_i \models D$ it holds that

- if there are two transitions with the same source state and input action $(l, \alpha) \xrightarrow{(i,o')} (l', \alpha')$ and $(l, \alpha) \xrightarrow{(i,o'')} (l'', \alpha'')$ the output action distinguishes the destination states $(l', \alpha') \neq (l'', \alpha'') \implies o' \neq o''$.
- if there is a sequence of transitions with consequent ϵ output and input $(l''', \alpha''') \xrightarrow{(i,\epsilon)} (l, \alpha)$ and $(l, \alpha) \xrightarrow{(\epsilon,o)} (l'', \alpha'')$ then there is no transition $(l, \alpha) \xrightarrow{(i',o')} (l', \alpha')$ such that $l' \neq l'' \vee \alpha' \neq \alpha''$ that can lead to different state. The ϵ-input action following to ϵ-output action is chosen deterministically based on the state.

It follows from the output-observability that the combination of source state, input and output actions of a transition identifies the next state of the IUT unambiguously.

The output-observability could be defined in the level of syntax, semantics or behaviour. We have chosen the semantic level as a compromise between the expressibility of the model and possibility of static checking. The following example illustrates the difference.

Example 3.4. The automaton in Figure 3.3

- is not syntactically output-observable. It would need a different output label on one of the alternative edges.
- is semantically output-observable if $D \implies x > 2 \wedge u > 2$. I.e. it is possible to check in the background theory that $\not\models D \wedge x + u = x * u$.

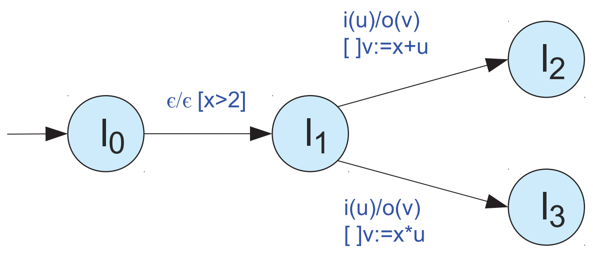


Figure 3.3: Output-observability

- is behaviourally output-observable if $D \implies u > 2$. I.e. for reachable states (l_1, α) it is possible to show that $\alpha \nvDash D \wedge x + u = x * u$.

The model of the vending machine in the examples is also output-observable.

Output-observability can be viewed as limited (one step) non-determinism. We have found this to be a useful and practical subclass of automaton both from the perspective of modelling and test generation. The testing procedure described in Section 6.1, specially the simulation part is based on the output-observability assumption. The output-observability property of the model enables us to relax the input-enabledness requirement on the IUT as discussed in Section 4.2. We have defined the I/O-EFSM in Mealy style to be able to define and build output-observable models intuitively. We elaborate on the consequences and relations to other formalisms in Section 3.2.4.

3.2.4 *Relation to Labelled Transition Systems of IOCO-testing theory*

The LTS of IOCO-testing theory [59] and its extension to STS with data variables [26] form a well established semantic framework for model-based testing. We do not base our work on these results as Mealy style EFSM and labelled transition systems form a close, but a separate tradition of modelling and there are some subtle differences in the semantics. The goal of the subsection is to give an overview and intuition of the differences of the paradigms without going to formal details.

The main difference is that a transition of LTS can have either an input or an output associated with it, but not both. An STS [26] can be defined to correspond to an I/O-EFSM by splitting each edge to two elements of the STS switch relation with an intermediate location between the input and output action. An non-deterministic output-observable I/O-EFSM with corresponding the STS is depicted in Figure 3.4. It is easy to see why the output-observability can be characterized as one step non-determinism. It is non-deterministic for input, but determined if input and output are

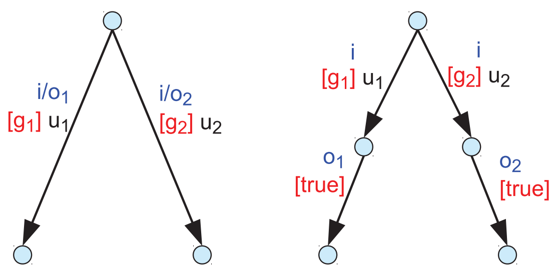


Figure 3.4: I/O-EFSM and STS

taken as one interaction. The other semantic difference is that the scope of the input and output variables is global for the STS, but local to transition for I/O-EFSM.

Quiescence

The *quiescence* defined for STS framework is a central term in the IOCO-testing theory. It is defined as a property of a state, where no output or internal transition is possible, i.e. all the outgoing transitions require an input. The LTS model is augmented with δ-loops for such states to denote that quiescence is observable in these states. E.g. the initial and final states of Figure 3.4 are quiescent. The δ-transitions are included in the *suspension traces* that express the observable behaviours. If the IUT is found to be quiescent during the testing then the observation of δ-transitions is assumed and the current state of the specification model is restricted to quiescent states or concluded that the IUT is non-conforming if no quiescent state is reachable with the observed suspension trace. This is a kind of implicit quiescence derived from the absence of output.

There is no need for adding δ-loops into the model in case of output-observable I/O-EFSMs. All the states without outgoing ϵ-input transition can be considered quiescent, where the system is waiting for the next input. The consequences of observing quiescence is taken into account directly in the testing procedure explained in Chapter 6. There is no need to constrain the set of states using the observation of quiescence, because the output-observability of the model constrains it to single state anyway. And observing quiescence when real output is expected (no ϵ-output allowed) is considered to be a non-conforming behaviour. In addition, the I/O-EFSM offers a possibility to model explicit quiescence in form of ϵ-input and ϵ-output, so that the IUT and tester can (non-deterministically) choose if they send the action other party is waiting for or refuses to send and delivers quiescence. This is another form of limited non-determinism that an output-observable I/O-EFSM offers. ϵ-output can be modelled by a non-deterministic τ-switch and ϵ-output by a special input symbol in STS model [26].

Composition

An important property of LTS based formal models is the well-defined parallel composition. It is known that it is difficult to define a natural parallel composition for Mealy style EFSM based models [38, 69]. Actually there are at least two different ways a parallel synchronous composition have been used in the context of model-based testing.

One is to synchronize on the transitions with the same label and interleave all other transitions. One application of that kind of composition is used for defining conformance in the parallel composition of an implementation and specification [59, 67]. Another application is composing the model of the IUT to the scenario automata to constrain the set of possible behaviours during the test and guard the testing towards a goal [69, 40]. This kind of parallel composition is applicable to I/O-EFSM models when taking the whole interaction as a single label. The synchronization to only input or output actions is more problematic. This kind of composition reduces non-determinism and also preserves output-observability if the hiding of actions is not used.

The second kind of parallel composition is handshake synchronization for composing the model from components, e.g., the parallel composition of timed automata for testing with Tron [35] originating from CSP [36] style synchronization mechanism. Inputs and outputs are synchronized to each other and hidden from the product of the composition. This kind of composition is problematic for Mealy style EFSMs and the output-observability would be not preserved also because of hiding the labels.

3.3 SUMMARY

We have defined a formal framework for model-based testing in this chapter including first order language and I/O-EFSM over a first order theory with its syntax and semantics. Also the output-observability property of the model of the IUT was defined that requires an internal state of the model to be uniquely detectable by observing its I/O-behaviour. It is assumed in the following chapters that the models have the output-observability property. Also the meaning, possibilities to observe, cause and represent the quiescence were discussed.

The I/O-EFSM formalism is proposed for feasible coverage oriented model-based testing of IUTs with structural coverage oriented test purposes. The property of output-observability is motivated by the possibility of efficient online test generation and evaluation that cannot be avoided for non-deterministic models. Restricting models by output-observability is a compromise between the expressibility of non-deterministic behaviours and feasibility of the test generation. I/O-EFSMs provide an intuitive way for building such models. Defining a generic semantic framework as LTS, STS and GLAS models [59, 26, 67] is not an intention of this thesis. Our approach is oriented to support feasibility and practical usability of MBT processes, with the main focus on automated test generation and execution.

4 TEST SPECIFICATION

The model of the IUT is a black-box view of the system that expresses the behaviour of the IUT on the observable testing interface. This model could come from expectations, requirements, design or concrete knowledge about the implementation, but the origin of the model is not important for our approach. The relation of the IUT, model and tester is described in Section 4.1. The goal is to test if the IUT behaves as specified by the model. This is formalized as a conformance relation in Section 4.2. Testing often presumes more targeted traversal of the IUT model than just a random walk. In Section 4.3 we propose a concept of *traps* to express such goals. The use of traps is presented also in [42, 64]. We extend the traps used for test goal specification in [65] by trap conditions for being able to express data-dependent conditions more naturally. Finally a simple approach for using model-checking for testing scenario generation is show in Section 4.4.

4.1 TESTING PROCESS

Execution of a test means that an IUT and a tester run in parallel and communicate through input-output actions. The tester acts as an artificial environment to the IUT generating input actions and analysing output actions. Many approaches [59, 35, 66] share the view that a test case or test suite is generated as an automaton or a collection of automata and its outputs connected to the inputs of the IUT and vice versa. We do not produce test automaton, but generate the test stimuli and check the response of the IUT on-the-fly. So it is better to think of a tester as conformance checking engine that generates input, receives output from the IUT and simulates the same input and output on the model as depicted in Figure 4.1. Similar approach is used also in [26] and other frameworks where tests are generated from the non-deterministic models with data components to avoid generating infeasibly large or infinite tester automatons.

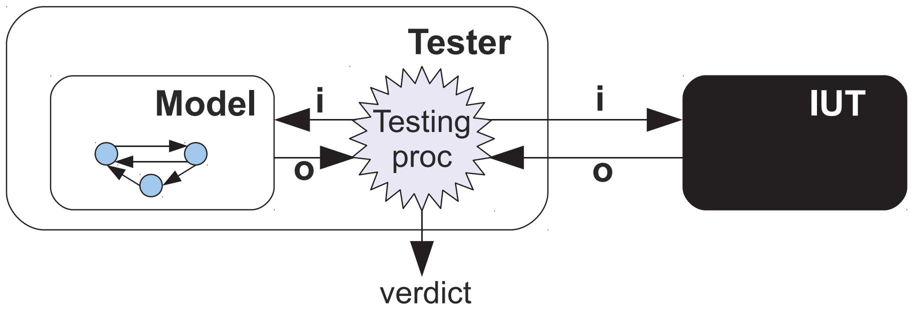


Figure 4.1: Testing process

The communication is captured in interactions that are pairs of an input and output actions, where one of those may be missing. Every interaction corresponds to a transition of the model of the IUT. The tester is usually responsible for initiating the interaction by giving an input. Based on the output (or absence of the output) it can finish the test giving a test verdict or initiate the next interaction. The test verdict "passed" is given when the test purpose is achieved and resulting behaviour conforms to the model (is an observable behaviour of the model). The test verdict "failed" is given when the behaviour does not conform to the model. Also a verdict "inconclusive" may be given when a timeout is associated to the testing process and the test purpose is not achieved before the timeout. More detailed description of the testing process is described in the Chapter 6.

4.2 CONFORMANCE

The goal of the testing process is to confirm that the IUT (implementation) conforms to the model (specification), at least to the extent covered by the tests or to show a behaviour of the IUT that does not conform to the model. A comprehensive overview of different conformance relations and the definition of the well-known IOCO relation is given in [59] and its lifting to STS with data components in [26]. It has been shown in [67] that the IOCO relation can be viewed as an alternating simulation [4] relation.

The intuitive meaning of the alternating simulation relation between the model and implementation is that the model can accept only inputs the implementation can accept and implementation can produce only outputs that the model can produce illustrated in Figure 4.2. The implementation and model are in the alternating simulation relation when all the behaviours where the input is chosen by the model and output by the implementation are possible for both. The rationale behind the relation is that the specification may contain more (non-deterministic) freedom of

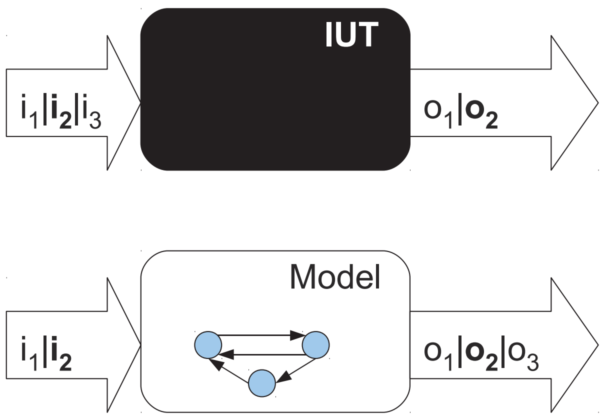


Figure 4.2: Alternating simulation relation

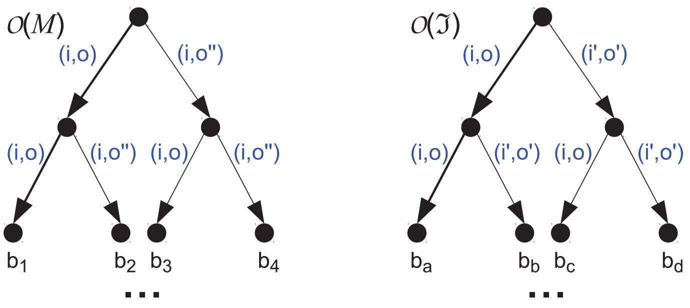


Figure 4.3: Observable behaviours of M and $\mathfrak{I}$

reactions to the same input and the specification may be incomplete and does not specify any reaction to some unspecified input. We define the conformance relation between an IUT and model based on the idea of alternating simulation of the observable behaviours.

Example 4.1. Let us have a model M with one location l and two edges $l \xrightarrow{i/o} l$ and $l \xrightarrow{i/o''} l$ and an implementation $\mathfrak{I}$ with observable behaviours represented by regular expressions $((i,o)|(i',o'))*$. The fragment of observable behaviours of both are depicted in Figure 4.3. Although the leftmost behaviour is the only one common to both, we can say that the implementation $\mathfrak{I}$ conforms to model M because they are in alternating simulation relation. But the relation does not hold other way round. The formal definition of the conformance is given in the following definition.

Definition 4.1. Let $\mathcal{O}(M)$ and $\mathcal{O}(\mathfrak{I})$ be the sets of all finite observable behaviours of model M and implementation $\mathfrak{I}$, i is an input action, o, o' are output actions, ω is a sequence of interactions. An implementation $\mathfrak{I}$ conforms to model M, denoted by $\mathfrak{I} \preccurlyeq M$ iff the following conditions are satisfied for all $\langle \omega, (i,o) \rangle \in \mathcal{O}(M) \cup \mathcal{O}(\mathfrak{I})$

$$\begin{aligned}
\langle\omega,(i,o)\rangle \in \mathcal{O}(M) \wedge \langle\omega\rangle \in \mathcal{O}(\mathfrak{I}) &\implies \exists o' : \langle\omega,(i,o')\rangle \in \mathcal{O}(\mathfrak{I}) \\
\langle\omega,(i,o)\rangle \in \mathcal{O}(\mathfrak{I}) \wedge \langle\omega\rangle \in \mathcal{O}(M) &\implies \langle\omega,(i,o)\rangle \in \mathcal{O}(M) \vee \\
&\qquad \forall o' : \langle\omega,(i,o')\rangle \notin \mathcal{O}(M)
\end{aligned}$$

The conditions apply only to the behaviours with a common prefix up to the last interaction (e.g. b_1, b_2, b_a, b_b in the previous example). The rest of the behaviours are restricted implicitly through the prefix inclusion property and other properties of the output observable behaviours given in Definition 3.9. The first condition formalizes that the implementation must accept as input what is possible according to the model after $\langle\omega\rangle$. The implementation is not forced to be able to give the same output. For instance the pair of behaviours b_2, b_a in Figure 4.3 satisfies the first condition. On the other hand, the second condition formalizes that the model should be able to produce the same output as implementation if the inputs match (e.g. b_1 because of b_a) or the behaviour with the input is unspecified in the model and there is no following interaction with the same input (e.g. no matching behaviour for b_b).

We need only the set of observable behaviours defined for the IUT and nothing about its internal structure for defining conformance. The observable behaviours should have the same form and properties as for I/O-EFSM model given in Definition 3.9. This is actually all we have in the case of black-box testing, we cannot assume any knowledge of the internals of the IUT. This is different with the definitions of IOCO [59] and alternating simulation relation on GLASs [67], where the IUT is assumed to be represented by some kind of LTS. The LTSs must not be directly related to the IUT, but are used as finite representations of possible infinite sets of traces (behaviours). We avoid the assumption and define conformance directly on observable behaviours.

The defined conformance relation is not identical to IOCO, but very similar. One of the reasons is in the modelling formalism and modelling of quiescence discussed in Section 3.2.4. The other difference is the requirements in the IOCO testing theory that an IUT must be represented by an Input Output Transition System (IOTS). An IOTS is an LTS with disjoint sets of input and output labels and being input-enabled for all input labels in all states. It is not quite clear why the requirement of input-enabledness is needed. One way to understand it is that the IUT cannot avoid receiving an arbitrary input in any state and it should react to it somehow, e.g. by quiescence (no observable reaction), responding with some output, giving exception, exploding or in some other way. On the other hand, it seems a little too strong requirement, because a tester should not give arbitrary input in every situation. A theoretical reason for input-enabledness is that the LTS model does not constrain non-determinism and the testing procedure keeps track of the set of states that are consistent with the current trace and where the IUT can be at the moment. That implies that any of the inputs possible from any consistent state may be sent to the IUT and it must be ready to react to it. Being able to accept any input that is possible according to the model (specification) at every given state is a weaker condition than input-enabledness. This is what an IOCO test generation algorithm must assume and alternating simulation relation requires. As we require in general that the models are output-observable, it is possible to constrain the set of consistent states to a single state and the IUT must be able to accept only the inputs of the transitions having this state as a source. One benefit of output-observability is the possibility of not requiring too much robustness from the IUT.

The definition of the conformance relation suggest a simple basic testing algorithm. After the test run resulting in observable behaviour ω and acceptable both to model and the IUT, a new input is generated what is consistent with the possible behaviours of the model. Then the output is expected what is consistent with the behaviour of the IUT. The second condition suggests to check if the interaction generated is also consistent with the model. The process continues from where it was. The second disjunct of the second condition will never be satisfied in the testing process, because the input is never generated that is inconsistent to the model. This basic algorithm is common to many testing frameworks. The essential question is how the input is selected from the potentially large set in each iteration. The tester should be able to choose an input in infinitely many different states in case the data variables are involved and from infinite many inputs in case the input has data parameters. The testing can be random or goal-driven depending on how the selection of input is carried out. A method for goal-driven testing is one of the main contributions of the thesis.

4.3 TEST PURPOSE

The goal of testing is to give some certainty that an IUT conforms to its specification. Testing cannot be used to show conformance in general, because it would need an infinite or infeasible amount of tests executed on the IUT. Testing can be regarded also as a means of checking non-conformance, because a single test can reveal it. Thus, a test purpose in addition to the model is needed to increase the testers chance to find the potential non-conforming behaviours. The test purpose can be give in the form of coverage criteria or test requirement.

A coverage criteria helps to distribute the parts of the IUT model covered by test cases more evenly. The typical structural coverage criteria cover all edges, pairs of edges or border conditions of the constraints involved. Although the coverage seems like an even distribution, it is usual that covering some elements of the coverage is much more difficult than other elements and the IUT needs guidance to traverse the elements with minimal exploration effort avoiding unnecessary parts of the model [66].

Another approach is an additional specification of test requirements. A standard can be given as a general model of operation and lot of specific requirements that must be met, e.g. "every packet sent should be resent if an acknowledgement is not received". One way of formalizing the test requirements is to build an additional partial model, so called scenario model that is composed with the specification to reduce non-determinism and possible behaviours into a manageable set [40, 41]. Another approach to formalize the requirement is to express it by a reachability condition over the elements of the specification.

We model the purpose of the test by a set of *traps* associated with the edges of the specification automaton. The term is also used with a different meaning in [41] for the states associated to test verdicts. The use of traps is similar to [65], but we lift the meaning to I/O-EFSM models for being able to handle conditions on state variables.

Definition 4.2. A *trap* tr is a pair (e_{tr}, C_{tr}) where $e_{tr} \in E$ is an edge and constraint C_{tr} is a formula of $\mathcal{T}_\Sigma(X_i \cup X_s \cup X_a)$. The set of all traps is denoted by Tr. A trap is *covered* if a transition $(l, \alpha) \xrightarrow{(i(\overline{v}_i), o(\overline{v}_o))} (l', \alpha')$ is taken over edge e_{tr} and $\alpha\,[\overline{x}_i \mapsto \overline{v}_i] \models C_{tr}$.

The trap predicate C_{tr} expresses a pre-condition that must be satisfied before the edge e_{tr} is used for a transition. There can be many traps associated to the same edge. Under the *test goal* we mean a single trap and the *test purpose* the set of all traps given.

Defining traps in this way allows one to express the goals of a test in connection to the edges. The edges are the primary structural elements of the model because the communication interactions are related to edges. The edge associated to a trap determines the source location where the edge is taken, but the state of the system depends also on variables. The trap predicate C_{tr} is used to specify the valuation of variables and input parameters before the edge is taken. For instance the guard of the edge

e may constrain that the input parameter j must be non-negative ($j \geq 0$). One may want to test that the IUT behaves correctly when the border condition is satisfied and add a trap $(e, j = 0)$. The purpose of border condition coverage can be achieved by having a trap for every border condition. Full transition coverage can be formalized by associating a trap with condition *true* to every edge.

Some usual test purposes may be difficult to formalise by the simple set of traps alone, e.g. "pass a transition at least 3 times" or "take the transition A followed by the transition B". An additional auxiliary counter of history variables can be used to record some aspects of the history of test execution to model such properties. Introducing auxiliary variables does not change what runs are feasible in the model. This concept of traps together with auxiliary variables is powerful enough to test all the reachability properties.

The model should specify the correct behaviours only. The safety properties and general assertions in the form "free coffee is never served" or "it never happens that the lift is moving and the doors are open" is usually not a goal of testing, but functional verification of a model. If such a behaviour is present in the model, the model can be considered wrong in respect to the safety requirements. If such a behaviour is present in the IUT, but not in the model, a test set with a good coverage will hopefully reveal a non-conforming behaviour. For example in case of testing a lift controller an output "close doors" is not received where the model expects this.

4.4 TEST GENERATION USING MODEL-CHECKING

Model-checking is used for generating tests in different contexts, by extracting test cases from the traces of the model-checking. This is best suited to deterministic models, because it allows to generate a finite test case and no expensive computations are needed during the test run. We do not use the following process directly, but use it to demonstrate how the model-checking could be used in our setting. To formalize the test purpose the model of the IUT is augmented by trap variables as done also in [66].

A tester is generated based on a model $M = (L, l_0, X, D, I, O, E)$ of the IUT and the set of traps Tr representing the test purpose. An augmented model $M^{Tr} = (L, l_0, X \cup X^{Tr}, D, I, O, E^{Tr})$ is generated such that

- X^{Tr} consists of a Boolean variable x_{tr} for every trap $tr \in Tr$. The variables are auxiliary $X^{Tr} \subseteq X_a^{Tr}$.

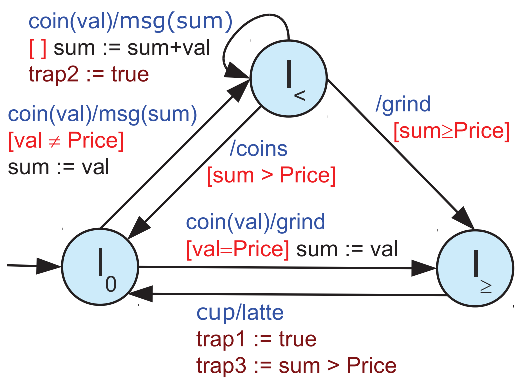


Figure 4.4: Automata augmented with trap variables

- E^{Tr} is generated from E by augmenting the update U_{tr} of the edge $e_{tr} = (l_{tr}, i_{tr}, o_{tr}, g_{tr}, U_{tr}, l'_{tr}) \in E$ associated to every trap $tr = (e_{tr}, C_{tr})$ by adding $x_{tr} := C_{tr} \vee x_{tr}$ to the beginning of the update sequence.
- we assume that all the augmented variables x_{tr} are initialized to false.

The resulting augmented model evaluates a trap variable to true whenever a transition is taken that covers the trap. The test generation can be viewed as finding a feasible run in an augmented model M^{Tr} that evaluates all the Boolean variables $x_t \in X_T$ to true.

Example 4.2. The tree requirements that the vending machine in Figure 3.2 is able to (1) brew latte, (2) accept more than one coin and (3) brew latte when the amount of coins inserted exceeds price. These requirements can be formalized as traps $(l_{\geq} \to l_0, true)$, $(l_{<} \to l_{<}, true)$ and $(l_{\geq} \to l_0, sum > Price)$. The resulting augmented automata is depicted in Figure 4.4.

4.5 SUMMARY

In this chapter we presented testing a tester as an environment that communicates to the IUT purposefully following a test purpose. The test oracle simulates the same communication on the model and gives verdicts if the test purpose is satisfied or the IUT is not conforming to the model. Testing can not be used for confirming that the IUT conforms to the model in general, because it would need an infinite amount of testing. It can assure that non-conforming behaviour was not detected while the behaviour of the IUT covered the test purpose. A test purpose is used to guide the IUT to the behaviours and corner cases and restrict the potentially infinite amount of behaviours to those that demonstrate some feature of the system. We use traps associated with the edges of the model to formalize such test purposes. Counter-example traces generated by model-checking can be used to derive tests to check the test purpose. It is possible to generate finite and deterministic test suites for deterministic models, but it is inefficient for non-deterministic systems, because extensive time-bounded modelchecking would be needed during the actual testing. We will propose a method for generating a symbolic test strategy in the following chapter, that can be used as efficient online test planning strategy for efficient guiding of the testing process towards satisfying the test purpose.

5 SYMBOLIC TESTING STRATEGY

The goal of the chapter is to demonstrate how a symbolic test strategy can be found that is a prerequisite for on-the-fly generation of concrete tests to cover the test goals in efficient way. The strategy is symbolic and operates on the sets of states represented by predicates (constraints). The strategy generation involves analysis of the model by backwards symbolic reachability analysis. This avoids expensive analysis during the actual testing time so that the test control decisions made on-the-fly can be based on the local data and the test strategy generated offline. The method is also presented in [42, 64].

We start by defining the meaning of symbolic state representation in Section 5.1. Pre-image calculation is introduced as the basic building block for performing the reachability analysis in Section 5.2. The symbolic representation of reachability relation is introduced in Section 5.3 and in the algorithms of reachability analysis explained in Section 5.4. Section 5.4.2 explains the reachability analysis in the context of test goals represented by traps.

The temporal direction of the notation and expressions follows the order of reasoning and computation in the current chapter. It is opposite to the direction of the edges and the run of the automaton, because the analysis is done backwards. So the current location l is the target and l' is the source of an edge $e = l' \longrightarrow l$, C' is the pre-image of C and so on.

5.1 SYMBOLIC STATE REPRESENTATION

In Section 3.2.2 we defined a state as a pair (l, α) of a location and assignment to state and auxiliary variables. In the following we will represent symbolic state $S \equiv (l, C)$ by a pair of a location l and constraint C and its interpretation $\mathfrak{I}$ is a set of states (l, α), such that $\alpha \models C$:

$$(l, C)^{\mathfrak{I}} \overset{\text{def}}{=} \{(l, \alpha) | \alpha \models C\}$$

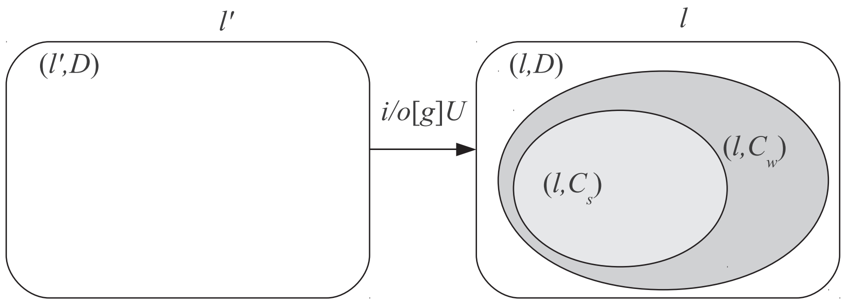


Figure 5.1: Symbolic states

The constraint C is a formula of chosen background theory with free variables from the state and auxiliary variable sets $X_s \cup X_a$. This enables symbolic representation of the data component of the state, the discrete control structure represented by locations is handled explicitly. The Figure 5.1 represents a fragment of an automaton with two locations l and l'. (l, D) represents the largest possible symbolic state associated to the location l, where D is a predicate denoting the domain of the variables. (l, C_s) and (l, C_w) are different symbolic states.

A symbolic state (l_s, C_s) is a *sub-state of a symbolic state* (l_w, C_w), iff $l_s = l_w$ and C_w is *weaker* than C_s, expressed formally as $\models C_s \Rightarrow C_w$. It holds in that case that $(l_s, C_s)^{\mathfrak{I}} \subseteq (l_w, C_w)^{\mathfrak{I}}$. For instance C_w is weaker than C_s in Figure 5.1. This reduces the set inclusion of symbolic states to validity checking. Validity checking can be reduced to satisfiability checking, such that the problem can be solved by an appropriate SMT or SAT solver. C_w is weaker than C_s if and only if $\neg(C_s \implies C_w)$ is not satisfiable. It is equivalent to $(l, C_s)^{\mathfrak{I}} \setminus (l, C_w)^{\mathfrak{I}} = \emptyset$.

We do not assume any normal form for the constraints in representation of the symbolic states. The procedures used do not need to check the equivalence of two constraints and the set inclusion checking is reduced to the satisfiability problem. However, it is important from the practical point of view that the constraints are as concise as possible. Fast heuristic simplification methods are used for that purpose to have close to optimal tradeoff between the complexity of simplification and complexity resulting from the more complex representation of the constraints. More details about the simplification methods used are presented in Section 7.2.

5.2 PRE-IMAGE OF A SYMBOLIC STATE

The set of states from where a state is reachable by one transition is called a *pre-image* of the state. Repeated application of the pre-image finding procedure locates the set of states from where the target state is reachable

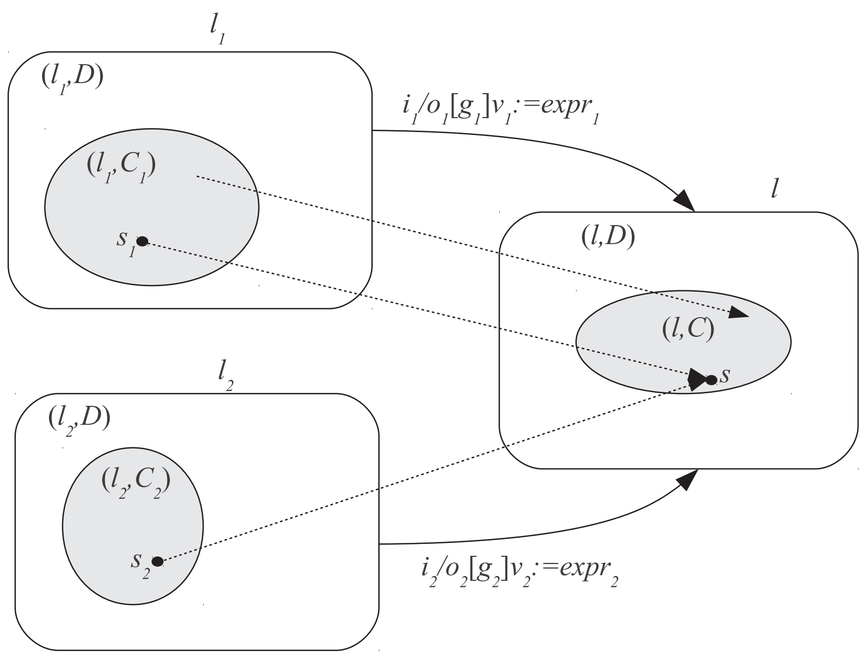


Figure 5.2: Pre-image

in finite number of steps. We use the backward reachability analysis based on pre-image calculation for finding the testing strategy.

Definition 5.1. Let s be a state, S a symbolic state and (S, A, T) an interaction transition system. A pre-image for a state $s \equiv (l, \alpha)$ and symbolic state $S \equiv (l, C)$ is defined as follows:

$$Pre(s) \stackrel{\text{def}}{=} \{s' | \exists a \in A : s' \xrightarrow{a} s \in T\}$$
$$Pre(S) \stackrel{\text{def}}{=} \{s' | \exists a \in A, s \in S^{\mathfrak{I}} : s' \xrightarrow{a} s \in T\}$$

An example of pre-images is shown in Figure 5.2. The pre-image of the state s is the set $\{s_1, s_2\}$. The pre-image of the symbolic state (l, C) is the set $(l_1, C_1)^{\mathfrak{I}} \cup (l_2, C_2)^{\mathfrak{I}}$.

These definitions of pre-image in the semantic domain are not very constructive. We can give the constructive definition of the pre-image predicate on the syntactic level of an I/O-EFSM in terms of the *weakest* pre-condition of the predicate transformer semantics of the guarded command language [21]. An update of an edge of an I/O-EFSM can be regarded as a sequence of assignments and the weakest pre-condition of the update can be calculated by a sequence of substitutions.

Definition 5.2. Let $U \equiv [x_1 := expr_1; \ldots; x_n := expr_n]$ be an update and C a constraint. The weakest pre-condition is defined using the substitution of all free occurrences of the variables x_i by the expression $expr_i$.

$$wp(U,C) \overset{\text{def}}{=} C[expr_n/x_n]\dots[expr_1/x_1]$$

The pre-image involves many locations in general. We define more fine-grained pre-images in respect to edges and locations of the automaton.

Definition 5.3. Let (l,C) be a symbolic state and $e \equiv (l', i(\overline{x}_i), o(\overline{x}_o), g, U, l)$ be an edge of the I/O-EFSM($\mathcal{T}_\Sigma$) where $target(e) = l$. An *edge pre-image* $Pre((l,C),e)$ of a symbolic state (l,C) is a constraint over free variables from $X_s \cup X_a \cup X_i \cup \{iLabel\}$.

$$Pre((l,C),e) \overset{\text{def}}{=} wp(U,C) \wedge g \wedge (iLabel = i) \wedge D$$

The special variable *iLabel* encodes an input label that triggers the edge e when enabled by a guard g. The conjunction with the domain constraint D means that the valuation of the variables should satisfy also the domain restrictions often called the domain invariant for the transition to be enabled.

We will use the following distributivity property later.

Lemma 5.1. *Let (l,C) be a symbolic state such that C is a disjunction of two constraints $C \equiv C_1 \vee C_2$ then it holds for an edge pre-image that*

$$Pre((l,C_1 \vee C_2),e) \equiv Pre((l,C_1),e) \vee Pre((l,C_2),e)$$

Proof. The proof reduces to the distributivity of substitution over logical connectives, because the substitution is done on the level of free variables. Let $e \equiv (l', i(\overline{x}_i), o(\overline{x}_o), g, [x_1 := expr_1; \dots; x_n := expr_n], l)$ in the following:

$$\begin{aligned}
& Pre((l,C_1 \vee C_2),e) \\
\equiv_{Def\ 5.3}\ & wp([x_1 := expr_1; \dots; x_n := expr_n], C_1 \vee C_2) \wedge \\
& g \wedge (iLabel = i) \wedge D \\
\equiv_{Def\ 5.2}\ & (C_1 \vee C_2)[expr_n/x_n]\dots[expr_1/x_1] \wedge g \wedge (iLabel = i) \wedge D \\
\equiv\ & (C_1[expr_n/x_n]\dots[expr_1/x_1] \vee C_2[expr_n/x_n]\dots[expr_1/x_1]) \wedge \\
& g \wedge (iLabel = i) \wedge D \\
\equiv_{Def\ 5.2, 5.3}\ & Pre((l,C_1),e) \vee Pre((l,C_2),e)
\end{aligned}$$

□

5.3 SYMBOLIC REPRESENTATION OF REACHABILITY

We can say that a state s is *reachable* from another state s' if there is a run $\langle s', \ldots, s\rangle$. A symbolic state S is reachable from the symbolic state S' if there is some state $s \in S$ for every state $s' \in S'$ such that s is reachable from s'.

We express the reachability of a symbolic state by the set of constraints and distance measures. The constraints are chosen so that they can be used both for checking reachability and encoding the runs. The result of the reachability analysis of a symbolic state S gives a set of constraints and distance measures for every location and edge of the automaton, namely:

- a *weakest reachability constraint* $C^{+}_{l\to S} \in \mathcal{T}_{\Sigma}(X_s)$ with associated *length* $\mathcal{L}^{+}_{l\to S} \in \mathbb{N}$ for every location l. $C^{+}_{l\to S}$ represents a symbolic state $(l, C^{+}_{l\to S})$ for which there is a run to some state in S with length no more than $\mathcal{L}^{+}_{l\to S}$.
- a *shortest run reachability constraint* $C^{0}_{l\to S} \in \mathcal{T}_{\Sigma}(X_s)$ with associated *length* $\mathcal{L}^{0}_{l\to S} \in \mathbb{N}$ for every location l. $C^{0}_{l\to S}$ represents a symbolic state $(l, C^{0}_{l\to S})$ for which there is a run with length $\mathcal{L}^{0}_{l\to S}$ to some state in S and there is no run from (l, α') to S that is shorter than $\mathcal{L}^{0}_{l\to S}$.
- a *guarding constraint* $C^{g}_{e\to S} \in \mathcal{T}_{\Sigma}(X_s \cup X_i \cup \{iLabel\})$ for every edge e. $C^{g}_{e\to S}$ represents a symbolic state and input for which the edge e is the initial transition of a shortest path to state S.

The constraints are used for guiding the test data generation process and it is described in detail in Section 6.3.

The constraints are related to each other. It is shown by Lemma 5.2 that $C^{+}_{l\to S}$ is weaker than $C^{0}_{l\to S}$ and by Lemma 5.3 that if any symbolic state S is reachable from location l with $C^{+}_{l\to S}$ being true then a guarding constraint $C^{g}_{e\to S}$ of one of the edges leaving from l is satisfiable identifying the first edge on the path from the current state to S.

5.4 REACHABILITY ANALYSIS

The goal of the reachability analysis of a symbolic state is to construct a set of constraints for each location of the model as described in Section 5.3. The procedure has similarities to the winning strategy generation in the game theory [5, 12, 54] due to the interactive nature of the model. It is also close to the bounded symbolic model-checking [6, 8], but more information is recorded than just reachability.

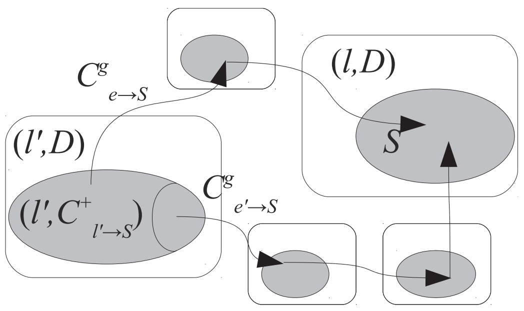


Figure 5.3: Reachability analysis

5.4.1 *General idea of the reachability analysis*

The reachability constraints are constructed by backwards breath-first propagation of the constraints starting from the symbolic state S. The basic idea of the computation is shown in Figure 5.3. The computation starts from the given symbolic goal state S and propagates backwards over all edges leading to the state. This gives the initial guarding constraints for the edges entering to location l and reachability constraints for the neighbouring locations. The reachability constraint on the neighbouring location constitutes a symbolic state. The reachability of the symbolic state is propagated further. The reachability constraints on the same location resulting from the propagation along the different paths are combined together by disjunction (or union of the symbolic states determined by the constraints). The guarding constraints on the edges serve as signs that associate a subset of the reachable symbolic state $(l', C^+_{l'\to S})$ to the edge with shortest run to target symbolic state S.

The abstract version of the reachability constraints calculation is given as Algorithm 5.1 and a more detailed version as Algorithm 5.2.

The algorithm takes three arguments, a target symbolic state S, initial location l_0 and *bound* to the number of iterations. It generates a constraint $C^+_{l\to S}$ for every location and constraints $C_{e\to S}$ and $C^g_{e\to S}$ for every edge of the automaton. Constraints $C_{e\to S}$ are regarded as intermediate results and not returned finally.

The main loop of the algorithm (line 2–17) propagates the constraints by one transition further from the target state S at each iteration. When the loop has iterated for $\mathcal{L}$ times then the constraints express the reachability of S in $\mathcal{L}$ transitions. The loop has three termination conditions:

- The parameter *bound* is an upper limit to the number of iterations of the while loop. It is used for the bounded reachability analysis, when it is important to limit the process and the following termination conditions do not apply.

Algorithm 5.1 Reachability constraint generation

```
REACHABLE($S$, $l_0$, $bound$)
 1: initialize constraints $C^+_{l\to S}, C^+_{e\to S}, C^g_{e\to S}$ for all $l, e$ to $false$
 2: while $\mathcal{L} < bound$ and not $fixpoint$ [and not reachable from $(l_0, \alpha_0)$] do
 3:    $\mathcal{L} \leftarrow \mathcal{L} + 1$
 4:    for all edges $e$ do
 5:       $C_{e\to S} \leftarrow Pre((target(e), C^+_{target(e)\to S}), e)$
 6:       $C^g_{e\to S} \leftarrow C^g_{e\to S} \vee (C_{e\to S} \wedge \neg C^+_{source(e)\to S})$
 7:    end for
 8:    $fixpoint \leftarrow true$
 9:    for all locations $l$ do
10:       $C^{+'}_{l\to S} \leftarrow \bigvee_{e\in out(l)} \exists \overline{x}_i, iLabel : C_{e\to S}$
11:       if $C^{+'}_{l\to S} \neq C^+_{l\to S}$ then
12:          $fixpoint \leftarrow false$
13:          $C^+_{l\to S} \leftarrow C^{+'}_{l\to S}$
14:          $\mathcal{L}^+_{l\to S} \leftarrow \mathcal{L}$
15:       end if
16:    end for
17: end while
18: return all $C^+_{l\to S}, C^g_{e\to S}, \mathcal{L}^+_{l\to S}$
```

- Reaching the smallest fixpoint, when none of the constraints $C^{+}_{l \to S}$ are updated during the execution of the body of the while loop. It is detected by the condition on line 11. Reaching the fixpoint means that the resulting constraints express the maximal set of symbolic states from where the target state S is reachable.

- The *initial state* condition is added if it is enough to check if the goal state S is reachable from the initial state (l_0, α_0). This condition can be used in case the test purpose consists of single goal. The strategy generated this way may not be sufficient for many goals, because the reachability from arbitrary state is important, e.g. from the state were the previous goal was achieved. Reachability from the initial state can be detected by checking if $\alpha_0 \models C^{+}_{l_0 \to S}$ holds.

The body of the while loop is divided into two phases. At first the constraints for edges are updated (lines 4–7) and then the constraints for locations are updated (lines 9–16). The constraints $C_{e \to S}$ are updated (line 5) to edge pre-image of the reachability constraint on the target location of e. The guarding constraint $C^{g}_{e \to S}$ is weakened (line 6) by the states from where the reachability of S was discovered in the current step through edge e. This is denoted by the constraint $(C_{e \to S} \wedge \neg C^{+}_{source(e) \to S})$. The symbolic states of the location of $source(e)$ for which the edge e is the first transition of the shortest run to S are collected in $C^{g}_{e \to S}$ in this way.

The reachability conditions are propagated from edge constraints to location constraints (lines 9–15). A new version of the location constraint $C^{+'}_{l \to S}$ is a disjunction of all the constraints of the outgoing edges where the inputs are hidden by the existential quantification of input variables (line 10). The inputs are hidden because we are free to choose input in the testing process. The reachability of a location depends only on the valuation of the state variables. When the new version of the location constraint is not equivalent to the previous version, the conclusion is made that the process is not converged to the fixpoint and the location constraint is updated (line 12).

5.4.2 *Reachability of traps*

We are interested in reachability of traps in the context of test strategy generation and it is defined through the reachability of the pre-image of the trap. The pre-image of a trap is a symbolic state from where the trap transition can be taken so that the trap condition is satisfied before the transition and domain constraints are not violated in the target location.

Definition 5.4. Let $tr = (e_{tr}, C_{tr})$ be a trap, where $e_{tr} = (l', i(\overline{x}_i), o(\overline{x}_o), g, U, l)$ is the edge the trap is associated with. The pre-image of the trap tr is defined as follows

$$Pre(tr) \overset{\text{def}}{=} \exists \overline{x}_i, iLabel : C_{tr} \wedge Pre((l, D), e_{tr})$$

For a trap tr we define similar reachability constraints $C^0_{l\to tr}$, $C^+_{l\to tr}$, $C^g_{e\to tr}$ and distance measures $\mathcal{L}^0_{l\to tr}$, $\mathcal{L}^+_{l\to tr}$ as follows:

$$\begin{aligned}
C^0_{l\to tr} &\overset{\text{def}}{=} C^0_{l\to Pre(tr)} \\
C^+_{l\to tr} &\overset{\text{def}}{=} C^+_{l\to Pre(tr)} \\
C^g_{e\to tr} &\overset{\text{def}}{=} C^g_{e\to Pre(tr)} \\
\mathcal{L}^0_{l\to tr} &\overset{\text{def}}{=} \mathcal{L}^0_{l\to Pre(tr)} + 1 \\
\mathcal{L}^+_{l\to tr} &\overset{\text{def}}{=} \mathcal{L}^+_{l\to Pre(tr)} + 1
\end{aligned}$$

Example 5.1. We demonstrate the calculation of the test strategy on the example of the vending machine controller introduced in Chapter 2. The intermediate and final results of the strategy calculation are given in Table 5.1. Table 5.2a shows how the constraints $C^+_{l\to tr}$ associated to locations and the Table 5.2b shows how the constraints $C^g_{e\to tr}$ associated to edges are generated for $trap3$. Table 5.2c shows the resulting distance measures for all traps. The model of the vending machine controller is repeated here for convenience.

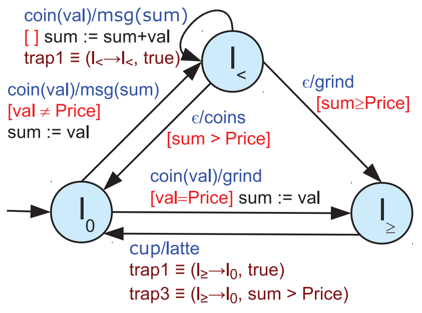
Row 0 of the table has the default values $false$ or the pre-image of the trap as explained in Section 5.4.2. Rows 1–4 correspond to the iteration of the main loop of the reachability algorithm (Algorithm 5.1). The last row (*) is the result of the fixed point calculation that is returned as the result of the algorithm. Only the changes are shown in the tables, an empty space denotes that nothing was changed for the particular constraint in that step. The results are generated by our prototype implementation, but further simplified by hand for an equivalent, but more concise, representation. Particularly the components of the domain constraint D are removed, because the constraints should be always used in conjunction to D. We use a shorthand notation for denoting edges: $e_{0,<}$ denotes the edge $l_0 \longrightarrow l_<$ and $e_{<,<}$ denotes the self-loop at location $l_<$.

The constraints in the Table 5.2a correspond to the constraints of the symbolic states from where the trap is reachable in $row\# + 1$ transitions.

Table 5.1: Symbolic test strategy

#	$C^{+}_{l_{\geq}\to trap3}$	$C^{+}_{l_{<}\to trap3}$	$C^{+}_{l_{0}\to trap3}$
0	$sum > 20$	$false$	$false$
1		$sum > 20$	
2		$sum \geq 1$	
3		$true$	$true$
4	$true$		
*	$true$	$true$	$true$

(a) Constraints and distances associated to locations



#	$C^{g}_{e_{\geq,0}\to trap3}$	$C^{g}_{e_{<,\geq}\to trap3}$	$C^{g}_{e_{<,<}\to trap3}$	$C^{g}_{e_{0,<}\to trap3}$
0	$iLabel = cup$ $\wedge sum > 20$	$false$	$false$	$false$
1		$iLabel = \epsilon \wedge$ $sum > 20$		
2			$iLabel = coin \wedge$ $sum + val > 20 \wedge sum \leq 20$	
3			$iLabel = coin \wedge$ $((sum + val > 20 \wedge sum \leq 20) \vee$ $(sum + val > 0 \wedge sum < 1))$	$iLabel =$ $coin \wedge val > 0$
4	$iLabel = cup$			
*	$iLabel = cup$	$iLabel = \epsilon \wedge$ $sum > 20$	$iLabel = coin \wedge$ $((sum + val > 20 \wedge sum \leq 20) \vee$ $(sum + val > 0 \wedge sum < 1))$	$iLabel =$ $coin \wedge val > 0$

(b) Constraints associated to edges

tr	$\mathcal{L}^{0}_{l_{\geq}\to tr}$	$\mathcal{L}^{0}_{l_{<}\to tr}$	$\mathcal{L}^{0}_{l_{0}\to tr}$	$\mathcal{L}^{+}_{l_{\geq}\to tr}$	$\mathcal{L}^{+}_{l_{<}\to tr}$	$\mathcal{L}^{+}_{l_{0}\to tr}$
$trap1$	1	2	2	1	4	4
$trap2$	3	1	2	3	1	2
$trap3$	1	2	4	5	4	4

(c) Distance measures for all traps

E.g. $sum > 20$ for $C^{+}_{l_< \to trap3}$ means that $trap3$ is reachable from the symbolic state $(l_<, sum > 20)$ with two transitions. The guiding constraints in Table 5.2b give more concrete strategy to drive the IUT to the trap. The corresponding constraint $iLabel = \epsilon \wedge sum > 20$ can be used to conclude that the right thing to do in such state is to provide input ϵ, that means to avoid any input, particularly inserting more coins. The constraints for the edges $e_{0,\geq}$ and $e_{<,0}$ remain $false$, because there is no state from where these edges would be preferable to other alternative edges for reaching $trap3$.

The distances in the Table 5.2c show both the minimal $\mathcal{L}^{0}_{l \to tr}$ and the maximal $\mathcal{L}^{+}_{l \to tr}$ distance to trap from that location. E.g. the optimal run from the initial location to the $trap3$ has length 4. The maximal (minimal) distance is equal to $row\# + 1$ of the last (first) update of the corresponding constraint.

5.4.3 *Description of the reachability analysis algorithm*

The more detailed algorithm introduced as Algorithm 5.2 in this section refines the basic idea presented in Section 5.4.1 in many directions to make it computationally feasible. The most notable refinements are:

- selection of the constraints that need to be propagated
- propagation only the relevant parts of the constraints
- simplification of the constraints to have a more concise representation
- reduction of equivalence checking to satisfiability checking

The constraints C^{+} expressing reachability are weakened in each iteration of the algorithm by the disjunct C^{Δ}, i.e. $C^{+'} = C^{+} \vee C^{\Delta}$. Using the distributivity of *Pre* stated in Lemma 5.1, it suffices to find the pre-image of the disjunct C^{Δ} for finding the pre-image of $C^{+'}$, provided that we have found the pre-image of C^{+} in the previous iterations. So the main idea of the algorithm is to propagate the disjuncts C^{Δ} and it is usually more efficient than propagating the full constraints.

The Algorithm 5.2 refines the more abstract algorithm 5.1. The overall structure is similar consisting of initialization (lines 1–6) and the main loop (lines 7–34) with sub-loops updating the edge and location constraints respectively. Not all the edges and locations are considered at each iteration,

Algorithm 5.2 Reachability constraint generation

```
REACHABLE($S$, $l_0$, $bound$)
 1: initialize all constraints $C^0_{l\to S}, C^+_{l\to S}, C^g_{e\to S}$ for all $l, e$ to $false$
 2: initialize all distances $\mathcal{L}^0_{l\to S}, \mathcal{L}^+_{l\to S}$ for all $l$ to $0$
 3: $(l, C) \leftarrow S$
 4: $C^0_{l\to S} \leftarrow C^+_{l\to S} \leftarrow C^\Delta_{l\to S} \leftarrow C$
 5: $\mathcal{L} \leftarrow 0$
 6: $Q \leftarrow \{l\}$
 7: while $\mathcal{L} \leq bound$ and $Q \neq \emptyset$ [and $\alpha_0 \models C^+_{l_0\to S}$ ]) do
 8:   $\mathcal{L} \leftarrow \mathcal{L} + 1$
 9:   for all edge $e$ do
10:     $C^\Delta_{e\to S} \leftarrow false$
11:   end for
12:   for all (edge $e = (l', i(\overline{x}_i), o(\overline{x}_o), g, U, l)$ such that $target(e) \in Q$ )
      do
13:     $C^\Delta_{e\to S} \leftarrow Pre((l, C^\Delta_{l\to S}), e)$
14:     if $SAT^{\mathcal{T}_\Sigma}(C^\Delta_{e\to S} \wedge \neg C^+_{l'\to S})$ then
15:       $C^g_{e\to S} \leftarrow$ SIMPLIFY($C^g_{e\to S} \vee (C^\Delta_{e\to S} \wedge \neg C^+_{l'\to S})$)
16:       $\mathcal{E}_{l'\to S} \leftarrow \mathcal{E}_{l'\to S} \cup \{e\}$
17:       $Q' \leftarrow Q' \cup \{l'\}$
18:     end if
19:   end for
20:   for all location $l' \in Q'$ do
21:     $C^\Delta_{l'\to S} \leftarrow$ SIMPLIFY($\bigvee_{e\in\mathcal{E}_{l'\to S}} \exists iLabel, \overline{x}_i : C^\Delta_{e\to S}$)
22:     $C^+_{l'\to S} \leftarrow$ SIMPLIFY($C^+_{l'\to S} \vee C^\Delta_{l'\to S}$)
23:     if $C^+_{l'\to S}$ is more compact than $C^\Delta_{l'\to S}$ then
24:       $C^\Delta_{l'\to S} \leftarrow C^+_{l'\to S}$
25:     end if
26:     $\mathcal{L}^+_{l'\to S} \leftarrow \mathcal{L}$
27:     if $C^0_{l'\to S} = false$ then
28:       $C^0_{l'\to S} \leftarrow C^+_{l'\to S}$
29:       $\mathcal{L}^0_{l'\to S} \leftarrow \mathcal{L}^+_{l'\to S}$
30:     end if
31:   end for
32:   $Q \leftarrow Q'$
33:   $Q' \leftarrow \emptyset$
34: end while
35: for all location $l$ and edge $e$ do
36:   simplify and return all constraints $C^0_{l\to S}, C^+_{l\to S}, C^g_{e\to S}$, distance mea-
      sures $\mathcal{L}^0_{l\to S}, \mathcal{L}^+_{l\to S}$ , and promising outgoing edges $\mathcal{E}_{l\to S}$ .
37: end for
```

but only those that that have the pre-condition changed in the previous iteration. The set of locations that have their constraint changed is handled by the queue Q. An empty queue denotes that a fixed point is reached.

The constraints are propagated only for edges that have the target location in Q (line 12). The change $C^{\Delta}_{e\to S}$ of the edge constraint is calculated based on the change $C^{\Delta}_{l\to S}$ of the target location (line 13). The change is propagated further only when it includes some states that are not found reachable in the source location l' of the edge. It is checked using the satisfiability check modulo the background theory $\mathcal{T}_{\Sigma}$ (line 14). If the new reachable states are found, the guarding constraint $C^{g}_{e\to S}$ is updated, the edge is added to the set of *promising outgoing edges* $\mathcal{E}_{l'\to S}$ from l' and l' is added to the queue Q' of the locations with updated constraints.

Procedure Simplify() is used to apply algebraic simplification to the argument formula and yield an equivalent, but more concise and simple formula. Lot of repeating structure occurs usually in the constraints because of the iterative nature of the algorithm, and the simplification helps to reduce the complexity of handling the constraints. No normal form is assumed for the constraints. The constraints $C^{\Delta}_{l'\to S}$ associated with the locations include also existential quantifiers because of the input hiding. It is guaranteed for many background theories (e.g., linear arithmetic) that the quantifiers can be eliminated. The simplification also applies the quantifier elimination procedures where applicable to get a quantifier free formulae. The simplification is the crucial procedure for achieving feasibility of the method. It is discussed more extensively in Section 7.2.

The constraints are only updated for the locations in Q' that have additional state(s) reachable. At first, the change $C^{\Delta}_{l'\to S}$ is found (line 21) and then the main reachability constraint $C^{+}_{l'\to S}$ is updated (line 22). It happens sometimes that the full constraint simplifies to more concise result than the change. The change is replaced by the full constraint in that case (line 24) to use it for further propagation. The shortest run constraints $C^{0}_{l'\to S}$ and distance measures $\mathcal{L}^{0}_{l'\to S}$ are recorded also when the location occurs first in the reachability analysis (lines 27–29). Both are later used in estimation of the distance between the traps (Subsection 6.2.1).

Lemma 5.2. *It holds that $C^{0}_{l\to S} \Rightarrow C^{+}_{l\to S}$. $C^{+}_{l\to S}$ is weaker than $C^{0}_{l\to S}$.*

Proof. $C^{+}_{l\to S}$ is updated once in every iteration of the main while loop and $C^{0}_{l\to S}$ in one iteration of the loop when the location l is considered. There are two possibilities:

$C^{0}_{l\to S} \Leftrightarrow C^{+}_{l\to S}$ when $C^{+}_{l\to S}$ gets a value first time around (line 22), the same value is assigned to $C^{0}_{l\to S}$ (line 28) and this is the only time $C^{+}_{l\to S}$ is updated.

$C^0_{l \to S} \Rightarrow C^+_{l \to S}$ when $C^+_{l \to S}$ is weakened to $C^+_{l' \to S} \vee C^\Delta_{l' \to S}$ (line 22) on the subsequent iterations. □

Lemma 5.3. *If the symbolic state S is reachable from the state (l', α), such that $\alpha \models C^+_{l' \to S}$ then the guarding constraint $C^g_{e \to S}$ of at least one of the edges $e \in out(l)$ is satisfiable $\alpha \models C^g_{e \to S}$.*

Proof. Let us assume that for a state (l', α) that $\alpha \not\models C^+_{l' \to S}$ is true in the beginning of an iteration of the while loop and $\alpha \models C^+_{l' \to S}$ at the end. The argument for the iteration goes as follows:

$$\begin{array}{rl}
\alpha \models C^\Delta_{l' \to S} & \text{on line 22, because } C^+_{l' \to S} \text{ is weakened by } C^\Delta_{l' \to S}; \\
\alpha \models \exists iLabel, \overline{x}_i : C^\Delta_{e \to S} & \text{for some } e \in \mathcal{E}_{l' \to S} \text{on line 21;} \\
\alpha \models C^\Delta_{e \to S} & \text{for some } e \in \mathcal{E}_{l' \to S} \text{on line 21, because } \alpha \text{is a} \\
 & \text{partial assignment (Definitions 3.7, 3.5);} \\
\alpha \models C^g_{e \to S} & \text{for some } e \in \mathcal{E}_{l' \to S} \text{on line 15, because} \\
 & \alpha \models C^\Delta_{e \to S} \text{ and } \alpha \not\models C^+_{l' \to S}.
\end{array}$$

The status of $\alpha \models C^+_{l' \to S}$ and $\alpha \models C^g_{e \to S}$ will not change from that point until the end of the algorithm, because both are only weakened at each iteration. □

Example 5.2. We demonstrate constraint calculation in details for one iteration of the main loop of the algorithm from the state where row #1 of the Example 5.1 has been calculated. The values of the main variables of the algorithm are as follows

$$Q = \{l_<\}, C^0_{l_< \to trap3} = C^+_{l_< \to trap3} = C^\Delta_{l_< \to trap3} = (sum > 20)$$

The constraint for edge $e_{<,<}$ (self-loop) is found by calculating a pre-image (line 13):

$$\begin{array}{rcl}
C^\Delta_{e_{<,<} \to trap3} & \leftarrow & Pre((l, C^\Delta_{l_< \to trap3}), e_{<,<}) \\
C^\Delta_{e_{<,<} \to trap3} & \leftarrow & wp(update(e_{<,<}), C^\Delta_{l_< \to trap3}) \wedge guard(e_{<,<}) \wedge \\
 & & (iLabel = coin) \wedge D \\
C^\Delta_{e_{<,<} \to trap3} & \leftarrow & (sum > 20)[sum + val/sum] \wedge true \wedge (iLabel = coin) \wedge D \\
C^\Delta_{e_{<,<} \to trap3} & \leftarrow & sum + val > 20 \wedge (iLabel = coin) \wedge D
\end{array}$$

After finding the pre-image, we check that it contains some states that are not contained in the constraint of the source location $C^{+}_{l_{<}\rightarrow trap3}$. It is done by checking satisfiability of $C^{\Delta}_{e_{<,<}\rightarrow trap3} \wedge \neg C^{+}_{l_{<}\rightarrow trap3}$ (line 14) and that turns out to be true:

$sum + val > 20 \wedge (iLabel = coin) \wedge D \wedge \neg(sum > 20)$ has a satisfying assignment $< sum \mapsto 5, val \mapsto 20, iLabel \mapsto coin >$ and the first version of guarding constraint for $e_{<,<}$ is found (line 15):

$$C^{\Delta}_{e_{<,<}\rightarrow trap3} \leftarrow sum + val > 20 \wedge (iLabel = coin) \wedge D \wedge sum \leq 20$$

Similar procedure for $e_{0,<}$ results an unsatisfiable constraint. The largest possible value of val allowed by D is 20 and it is in contradiction to the rest of the formula:

$$\begin{aligned} C^{\Delta}_{e_{0,<}\rightarrow trap3} &\leftarrow val > 20 \wedge (iLabel = coin) \wedge D \\ C^{\Delta}_{e_{0,<}\rightarrow trap3} &\leftarrow false \end{aligned}$$

and the guarding constraint $C^{g}_{e_{0,<}\rightarrow trap3}$ is not updated. When updating the location constraint for $l_{<}$ the disjunction of changes of the edges leaving the location is found and inputs hidden. $e_{<,<}$ is the only leaving edge with updated constraint. The result is found by quantifier elimination and simplification (line 21,22). Quantifier elimination is easy in that case, because there is only one value of $iLabel$ that satisfies the formula and three possible values for val allowed by D:

$$\begin{aligned} C^{\Delta}_{l'_{<}\rightarrow trap3} &\leftarrow \exists val, iLabel : sum + val > 20 \wedge sum \leq 20 \wedge \\ &\quad\ (iLabel = coin) \wedge D \\ C^{\Delta}_{l'_{<}\rightarrow trap3} &\leftarrow (sum + 1 > 20 \vee sum + 5 > 20 \vee sum + 20 > 20) \wedge \\ &\quad\ sum \leq 20 \\ C^{\Delta}_{l'_{<}\rightarrow trap3} &\leftarrow sum > 0 \wedge sum \leq 20 \\ C^{+}_{l_{<}\rightarrow trap3} &\leftarrow sum > 20 \vee (sum > 0 \wedge sum \leq 20) \\ C^{+}_{l_{<}\rightarrow trap3} &\leftarrow sum > 0 \end{aligned}$$

The full constraint $C^{+}_{l_{<}\rightarrow trap3}$ (line 23) turns out to be more compact than the change $C^{\Delta}_{l_{<}\rightarrow trap3}$ after simplification and is taken as the basis for further propagation (line 24).

The example demonstrates the power of symbolic representation. Although the set of represented states "grows" and formulas are combined and grow during the computation, the final result may be as compact as the initial data if the algebraic simplification procedures are able to discover the possible ways to simplify the generated formulas.

5.5 SUMMARY

In this chapter we have shown how a symbolic test strategy, that helps to cover the IUT test goals can be be represented and automatically generated. The test strategy is represented by predicates that relate the states to reachable traps and inputs that help to guide the IUT towards the traps. Also the estimates of the distances to traps are found to help to decide the reasonable order of covering the traps. The strategy generation is achieved by backwards reachability analysis through constraint propagation starting from the trap constraints. All the analysis is done on the syntactic level of I/O-EFSM avoiding the use of general semantic definition of pre-image with existential quantification. The pre-images are found through weakest pre-condition calculation by substitution instead. The decision about the need to continue propagation is reduced to satisfiability check. Algebraic simplifications are used during the process for compacting the symbolic representation and eliminating existential quantification rising from input hiding.

The test strategy generation can also be viewed as finding a winning strategy of the game between the tester and the IUT where the covering of a trap is taken as a goal. This kind of game-theoretic view have been successfully applied for model-based testing [5, 54, 12] It is shown in [12] that a clever combination of forward and backward analysis yields a winning strategy with optimal effort. It could also be applied to our case also if we have to find a strategy from the initial state to a trap, because the forward analysis need a set of initial states as a starting point. But we have to guide the IUT to traps from other traps or from arbitrary states because of non-deterministic behaviour and that makes the use of forward analysis complicated. The strategy should also support choosing the right parameters for inputs. The issues of applying the strategy are discussed in the next chapter.

6 EXECUTION OF SYMBOLIC TESTING STRATEGY

Non-determinism in the model, infinite state space and branching prevents pre-computation of a fixed set of tests for a test purpose. Concrete test inputs should be found on-the-fly depending on the current state and behaviour of the IUT. Test strategy discussed in Chapter 5 is used for efficient test planning to satisfy the timing expectations of the tester. It is usually assumed that the tester is fast, to be able to produce the next stimulus faster than it takes to communicate to and respond to the IUT.

Each online planning step consists of selecting a next trap to cover, explained in Section 6.2, selecting input, explained in Section 6.3, simulation of the interaction on the model and decision on the conformance of the observed behaviour, explained in the Section 6.1. The overall test execution process is illustrated in Figure 6.1. The method is concisely presented in [42, 64]. An alternative method for input selection presented in [2] is briefly described in Section 6.4.

6.1 TESTING PROCEDURE IN GENERAL

The goal of the testing procedure is to achieve the highest possible coverage of the traps with minimal effort. The measure of effort is usually the

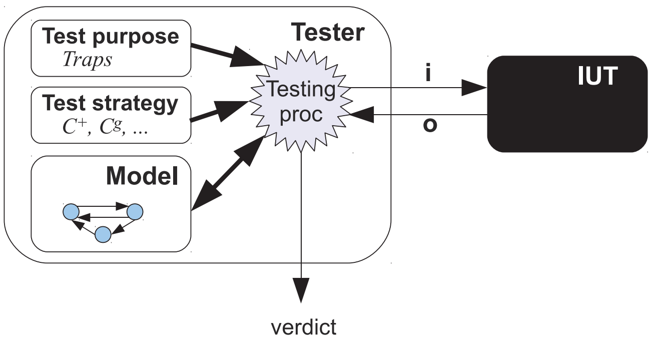


Figure 6.1: Tester components

time spent for achieving coverage, that depends on the number of interactions and time of performing each interaction step, including the time for generating the next stimuli. There is a tradeoff as thorough planning could result in shorter test sequences. The best balance is dependent on the system and can be tuned by different parameters.

The testing procedure interacts with the IUT during the process and is thus carried out on an explicit state and in forward direction differently from the reachability analysis, presented in Chapter 5. The testing procedure TESTING(s_0, $traps$), presented as Algorithm 6.1, starts from an initial state s_0 and a set of traps that specify the test goal. One step of test execution consist of several stages:

1. selecting the trap to be covered next (line 3);
2. selecting the input that leads towards the trap (line 5);
3. sending the input to the IUT and observing the output (line 6);
4. simulating the behaviour on the model (line 8);
5. evaluating the result of the simulation (lines 9–19)

The trap selection stage may not be performed at each step. The goal of the trap selection is to determine what trap to approach next so that all the uncovered traps could be covered with minimal effort. This kind of planning may be expensive and is avoided if possible. The trap selection can be avoided and the process stays in the inner repeat loop (lines 4–20) when the testing process approaches the selected trap according to the test strategy. The loop ends when the trap is covered ($trap \notin traps$ condition on line 20) or the process deviates from the intended behaviour ($e \neq e_{best}$ condition on line 20) because of non-determinism. The deviation can lead the IUT to the state from where the estimation of the distance to the planned trap increases and thus the trap selection is initiated again. The details of trap selection are described in Subsection 6.2.

It is possible that the trap selection procedure returns no trap. That result is denoted by $\emptyset$. This can happen when none of the uncovered traps are reachable or in the case when only the bounded symbolic test strategy analysis have been carried out and none of the traps are "visible" from current state within the bound. The tester switches to “walk around” mode without a certain goal, in hope to reach a state where some trap gets “visible”. This situation is discussed in the section 6.3.

The next stage after the trap selection is the input generation (line 5) that leads the IUT towards the trap. The input with its parameters are

Algorithm 6.1 online test procedure

```
TESTING($s_0$, $traps$)
 1: $s \leftarrow s_0$
 2: while $traps \neq \emptyset$ do
 3:    $trap \leftarrow$ SELECT-TRAP($traps$, $s$)
 4:    repeat
 5:       $(input, e_{best}) \leftarrow$ SELECT-INPUT($trap$, $s$)
 6:       $output \leftarrow$ COMMUNICATE-IUT($input$)
 7:       $log \leftarrow log \cup (input, output)$
 8:       $(e, s') \leftarrow$ SIMULATE($s$, $input$, $output$)
 9:       if $s' = \emptyset$ then
10:          Test-Failed, $log$
11:       end if
12:       $(l, \alpha) \leftarrow s$
13:       $s \leftarrow s'$
14:       for all $tr \in traps$ do
15:          $(e_{tr}, C_{tr}) \leftarrow tr$
16:          if $(e = e_{tr}$ and $(\alpha \cup input) \vDash C_{tr})$ then
17:             $traps \leftarrow traps \setminus \{tr\}$
18:          end if
19:       end for
20:    until $e \neq e_{best}$ or $trap \notin traps$
21: end while
22: Test-Passed
```

generated by solving the relevant reachability constraints. In addition to the input an edge is returned that is the initial transition of the shortest run to the trap from the current state. The generated input is guaranteed to enable the edge, but some other rival edges may also be enabled by the given input. The details of the input generation procedure are provided in the Section 6.3.

The actual test interaction stage is performed through the COMMUNICATE-IUT(*input*) procedure (line 6). This involves all the issues of interfacing to the IUT that are usually solved by the special test adapter. It also involves the implementation of ϵ actions, which are associated with system-dependent timeout. It means that when the goal is to communicate an ϵ input, it may be assumed that it is "sent" after the timeout has expired and when no actual output is received during the timeout after the input has sent, then an ϵ output is assumed to be "received".

The conformance of the observed behaviour of the IUT is checked by simulating the behaviour on the model. The simulation results an unique state because of the output-observability assumption on the model or an empty result if the output is non-conforming. The testing is stopped and non-conformance notification with a test log returned (line 9–11) in that case. The simulation procedure is basically the simulation of all possible transitions from the state s with generated *input* and checking which one conforms to the observed output. The resulting state s' and edge e that the transition corresponds to, are returned as the result.

Finally, the check if any of the traps was covered with the last transition is done (lines 14–19). Covering of the trap is checked (line 13) by comparing if the transition taken corresponds to the edge associated with the trap and checking if the trap condition was satisfied before the last transition. The satisfiability check checks if the assignments α and *input* of the state and input variables satisfy the trap condition C_{trap}. The covered traps are removed from the set of uncovered traps and new trap selection procedure is initiated.

6.2 TRAP SELECTION

The online planning during test execution consists of repetitive selection of a trap to be reached next and thereafter guiding the IUT towards the trap until all the traps are covered. The overall goal is to minimize the time and number of interactions with the IUT for covering all the traps. The trap selection procedure SELECT-TRAP(*uncovered_traps*, *s*) (used in Algorithm 6.1, line 2) should return a trap that needs to be approached next. Generally, to determine the order of traversing traps may involve complex planning. The task of finding an optimal ordering is unsolvable for non-deterministic models and would involve exponentially more symbolic test strategy analysis steps than for deterministic models in general case, thus making it infeasible. We propose an heuristic greedy approach SELECT-TRAP-K-GREEDY(*uncovered_traps*,*state*,*k*) where the tradeoff between the complexity of the analysis and optimality is tuned by the parameter k. The approach tries to minimize the number of test interactions for covering next k traps. The closest trap is selected in case of $k = 1$ and a greedy shortest ordering is chosen if k is larger. The distance between the traps is estimated based on the distances given by the symbolic test strategy. The overall idea of the approach presented as Algorithm 6.2 is that a sequence of traps is constructed starting with each of the uncovered traps and $k - 1$ of the remaining traps are appended in greedy (closest first) manner. The initial trap of the shortest overall distance is chosen as the next trap.

Although the preferred sequence of traps is found, no final decision is done for the whole sequence. Only the next trap is taken and returned as the short-term goal. The trap planning process is repeated after the trap is covered and it may turn out that some other trap will be chosen as the next goal, because the analysis will be carried out based on the actual state information instead of the estimation of reachability and distance between the traps.

Example 6.1. A model depicted in Figure 6.2 with traps $(e_2, true)$, $(e_3, true)$ and l_1 as the current location. The outcome depends on the value of k. In case $k = 1$ the trap $(e_3, true)$ is chosen, because it is closer. In case $k > 1$ the trap $(e_2, true)$ is chosen, because it results in a shorter overall test with length (number of interactions) 3, comparing to the alternative $(e_3, true)$ with overall test length 4. There are no state variables and guards that interfere to the control flow in the example.

The procedure SELECT-TRAP-K-GREEDY(*uncovered_traps*,*state*,*k*) presented as Algorithm 6.2 returns the preferred trap *next_trap* to be covered next, given the set *uncovered_traps*, current *state*, and parameter k. The procedure consists of two nested loops. The outer loop (lines 3–17) consider all the members of the *uncovered_traps* as the best trap. The inner loop (lines 7–12) adds up to $k - 1$ traps to the sequence in greedy manner.

The outer loop considers only those traps in the *uncovered_traps* that are found reachable from the current state (l, α). It is checked if the reachability constraint $C^{+}_{l \to trap}$ is satisfiable in the current assignment α of the state variables (line 3). The satisfiability checking is actually a pure eval-

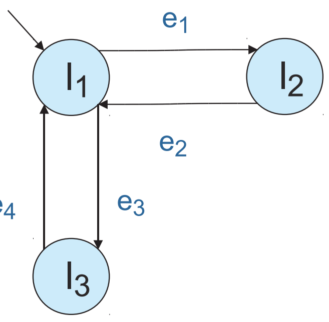


Figure 6.2: Trap ordering

Algorithm 6.2 Trap selection

```
SELECT-TRAP-k-GREEDY($uncovered\_traps, state, k$)
1:  $(l, \alpha) \leftarrow state$
2:  min $\leftarrow \infty$
3:  for all $trap \in uncovered\_traps$ and $\alpha \models C^{+}_{l \rightarrow trap}$ do
4:      $traps \leftarrow uncovered\_traps \setminus \{trap\}$
5:      $loc \leftarrow target(trap)$
6:      $len \leftarrow distance(l, trap)$
7:      for $j \leftarrow 2$ to $min(k, |traps|)$ do
8:          $tr \leftarrow closest\_trap(loc, traps)$
9:          $len \leftarrow len + distance(loc, tr)$
10:         $loc \leftarrow target(tr)$
11:         $traps \leftarrow traps \setminus \{tr\}$
12:     end for
13:     if $len < min$ then
14:         $min \leftarrow len$
15:         $next\_trap \leftarrow trap$
16:     end if
17: end for
18: $next\_trap$
```

uation in that case, because there are no free variables left in $C^{+}_{l \to trap}$ after the assignment is substituted. In the case where there are no traps found to be reachable an empty value $\emptyset$ is returned.

Next, the variables *traps* (set of uncovered traps except the one chosen for the first trap), *loc* (target location of the first trap associated transition), and *len* (length of the test from the current location to the first trap) are initialized (line 4–6) for the inner loop.

6.2.1 *Distance estimation*

The *distance* between a location and trap given by the function $distance(l, trap)$ is an estimate based on the values $\mathcal{L}^{+}_{l \to trap}$ and $\mathcal{L}^{0}_{l \to trap}$. The values denote the length of the longest and shortest run from the location l to the trap found during the symbolic test strategy analysis. $\mathcal{L}^{0}_{l \to trap}$ can be used as the distance if $C^{0}_{l \to trap}$ is satisfiable in the current state. Otherwise we know that there is a run leading to the trap no longer than $\mathcal{L}^{+}_{l \to trap}$ whenever $C^{+}_{l \to trap}$ is satisfiable in the current state, but the actual length of the run is not known. The simplest strategy for estimating the distance is to use the arithmetic mean of $\mathcal{L}^{+}_{l \to trap}$ and $\mathcal{L}^{0}_{l \to trap}$, but it is possible to use also only maximal value or some other more complex function. It would be possible to keep the information of the exact length in the symbolic analysis, but it would make the constraints much more complex and thus more difficult to handle.

It is possible that there is more than one trap with a different data condition associated with the same edge. It is not possible to always foresee if many traps can be covered together, it is possible when the condition of one of the traps is weaker than another. We can say that (l, C_s) covers (l, C_w) iff $C_s \implies C_w$. In that case the distance from the stronger trap to weaker trap is 0, because the weaker trap is always covered at the same time with the stronger.

The distance expresses the number of transitions (and interactions) that must be taken to reach a trap. Execution of different transitions may have very different costs in real systems, e.g., reset may be very complicated, expensive or time-consuming. It is possible to add weights to the edges of the model. It would need some modification of the symbolic analysis where the breath of the breath first traversal is based on weights and also summing up the weights in the distance calculation. The addition of weights does not change anything crucial in the whole method and is omitted in the current work for the sake of clarity.

The inner loop (lines 7–12) implements the repetitive greedy choice of next trap in the sequence based on the distance discussed above. The function $closest\_trap(loc, traps)$ is a simple minimization procedure for finding the closest trap to location loc amongst the traps not in the sequence. Finding the shortest sequence instead of the greedy estimate would require solving the NP-complete Asymmetric Travelling Salesman Problem (ATSP). There is no sense to put so much effort into this problem, because the distances are estimates anyway and there is no guarantee that the next trap is reachable from the actual target state of the covered trap although it is reachable from the target location. For instance the distance to the next trap tr from the target location l is no more than $\mathcal{L}^{+}_{l \to tr}$ only if the actual target state belongs to the symbolic state $(l, C^{+}_{l \to tr})$, but it may not be the case.

The current initial trap is finally recorded (line 13–16) if prefixes the shortest overall trap ordering.

6.3 INPUT SELECTION

Every testing step involves a decision on what input to send to the IUT to drive it towards the test goal. The decision is made based on the current state and the trap chosen to be covered next. The result of the procedure is the input that is sent to the IUT and an edge of the model the input is related to. This is the stage where the actual data generation or selection is done.

The returned $input$ is an assignment to the input variables. The assignment is relative to the input port that is determined by $iLabel$ and to the formal parameters associated with the input port determined by the value of the $iLabel$.

The input generation presented as Algorithm 6.3 has three different methods for handling different situations. The main method is able to generate a deterministic test step (line 4–7) with single allowed reaction from the IUT or a non-deterministic step (line 8–11) in which case the IUT can behave differently than planned, i.e. the IUT can react to the input in several ways and it is not controllable by the tester. The third alternative is considered (line 13) for the case when the goal trap is undefined.

The input generation for deterministic and non-deterministic steps are similar except for the constraints used. The input generation involves checking if a suitable input exists for any of the promising edges in $\mathcal{E}_{l \to trap}$ leaving the location l. The promising edge $\mathcal{E}_{l \to trap}$ is an edge which is a prefix of the shortest run to the trap $trap$ for some state of l. The existence

Algorithm 6.3 Input selection algorithm

```
SELECT-INPUT(trap, state)
 1: if $trap$ is defined then
 2:     $(l, \alpha) \leftarrow state$
 3:     $input \leftarrow \emptyset$
 4:     while $input = \emptyset$ do
 5:         select $e$ in $\mathcal{E}_{l \rightarrow trap}$
 6:         $input \leftarrow$ SOLVE-CONSTR$(\alpha, C^{g}_{e \rightarrow trap} \wedge \bigwedge_{r \in rivals(e)} \neg guard(r))$
 7:     end while
 8:     while $input = \emptyset$ do
 9:         select $e$ in $\mathcal{E}_{l \rightarrow trap}$
10:         $input \leftarrow$ SOLVE-CONSTR$(\alpha, C^{g}_{e \rightarrow trap})$
11:     end while
12: else
13:     $(input, e) \leftarrow$ ALTERNATIVE-SELECT-INPUT$(state)$
14: end if
15: $(input, e)$
```

of an input is found by solving a corresponding constraint in the current assignment α of the state variables. The SOLVE-CONSTR(α, C) procedure substitutes the state variables occurring free in C with their values in α and solves the resulting constraint. If there exists a satisfying assignment to the input variables occurring free in the substituted constraint then the solution is returned as an input. The constraint may have (infinitely) many solutions. Any of the solutions is equally good for reaching the trap. Any SMT solver that supports the chosen background theory and is able to return a satisfying model can be used as a solver to find an input. It is possible to define other data-dependent coverage criteria in addition to control structure based criteria expressed by traps, e.g. boundary criteria [45]. A more specific constraint solving strategy should be used in that case to get the boundary or corner values as input. It is also advisable to use *anti-ant* strategy [52] to avoid the same solution if possible when solving the same constraint in the same state repeatedly.

The solution to deterministic input generation constraint $C^{g}_{e \rightarrow trap} \wedge \bigwedge_{r \in rivals(e)} \neg guard(r)$ is not guaranteed to exist and it may be the case that there is no deterministic run to cover the trap. The deterministic constraint specifies an input that drives the IUT towards the trap and does not enable any rival transition (other transitions leaving from the same location and having non-disjoint guard). Then, the condition for a non-deterministic path $C^{g}_{e \to trap}$ is used instead and it should be solvable, because the *trap* selection procedure involved a test of $\alpha \models C^{+}_{l \to trap}$ and thus the guarding constraint $C^{g}_{e \to trap}$ for at least one $e \in \mathcal{E}_{l \to trap}$ must be satisfiable.

There may be also situations where none of the traps are set as a goal and an alternative strategy for input generation must be used (line 13), because none of the guarding constraints of the edges are satisfiable. It may happen because of several reasons. One possibility is that none of the uncovered traps are reachable from the current state and there is actually nothing that can be done with any strategy about this situation. This may happen in non-connected models where reset is not explicitly specified. But it is possible and quite usual for more complex models that only the bounded reachability analysis is done on the model. It does not mean in that case that the traps are unreachable, they are just "out of sight". The alternative strategy should suggest some input that helps to "walk the trap" and there is non-zero probability that the test path gets close enough to some trap so that the generated bounded test strategy could take over and direct the process to the trap.

The simplest alternative is a *random walk* strategy. It suffices to generate an input that satisfies any of the guards of the edges that depart from the current location. The anti-ant strategy [52] remembers the choices made and chooses a different edge or input when doing a repeated choice in the same location or state. Also heuristic strategies based on evolutionary algorithms have been studied in [43, 19]. One quite promising idea complementing the bounded reachability analysis is described in Section 6.4.

Example 6.2. We continue with the example of the vending machine introduced in Chapter 2. We show the details behind the test instance presented in Table 2.1.

The test generation starts with a goal trap selection. The trap is selected from the set of reachable traps. Since all the traps are reachable (reachability constraints *true* in Table 5.2a) they are considered in the selection. The selection is done by iterated greedy selection criteria that is based on the distances between the traps (Algorithm 6.2). The results of the 3 different variants compared are provided in the Table 6.1. The distances are taken from the test strategy presented in Table 5.2c, except the distance from $trap3$ to $trap1$ which is 0 because $trap1$ is always covered by $trap3$. The first distance is from the initial location l_0. $trap2$ is selected as the first goal to be covered.

Table 6.1: Trap selection

1. trap	*distance*	closest	distance	closest	distance	*len*
$trap1$	(2+4)/2	$trap2$	2	$trap3$	(2+4)/2	8
$trap2$	2	$trap3$	(2+4)/2	$trap1$	0	5
$trap3$	4	$trap1$	0	$trap2$	3	7

Table 6.2 shows one possible test run. The first goal $trap2$ is covered by the first two lines. The relevant guiding constraints for $trap2$ that are not equal to $false$ in the test strategy are the following:

$$\begin{aligned} C^g_{e_{0,<}\rightarrow trap2} &= (iLabel = coin \wedge val \neq 20) \\ C^g_{e_{<,<}\rightarrow trap2} &= (iLabel = coin) \\ C^g_{e_{\geq,0}\rightarrow trap2} &= (iLabel = cup) \end{aligned}$$

The rest of the lines show how the test is driven to $trap3$. The relevant constraints are taken from the test strategy for $trap3$ (Table 5.1). The table contains the current state with its components (location and variable sum), the selected edge with the associated guarding constraint, the result of solving the constraint, and the output generated by the IUT. The input sent to the IUT is constructed from the solution found and the output received is used for detecting the state of the model for the next step. The first step requires that a coin with value not equal to 20 and 5 is selected. This drives the IUT to the state that is modelled by location $l_<$ and $sum = 5$. The output corresponds to the model and the process continues with the next step.

It is worth noticing that the steps 2 and 3 both take the same edge $e_{<,<}$, but use different constraint as a basis for input generation. As the goal of the step 2 is to cover $trap2$, it does not matter what coin is given. The goal is changed to $trap3$ for step 3, and a more complicated constraint is used to drive the system to the state where $sum > 20$. The component $sum + val > 20$ of the constraint is crucial and the only solution to this in conjunction with the domain constraint in the current state is $< val \mapsto 20 >$.

The guarding constraint for $e_{<,<}$ is not satisfiable any more in case $sum = 26$ and the constraint of $e_{<,\geq}$ is used to conclude that the tester should wait for the reaction giving no input (coin). The IUT has a freedom to choose to return the coins or give latte and it chooses to give the coins

Table 6.2: online input generation for vending machine example

Step	State		Edge	Guiding constraint	Solution		Output
	loc	sum			iLabel	val	
1	l_0	0	$e_{0,<}$	$iLabel = coin \wedge val \neq 20 \wedge D$	$coin$	5	$msg(5)$
2	$l_<$	5	$e_{<,<}$	$iLabel = coin \wedge D$	$coin$	1	$msg(6)$
3	$l_<$	6	$e_{<,<}$	$iLabel = coin \wedge ((sum + val > 20 \wedge sum \leq 20) \vee (sum + val > 0 \wedge sum < 1)) \wedge D$	$coin$	20	$msg(26)$
4	$l_<$	26	$e_{<,\geq}$	$iLabel = \epsilon \wedge sum > 20 \wedge D$	ϵ		$coins$
5	l_0	26	$e_{0,<}$	$iLabel = coin \wedge val > 0 \wedge D$	$coin$	1	$msg(1)$
6	$l_<$	1	$e_{<,<}$	$iLabel = coin \wedge ((sum + val > 20 \wedge sum \leq 20) \vee (sum + val > 0 \wedge sum < 1)) \wedge D$	$coin$	20	$msg(21)$
7	$l_<$	21	$e_{<,\geq}$	$iLabel = \epsilon \wedge sum > 20 \wedge D$	ϵ		$grind$
8	$l_\geq$	21	$e_{\geq,0}$	$iLabel = cup \wedge D$	cup		$latte$

back. The strategy should drive the IUT again from the location l_0. This time a coin with value 1 inserted. It may happen randomly or because of the anti-ant strategy used. The following four steps form an optimal test run to the trap and the non-deterministic choice is done in favour of the test strategy this time. It is possible that the real the IUT does not have non-deterministic behaviour internally, it may be that we just don't know the internal logic and non-determinism is used to abstract it. For instance, one possible decision that the engineers could have made is that the coins are given back when the sum of the coins inserted exceeds price by more than 10%.

6.4 alternative heuristic input selection

It is not feasible to generate the symbolic test strategy using fixed point computations for larger and more complex models. Having only a bounded strategy can leave the online test data generation in the situation where the main method of input generation is not applicable, because none of the traps are "visible" and an alternative method is needed. The easiest approach in this situation is to fall back to random testing with possible anti-ant strategy [52]. Another possibility is to add enough traps so that at least one trap is "visible" from the target location of the covered trap. The traps can be also ordered by dependences such that the order of traps to be covered is pre-fixed. This is closer to scenario-based testing [40, 41]. It is also possible to apply some search-based input generation approach [19, 43]. We propose an heuristic search-based test generation that complements and uses the bounded test strategy. This is joint work with Danel Ahman and we give only an brief overview of the method here, referring to [2] for the details.

The approach of heuristic reactive planning tester (χRPT) is inspired by the paradigm of constraint-based local search [34]. The reachability constraints for all traps evaluate to false in the current state in the situation where test strategy is not applicable. But it is possible to come up with a function that gives a measure of violation of the constraint. For instance, a difference of the values of the left and right hand side in the case of inequality or the sum of violations in the case of conjunction can be used:

$$\begin{aligned} \nu(a \geq b) &\overset{\text{def}}{=} abs(min(0, \nu(a) - \nu(b))) \\ \nu(A \wedge B) &\overset{\text{def}}{=} \nu(A) + \nu(B) \end{aligned}$$

The property of the violation function is that the value is 0 for the valid or satisfiable constraint and positive otherwise. This gives an heuristic hint of how far the system is from satisfying a constraint and if a transition helps to move towards the satisfying state.

For doing a test step some reachability constraints $C^{+}_{l\to tr}$ stronger than the domain constraint D are selected from the closest locations l. Next, the possible transitions from the current state are simulated and violations for all the selected constraints calculated from the simulated states. The input that leads to the state with the smallest violation is selected and sent to the IUT. The actual selection of the transition from the potentially infinite set of possible transitions is done in phases by first selecting the most promising pairs of constraints and edge(s) of the model by satisfiability check and then the input parameters are found by solving the constraints for minimal violation. It is not guaranteed that this approach actually leads to some state with satisfiable reachability constraint, but it seems to work quite efficiently at least for the case studies use in [2]. The selection is backed up by a complex taboo search to avoid the transitions taken in the same state to avoid loops. This can be viewed as an advanced anti-ant approach combined with an objective function assisted search.

6.5 SUMMARY

In this chapter we demonstrate how the symbolic test strategy is used for efficient online test planning. The trap ordering is done based on their distance estimates and inputs are generated by solving the guarding constraints that encode the test strategy. All the modelling including the output observability assumption and strategy generation technique are motivated by the possibility to perform the steps of the online testing efficiently to cover the test purpose expressed as a set of traps by test runs of optimal length.

7

IMPLEMENTATION AND TOOLS

The implementation of tester generation described in Chapters 5 and 6 consists of and uses many tools. The overall workflow on the toolchain is shown in the Figure 7.1. The workflow moves from the left to the right. The upper row describes data given and generated, the middle row describes the main components of the toolchain that implement the proposed method of test generation and the bottom row shows the relationship to the external tools used.

The tester generation consists of two stages. At first the testing strategy is generated by the model explorer as described in Chapter 5. After that a test script is made that implements the methods described in Chapter 6. Also the model of the IUT, test goals and test strategy are converted to data structures of the script suitable for efficient test execution. The test script execution engine interacts with the IUT through the test adapter.

7.1 TESTING STRATEGY GENERATION WITH THE ERPT TOOL

The model explorer is implemented as a tool Extended Reactive Planning Tester (ERPT) in Python. It takes a model of the IUT augmented with a set of test goals and generates a test strategy consisting of all needed constraints and distance measures. The Z3 SMT solver [17] version 2 was used in the implementation for satisfiability checks and simplification over ASCII representation of constraints in SMT-LIB v1 format. A new implementation currently under development is connected directly to the Z3 Application Programming Interface (API) avoiding parsing and a process call overhead for every invocation of satisfiability check and simplification.

Different tuning options of ERPT are presented in Figure 7.2. The options -i and -d allow to tune the termination conditions as explained in Subsection 5.4.1. Different guarding constraints can be generated using an option -g. The method of test execution explained in Chapter 6 assumes default behaviour with input label and parameters included in the

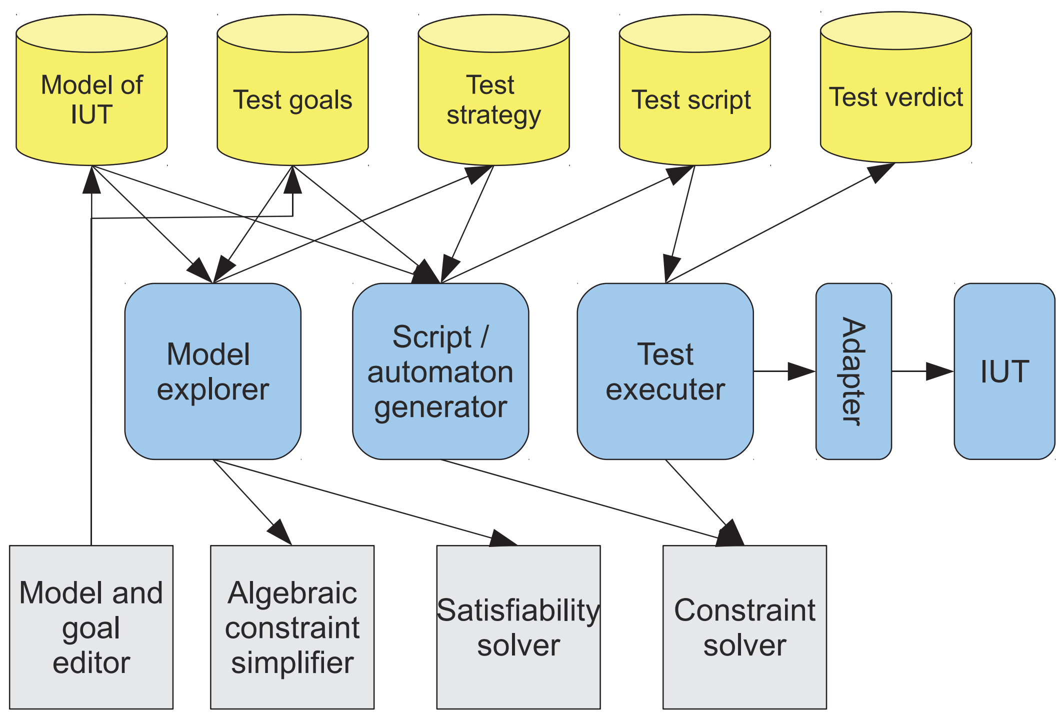


Figure 7.1: The testing workflow with supporting tools

constraints, but it is possible to use different approaches to online test execution with different guarding constraints. Inclusion of inputs can be avoided (-g1) for more efficient strategy generation in the case inputs do not have data parameters. The distances between traps can be estimated more precisely (-g3), doing it at the expense of computational complexity and much more complicated guarding constraints. The options -s and -f allow to tune the amount of simplification operations applied during the computation and to final results. The best options for particular models can be different and require case-by-case tuning.

7.2 SIMPLIFICATION

By simplification we mean algebraic manipulation of the initial formula in such a way that we get an equivalent, but possibly more compact and simpler formula. The simplification involves quantifier elimination, Boolean and background theory-specific simplification. By a simpler formula we mean a formula for which satisfiability check and constraint solving are more efficient.

The ability to simplify the generated constraints to more compact and quantifier-free representation is crucial to the feasibility of the method. The size and complexity of the formulae can grow exponentially during the process and rendering it unmanageable. The algorithms involve re-

```
usage: erpt.py [-h] [-i] [-d DEPTH] [-g #] [-t #] [-s #] [-f #] file
Find ERPT constraints
positional arguments:
file          input file name
optional arguments:
-h, –help     show this help message and exit
-i            finish constraint generation when the initial state
              is covered
-d DEPTH      search depth (default 10)
-g #          type of guarding constraints (default 2):
              0-none, 1-location, 2-inputs, 3-lengths
-s #          guarding constraint simplification during calculation
              (default 0): 0-none, 1-changes, 2-full
-f #          final constraint simplification (default 1):
              0-none, 1-transitions, 2-full
```

Figure 7.2: Options of the ERPT tool

```
(benchmark data :extrafuns ((sum Int)(val Int)(ilabel Int))
:formula(
    (or (and (>= sum 0)(>= sum 20)(not (<= sum 20)))
        (or (exists ((val Int)(ilabel Int))
            (and
              (and (>= sum 0)
                (>= val 1)(<= val 20))
              (= ilabel 1)
              (and (>= (+ sum val) 0)
                (>= (+ sum val) 20)
                (not (<= (+ sum val) 20)))
    ))))))
```

(a) Input to the simplifier

```
(>= sum 1)
```

(b) Output from the simplifier

Figure 7.3: Simplification example

peated satisfiability checking and although modern SMT solver are efficient for large formulae, they benefit from the simplification. It is especially important for online testing, where the inputs must be generated by constraint solving in real-time. Instead of applying simplification to the huge final result, it turns out to be easier and more efficient to simplify the formulae incrementally by the components used in the construction of the formula.

We do not rely on normal forms for checking equivalence. It would cause exponential blow-up of the formulas for the usual normal forms. The equivalence checking of two formulas can be reduced to one-way implication checking in our case that can be reduced to satisfiability checking. So it suffices to have some heuristic simplification that reduces the size of the formulas, but we do not have to put too much effort into simplification to get a pre-defined form. The overall goal of the simplification step is to speed up the whole process and to find a good balance between the effort put into simplification and additional complexity of using less-simplified formulas in constraint construction, satisfiability checking and solving.

We have experimented with using RedLog [22], the simplifier of the Duration Calculus modelchecking engine [32] and simplification functions of Z3 [17]. RedLog did not prove to be very efficient for one task. It is able to do quantifier elimination, Boolean simplification and conversion to conjunctive or disjunctive normal form, but does not have a good heuristic simplification procedures built in for different theories. The simplifier included in the decision procedure of Presburger arithmetic uses a novel guarded normal form that avoids exponential blow-up and is efficient for the formulas generated by Duration Calculus model checker as shown in [32], but Z3 resulted in better simplifications for the constraints that form the test strategy.

An example of the use of simplifier using functionality of Z3 is provided in Figure 7.3. The example is extracted from the debugging output of the ERPT tool for the model of the latte vending machine and corresponds to generating the result for $C^{+}_{l_{<} \rightarrow trap3}$ in row number 2 in Table 5.2a. The upper box shows the input in SMT-LIB v1 format given to the simplifier and the lower box shows the result.

Z3 has many different modes of simplification and these can be tuned by many parameters. The modes are basic simplifier, context simplifier and strong context simplifier, that involves the use of the satisfiability solver to check if any sub-formula is equivalent to true or false. The later simplifications are turned on by the parameter "CONTEXT_SIMPLIFIER" and "STRONG_CONTEXT_SIMPLIFIER" respectively. We achieved the best tradeoff between simplification and constraint complexity with the

```
Z3_set_param_value(cfg, "ELIM_QUANTIFIERS", "true");
Z3_set_param_value(cfg, "ELIM_AND", "false");
Z3_set_param_value(cfg, "STRONG_CONTEXT_SIMPLIFIER", "false");
Z3_set_param_value(cfg, "CONTEXT_SIMPLIFIER", "false");
simplified = Z3_simplify(ctx,formula);
Z3_update_param_value(ctx,"STRONG_CONTEXT_SIMPLIFIER", "true");
Z3_update_param_value(ctx,"ELIM_QUANTIFIERS", "false");
simplified = Z3_simplify(ctx,Z3_simplify(ctx,simplified));
Z3_update_param_value(ctx,"CONTEXT_SIMPLIFIER", "true");
Z3_update_param_value(ctx,"STRONG_CONTEXT_SIMPLIFIER", "false");
simplified = Z3_simplify(ctx,simplified);
```

Figure 7.4: A fragment of the simplifier code in C using Z3 API

approach where the basic simplification with quantifier elimination was applied first, then the double application of the strong context simplification and finally the context simplification. A fragment of the code used is depicted in Figure 7.4. The experiments were done using Z3 version 2.x, the new features of strategies and tacticals in the newer versions of Z3 give even more possibilities to tweak the simplification process. At the moment, we use a general simplification strategy, but using domain and application specific simplification could help to increase the feasibility of the method. We leave this for future research.

7.3 TEST EXECUTION

There are two different test execution environments used in the experiments. One approach generates TTCN-3 scripts and uses TestCast [58] for executing the test and communicating to the IUT. The external calls to the Z3 solver are made for satisfiability checks and constraint solving by satisfiable model finding. An alternative approach used for implementing the heuristic online testing described in Section 6.4 uses constraint-based local search programming environment Comet [34]. The tester is generated as a Comet script and constraint solver built into Comet is used in that case.

7.4 POTENTIAL IMPROVEMENTS TO THE IMPLEMENTATION

All the tools of the test generation are prototypes. More careful implementation and the use of parallel programming could improve the results significantly. As the solver technology currently undergoes rapid advances, we expect new opportunities for improving our approach to become available and further enhance the performance. In addition, many subtasks of the strategy generation could be done in parallel. Strategy generation for different traps is independent, also the constraint propagation over edges inside one propagation step could be made in parallel. This way it is possible to reduce the time spent on strategy generation significantly.

8

CASE STUDIES

We demonstrate the applicability and feasibility of our approach to test generation using three case-studies. We compare the time it takes to generate the test strategy, to generate the test that reaches a goal based on the strategy and using heuristic test generation based on the bounded strategy. The results are compared to random testing with anti-ant strategy [52] and Uppaal [50] that serves as an efficient explicit state reachability checker for EFSM models. The timers of Uppaal are not used. Some of the results of the case studies are presented in [64, 2].

All the tests were not executed against the real IUT, but against a simulation of the IUT model. No non-conformance result could be received in this way as long as erroneous behaviours have not injected in the IUT model purposefully. But it gives a better understanding of the performance, because the delays associated with the adapter, communication and the IUT are not involved.

The tests were performed on a workstation with 2.8GHz (3.4GHz with TurboBoost) Intel i7 processor and 8 GB memory. All the tools involved use a single core at a time. Most of the online test generations are done using heuristic online tester based on Comet. The tests for all-transitions and all-pairs-or-transitions are done using TestCast environment running generated TTCN3 scripts.

8.1 TRIPLE-COUNTER CASE STUDY

The case-study of triple-counter is an artificial model to demonstrate the tester's ability to select input and the order of transitions using the testing strategy in such a way that it leads to the state that is difficult to achieve randomly.

The Figure 8.1 depicts the model of triple-counter. It has three state variables x, y, z an input variable i and tree locations. The test goal is to achieve a state with $x = 10$, $y = 6$, and $z = 2$ in location l_1. The transitions

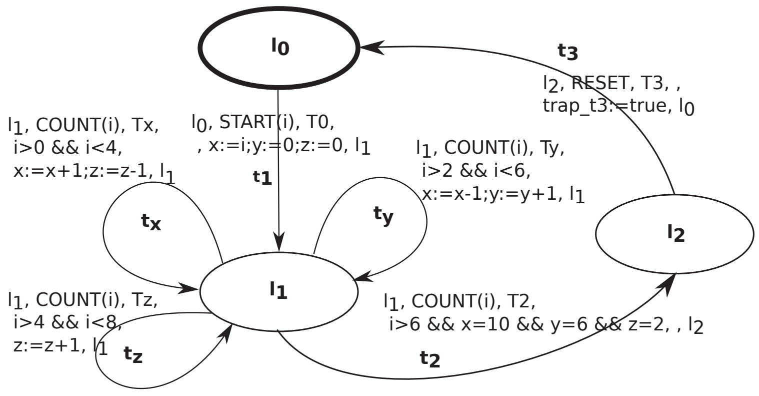

l0
t3
l2, RESET, T3, , trap_t3:=true, l0
l1, COUNT(i), Tx, i>0 && i<4, x:=x+1;z:=z-1, l1
l0, START(i), T0, , x:=i;y:=0;z:=0, l1
t1
l1, COUNT(i), Ty, i>2 && i<6, x:=x-1;y:=y+1, l1
tx
ty
l2
l1
l1, COUNT(i), Tz, i>4 && i<8, z:=z+1, l1
tz
l1, COUNT(i), T2, i>6 && x=10 && y=6 && z=2, , l2
t2

Figure 8.1: Model of triple-counter

Table 8.1: Testing results of triple-counter model

Method	time (s)	no. of transitions
strategy generation	7.45	11
online test generation	0.05	11
strategy generation, bound=2	0.70	2
χRPT online test generation	0.12	21
Uppaal	0.53	11
Anti-Ant	-	-

t_y and t_x are allowed to increase one and decrease other variable, but do not change the sum of the state variables. The transition t_z can increase z and the initial transition t_1 can assign to the value of the input parameter i to variable x. The optimal strategy seems to assign the intended sum 16 of x and y to x by the initial transition t_1, adjust the values of x and y by t_y and increase the value of z by t_z. A suitable value for the input parameter i should be selected at each step.

The results of the different test generation methods and algorithms used are provided in Table 8.1. An optimal test with 11 transitions was discovered by the strategy generator and concrete test generated on-the-fly. An optimal trace was found quite fast by explicit-state modelchecker Uppaal. The difference with Uppaal is that it finds one of the possible nondeterministic traces with inputs that may result in different reaction when applied to the IUT. For instance giving input COUNT(3) in location l_1 may trigger also transition t_x that deviates from the optimal test trace. RPT on the other hand gives strategy that is able to cope with the situation in case of the deviation and is also able to prove a deterministic test, if one exists, e.g. COUNT(4) to trigger t_y. The heuristic online tester was able to

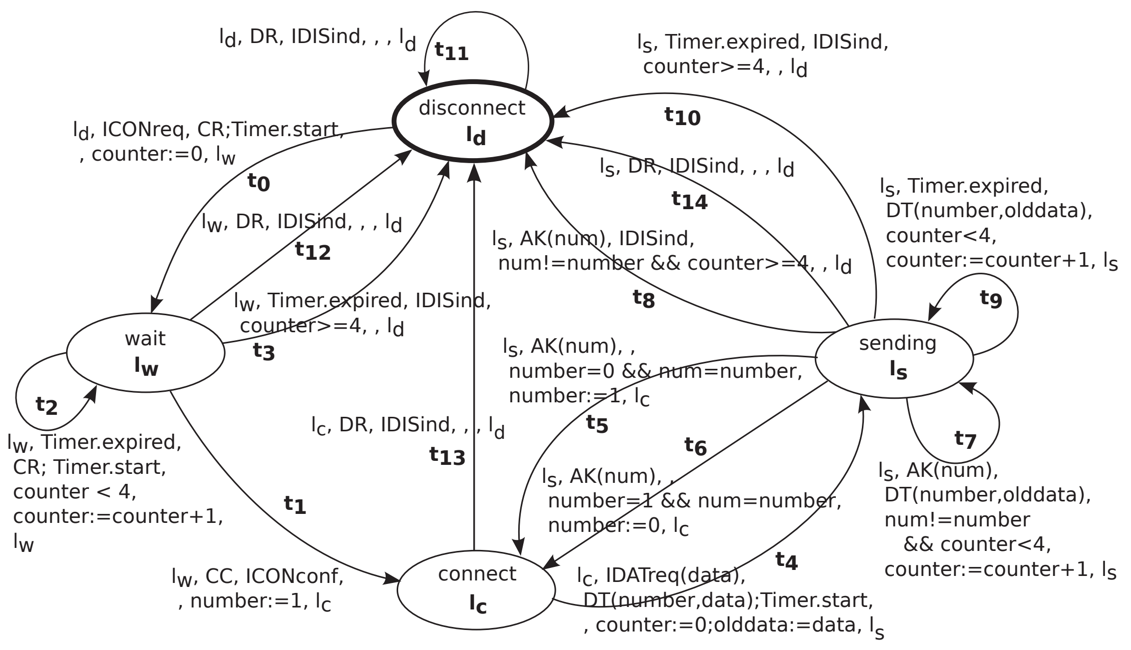

disconnect
ld
wait
lw
connect
lc
sending
ls
t0 ld, ICONreq, CR;Timer.start, , counter:=0, lw
t1 lw, CC, ICONconf, , number:=1, lc
t2 lw, Timer.expired, CR; Timer.start, counter < 4, counter:=counter+1, lw
t3 lw, Timer.expired, IDISind, counter>=4, , ld
t4 lc, IDATreq(data), DT(number,data);Timer.start, , counter:=0;olddata:=data, ls
t5 ls, AK(num), , number=0 && num=number, number:=1, lc
t6 ls, AK(num), , number=1 && num=number, number:=0, lc
t7 ls, AK(num), DT(number,olddata), num!=number && counter<4, counter:=counter+1, ls
t8 ls, AK(num), IDISind, num!=number && counter>=4, , ld
t9 ls, Timer.expired, DT(number,olddata), counter<4, counter:=counter+1, ls
t10 ls, Timer.expired, IDISind, counter>=4, , ld
t11 ld, DR, IDISind, , , ld
t12 lw, DR, IDISind, , , ld
t13 lc, DR, IDISind, , , ld
t14 ls, DR, IDISind, , , ld

Figure 8.2: Model of Inres protocol

find a non-optimal test with length 23, based on symbolic strategy with *bound* = 2. The anti-ant strategy of random testing was not able to reach the goal. It may be concluded, that even a bounded test strategy with only small bound can serve as basis for successful heuristic guidance towards the goal.

8.2 INRES PROTOCOL CASE STUDY

The initiator process of Inres protocol [37] case-study is used to illustrate many approaches of input generation to EFSM model based testing [72, 43, 20, 53]. The initiator of the connection-oriented Inres protocol is responsible for connection establishing and sending data. The IUT consists of 4 locations, 15 edges, 3 state variables and 1 input variable.

This is a simple model for test generation as can be seen from Table 8.2. It can be solved also by random Anti-Ant strategy, but the tests generated are considerably longer.

To illustrate the process of the test generation an excerpt of the constraints and distance measures generated by the offline tester synthesis are presented in Table 8.3. Traps are defined for transitions $t0 - t6$ with condition *true* and shown in the column *Goal*. The constraints and distance measures are given for a pair of transitions in columns *Via* and *Goal*, i.e. the constraint $counter < 4$ in the ninth line of the column $C^g \wedge guard$ is $C^g_{t_2 \to (t_3, true)} \wedge guard(t_2)$.

Table 8.2: Testing results of Inres model

Method	time (s)	no.of transitions
strategy generation	1.86	8
online test generation	0.032	8
strategy generation, bound=2	1.67	2
χRPT online test generation	0.043	8
Uppaal	0.530	8
Anti-Ant (min/avg/max)	0.06/0.35/0.88	17/80/193

The Table 8.4 explains the online tester behaviour for guiding the IUT towards the trap on transition *t3* from the initial state *Disconnected*. The data in the table expresses the results of constraint solving done in the online process. Empty entries mean that there was no need to solve the corresponding constraints. The column *Next* expresses the decision of the transition to be taken next and it always succeeds because of the deterministic nature of the model. Two steps similar to step 2 are omitted.

The Table 8.5 demonstrates the use of the generated constraints for guiding the IUT along the path $\langle t0; t1; t4; t6; t4; t5 \rangle$. This path is particularly difficult to achieve with random testing [20], but it is straightforward using the proposed method.

The test purpose of all-transitions is expressed by 15 traps, with one associated with different edge and having condition *true*. The offline calculation of the test strategy took 11.0 seconds and involved 296 simplification and 738 satisfiability checks. The online testing steps are fast, taking 0.055 seconds on average including trap selection, input data generation, input-output operations and simulation of the IUT. The resulting sequence of transitions is given in Table 8.6. Every row includes a sequence of transitions to the next trap and the trap covered. The traps are named according to the edges they are associated with. The next row continues the transition sequence from the state after the trap was covered.

The test purpose of all-pairs-of-transitions is expressed by 15 traps for separate transitions and 48 traps for the pairs of transitions. The offline calculation of the test strategy took 42.5 seconds and involved 1391 simplification and 3013 satisfiability checks. The online testing steps took 0.30 seconds in average including trap selection, input data generation, input-output operations and simulation of the IUT. The resulting sequence of transitions is given in Table 8.7. The traps for separate transitions are the same as for all-transitions purpose. The traps for the pairs are named

Table 8.3: Excerpt of generated constraints for the Inres Initiator example

Via	Goal	$C^g \wedge guard$	L	L^+
t0	t0	true	1	1
t11	t0	false	2	2
t1	t1	true	1	1
t2	t1	false	2	2
t3	t1	false	3	3
t12	t1	false	3	3
t0	t3	true	6	6
t1	t3	false	8	8
t2	t3	counter < 4	2	5
t11	t3	false	7	7
t3	t3	counter ≥ 4	1	1
t12	t3	false	7	7
t4	t4	true	1	1
t13	t4	false	4	4
t5	t5	number = 0	1	1
t6	t5	number = 1	3	3
t7	t5	false	2	2
t8	t5	false	7	7
t9	t5	false	2	2
t10	t5	false	7	7
t14	t5	false	7	7
t5	t6	number = 0	3	3
t6	t6	number = 1	1	1
t7	t6	false	2	2
t8	t6	false	5	5
t9	t6	false	2	2
t10	t6	false	5	5
t14	t6	false	5	5

Table 8.4: Creating a path to reach the transition t3 from the state Disconnected

Step	(L, counter, number)	Via	To	C^g	input	Next
1	(Disconnected,_,_,)	t0 t11	t3 t3	*true*	ICONreq	t0
2	(Waiting,0,_)	t3 t2 t1 t12	t3 t3 t3 t3	*false* *true*	Timer.timeout	t2
	...					
5	(Waiting,3,_)	t3 t2 t1 t12	t3 t3 t3 t3	*false* *true*	Timer.timeout	t2
6	(Waiting,4,_)	t3 t2 t1 t12	t3 t3 t3 t3	*true*	Timer.timeout	t3

Table 8.5: Executing the transition path t0;t1;t4;t6;t4;t5

Step	(L, counter, number)	Via	To	C^g	input	Next
1	(Disconnected,_,_)	t0	t0	*true*	ICONreq	t0
		t11	t0			
2	(Waiting,0,_)	t1	t1	*true*	CC	t1
		t2	t1			
		t3	t1			
		t12	t1			
3	(Connected,0,1)	t4	t4	*true*	IDATreq	t4
		t13	t4			
4	(Sending,0,1)	t6	t6	*true*	AK(1)	t6
		t7	t6			
		t9	t6			
		t5	t6			
		t8	t6			
		t10	t6			
		t14	t6			
5	(Connected,0,0)	t4	t4	*true*	IDATreq	t4
		t13	t4			
6	(Sending,0,0)	t5	t5	*true*	AK(0)	t5
		t7	t5			
		t9	t5			
		t6	t5			
		t8	t5			
		t10	t5			
		t14	t5			

Table 8.6: All transitions test purpose

transitions	traps
t0	t0
t1	t1
t4	t4
t7	t7
t6	t6
t4,t5	t5
t4,t14	t14
t0,t12	t12
t0,t2	t2
t1,t13	t13
t11	t11
t0,t1,t4,t9	t9
t9,t9,t9,t8	t8
t0,t2,t2,t2,t2,t3	t3
t0,t1,t4,t9,t9,t9,t9,t10	t10

Table 8.7: All transitions and pairs of transitions

transitions	traps	transitions	traps
t11	t11	t0,t1,t4,t14	t4t14
t0	t0,t11t0	t0	t14t0
t2	t2,t0t2	t1,t4,t7,t9	t7t9
t1	t1,t2t1	t9	t9t9
t4	t4,t1t4	t14	t9t14
t9	t9,t4t9	t0,t1,t4,t7,t6	t7t6
t6	t6,t9t6	t4,t7,t5	t7t5
t4	t6t4	t4,t7,t7	t7t7
t7	t7,t4t7	t7,t7,t8	t8,t7t8
t14	t14,t7t14	t11	t8t11
t11	t14t11	t0,t1,t4,t7,t7,t7,t7,t10	t10,t7t10
t0,t12	t12,t0t12	t11	t10t11
t11	t12t11	t0,t1,t4,t7,t7,t7,t9,t8	t9t8
t11	t11t11	t0	t8t0
t0,t1	t0t1	t12,t0	t12t0
t13	t13,t1t13	t2,t2	t2t2
t11	t13t11	t1,t4,t9,t7	t9t7
t0,t1,t4,t6	t4t6	t6,t4,t9,t5	t9t5
t13	t6t13	t4,t7,t7,t7,t9,t10	t9t10
t0	t13t0	t0	t10,t0
t1,t4,t6,t4,t5	t5,t4t5	t2,t2,t2,t2,t3	t3,t2t3
t4	t5t4	t11	t3t11
t6,t4,t5	t5t13	t0,t2,t2,t2,t2,t3,t0	t3t0

accordingly, e.g. "t0t1" means that a transition over edge t1 was taken immediately after t0. For formalizing the pair-of-transition goal, an auxiliary variable *prev* was added to the model and updated to the identifier of edge in every transition. The trap condition then requires a certain transition to precede the current edge. For instance, a trap "t0t1" is specified as $(t1, prev = t0)$ in that case.

8.3 INDUSTRIAL CASE STUDY: BILLING SYSTEM

We demonstrate the scalability of the method on industrial case-study of telecom billing system[1]. The model describes billing of mobile internet usage depending on the service used (WAP or general internet) and billing rules of the contract. The model is created by a test engineer based on the IUT informal specification. The values of the parameters and constants used here are different from the actual values of the real system tested.

The model formalises the rules of billing. The rules can give some bonus usage for using other services, some free usage for a fixed price and usage dependent price with possible daily limit of the price charged. The model has 13 locations and 43 edges between them, 1 input variables and 8 state variables, 5 of those having domain [0, 32000] and 1 unconstrained integer. In average, guards in this model consist of 20 terms connected by functions and predicates. The model of the billing system is depicted in Figure 8.3. It is formalized as an Uppaal model for compatibility with another test generation method. The variable inp can be taken as semantic variable *iLabel*, input labels encoded to integers 1-11 and val as an input parameter with domain $[0, 32000]$.

The test purpose of the results presented in Table 8.8 is to test the situation where the monthly internet usage limit is exceeded. This includes many data sessions for exceeding the free and bonus limits and taking daily limits into account. The shortest path to the trap has 189 transitions found by the symbolic analysis in strategy generation. The results are presented for generating and using the full strategy that covers the full path and bounded strategies with different bounds. χRPT online test generation is used to guide the testing in that case.

The results show that the strategy generation scales with the increase of the bound reasonably. The dependence is not linear, but feasible in practice, considering that the involved procedures with Presburger arithmetic have triple-exponential worst-case complexity. The χRPT approach com-

1 We cannot be more specific about the origin of the case study because of the non-disclosure agreement involved.

Table 8.8: Results of billing case study

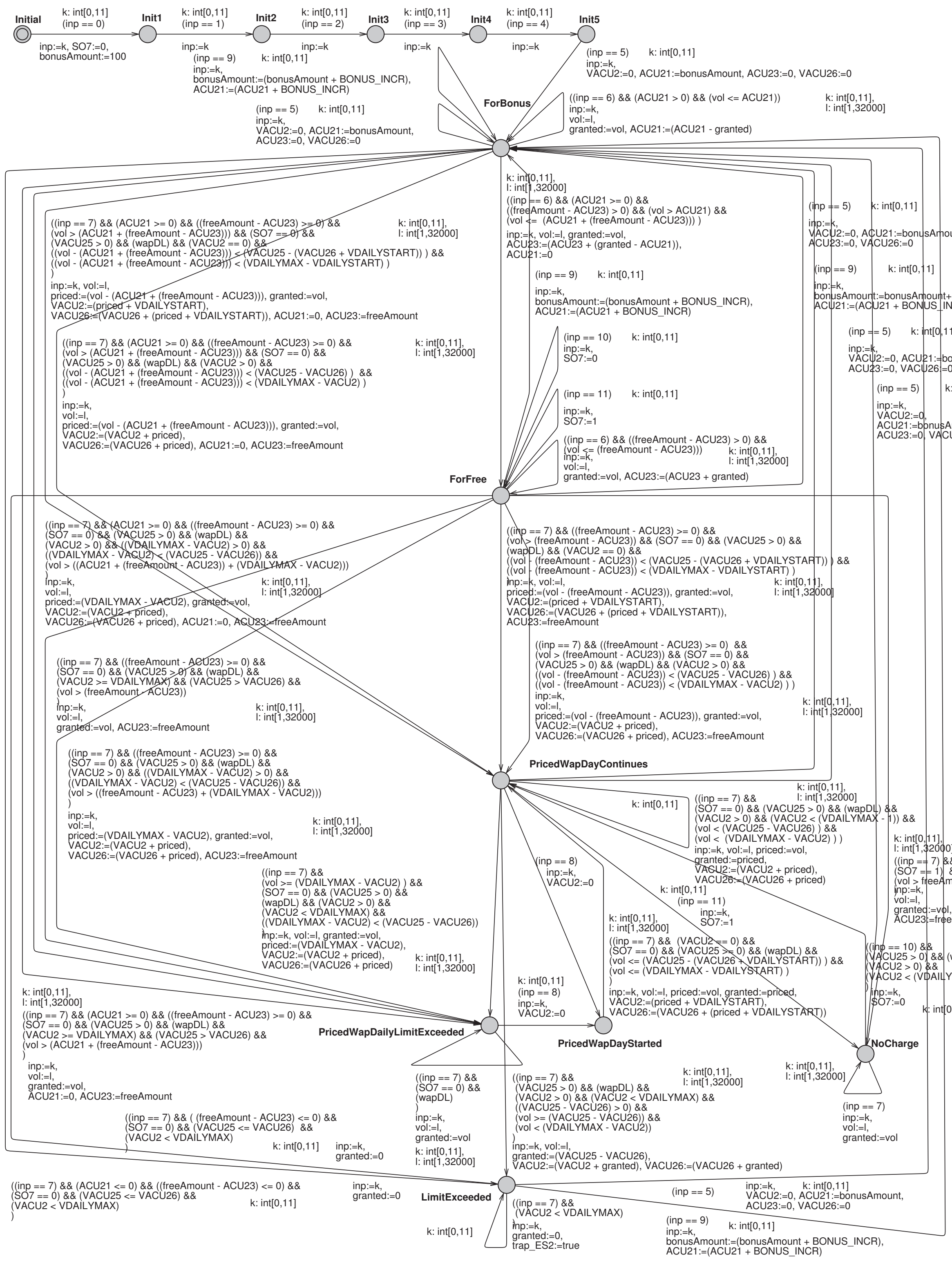
Bound	200	100	20	10	2
Strategy generation time (sec)	1958	1212	136.2	36.3	5.7
Online time (sec)	2.6	6.7	17.4	17.0	153.4
Online test length (transitions)	189	230	255	275	1051
Avg time for test step (sec)	0.014	0.051	0.084	0.063	0.146

plements the bounded strategy generation such that the online procedure finds shorter test and uses less time with a higher bound strategy and vice versa. This enables us to find a good balance between the effort of generating of strategy and online test generation. It also enables to test the systems with more complex models when the full strategy generation is not feasible.

Anti-ant and Uppaal were not able to find a test for this model because of the huge search space. Anti-ant did not complete within 24 hours of working and Uppaal reached the memory limit in minutes with all possible different tuning options (depth/breath-first search, state space reduction techniques) available.

8.4 SUMMARY

We showed the feasibility of the method using three case studies in this chapter. The case studies used are medium sized, but not trivial. The complexity of a model is not easy to express in terms of the number of locations, transitions, variables, size of domain or formulas used, because the actual complexity depends on the interplay of all these factors and the background theory used. It is possible to construct a small model that turns out to be too complex for the proposed method of test generation. We demonstrated on the case-study of Inres protocol that the problem that has been challenging for other approaches [43, 20] is easy for our approach. As the case studies show, this kind of test generation from symbolic analysis based test strategy starts to be feasible for practical usage and this is connected to the great advances in SMT solver development in the recent years.


Initial
Init1
Init2
Init3
Init4
Init5
k: int[0,11] (inp == 0)
inp:=k, SO7:=0, bonusAmount:=100
k: int[0,11] (inp == 1)
inp:=k
k: int[0,11] (inp == 2)
inp:=k
k: int[0,11] (inp == 3)
inp:=k
k: int[0,11] (inp == 4)
inp:=k
(inp == 9) k: int[0,11]
inp:=k, bonusAmount:=(bonusAmount + BONUS_INCR), ACU21:=(ACU21 + BONUS_INCR)
(inp == 5) k: int[0,11]
inp:=k, VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
ForBonus
((inp == 6) && (ACU21 > 0) && (vol <= ACU21)) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, granted:=vol, ACU21:=(ACU21 - granted)
(inp == 5) k: int[0,11]
inp:=k, VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
k: int[0,11], l: int[1,32000]
((inp == 6) && (ACU21 >= 0) && ((freeAmount - ACU23) > 0) && (vol > ACU21) && (vol <= (ACU21 + (freeAmount - ACU23))))
inp:=k, vol:=l, granted:=vol, ACU23:=(ACU23 + (granted - ACU21)), ACU21:=0
((inp == 7) && (ACU21 >= 0) && ((freeAmount - ACU23) >= 0) && (vol > (ACU21 + (freeAmount - ACU23))) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 == 0) && ((vol - (ACU21 + (freeAmount - ACU23))) < (VACU25 - (VACU26 + VDAILYSTART))) && ((vol - (ACU21 + (freeAmount - ACU23))) < (VDAILYMAX - VDAILYSTART))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(vol - (ACU21 + (freeAmount - ACU23))), granted:=vol, VACU2:=(priced + VDAILYSTART), VACU26:=(VACU26 + (priced + VDAILYSTART)), ACU21:=0, ACU23:=freeAmount
(inp == 9) k: int[0,11]
inp:=k, bonusAmount:=(bonusAmount + BONUS_INCR), ACU21:=(ACU21 + BONUS_INCR)
(inp == 5) k: int[0,11]
inp:=k, VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
(inp == 9) k: int[0,11]
inp:=k, bonusAmount:=bonusAmount+BONUS_INCR, ACU21:=(ACU21 + BONUS_INCR)
(inp == 5) k: int[0,11]
inp:=k, VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
(inp == 5) k: int[0,11]
inp:=k, VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
((inp == 7) && (ACU21 >= 0) && ((freeAmount - ACU23) >= 0) && (vol > (ACU21 + (freeAmount - ACU23))) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && ((vol - (ACU21 + (freeAmount - ACU23))) < (VACU25 - VACU26)) && ((vol - (ACU21 + (freeAmount - ACU23))) < (VDAILYMAX - VACU2))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(vol - (ACU21 + (freeAmount - ACU23))), granted:=vol, VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced), ACU21:=0, ACU23:=freeAmount
(inp == 10) k: int[0,11]
inp:=k, SO7:=0
(inp == 11) k: int[0,11]
inp:=k, SO7:=1
((inp == 6) && ((freeAmount - ACU23) > 0) && (vol <= (freeAmount - ACU23))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, granted:=vol, ACU23:=(ACU23 + granted)
ForFree
((inp == 7) && (ACU21 >= 0) && ((freeAmount - ACU23) >= 0) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && ((VDAILYMAX - VACU2) > 0) && ((VDAILYMAX - VACU2) < (VACU25 - VACU26)) && (vol > ((ACU21 + (freeAmount - ACU23)) + (VDAILYMAX - VACU2)))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(VDAILYMAX - VACU2), granted:=vol, VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced), ACU21:=0, ACU23:=freeAmount
((inp == 7) && ((freeAmount - ACU23) >= 0) && (vol > (freeAmount - ACU23)) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 == 0) && ((vol - (freeAmount - ACU23)) < (VACU25 - (VACU26 + VDAILYSTART))) && ((vol - (freeAmount - ACU23)) < (VDAILYMAX - VDAILYSTART))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(vol - (freeAmount - ACU23)), granted:=vol, VACU2:=(priced + VDAILYSTART), VACU26:=(VACU26 + (priced + VDAILYSTART)), ACU23:=freeAmount
((inp == 7) && ((freeAmount - ACU23) >= 0) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 >= VDAILYMAX) && (VACU25 > VACU26) && (vol > (freeAmount - ACU23))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, granted:=vol, ACU23:=freeAmount
((inp == 7) && ((freeAmount - ACU23) >= 0) && (vol > (freeAmount - ACU23)) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && ((vol - (freeAmount - ACU23)) < (VACU25 - VACU26)) && ((vol - (freeAmount - ACU23)) < (VDAILYMAX - VACU2))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(vol - (freeAmount - ACU23)), granted:=vol, VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced), ACU23:=freeAmount
PricedWapDayContinues
((inp == 7) && ((freeAmount - ACU23) >= 0) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && ((VDAILYMAX - VACU2) > 0) && ((VDAILYMAX - VACU2) < (VACU25 - VACU26)) && (vol > ((freeAmount - ACU23) + (VDAILYMAX - VACU2)))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=(VDAILYMAX - VACU2), granted:=vol, VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced), ACU23:=freeAmount
k: int[0,11]
((inp == 7) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && (VACU2 < (VDAILYMAX - 1)) && (vol < (VACU25 - VACU26)) && (vol < (VDAILYMAX - VACU2))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, priced:=vol, granted:=priced, VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced)
((inp == 7) && (vol >= (VDAILYMAX - VACU2)) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && (VACU2 < VDAILYMAX) && ((VDAILYMAX - VACU2) < (VACU25 - VACU26))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, granted:=vol, priced:=(VDAILYMAX - VACU2), VACU2:=(VACU2 + priced), VACU26:=(VACU26 + priced)
(inp == 8) inp:=k, VACU2:=0
k: int[0,11] (inp == 11) inp:=k, SO7:=1
k: int[0,11], l: int[1,32000]
((inp == 7) && (VACU2 == 0) && (SO7 == 0) && (VACU25 >= 0) && (wapDL) && (vol <= (VACU25 - (VACU26 + VDAILYSTART))) && (vol <= (VDAILYMAX - VDAILYSTART)))
inp:=k, vol:=l, priced:=vol, granted:=priced, VACU2:=(priced + VDAILYSTART), VACU26:=(VACU26 + (priced + VDAILYSTART))
k: int[0,11], l: int[1,32000] ((inp == 7) && (SO7 == 1) && (vol > freeAmount-ACU23) inp:=k, vol:=l, granted:=vol, ACU23:=freeAmount
((inp == 10) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && (VACU2 < (VDAILYMAX - 1))) inp:=k, SO7:=0 k: int[0,11]
k: int[0,11], l: int[1,32000]
((inp == 7) && (ACU21 >= 0) && ((freeAmount - ACU23) >= 0) && (SO7 == 0) && (VACU25 > 0) && (wapDL) && (VACU2 >= VDAILYMAX) && (VACU25 > VACU26) && (vol > (ACU21 + (freeAmount - ACU23))))
inp:=k, vol:=l, granted:=vol, ACU21:=0, ACU23:=freeAmount
PricedWapDailyLimitExceeded
k: int[0,11] (inp == 8) inp:=k, VACU2:=0
PricedWapDayStarted
NoCharge
((inp == 7) && (SO7 == 0) && (wapDL)) inp:=k, vol:=l, granted:=vol k: int[0,11], l: int[1,32000]
((inp == 7) && (VACU25 > 0) && (wapDL) && (VACU2 > 0) && (VACU2 < VDAILYMAX) && ((VACU25 - VACU26) > 0) && (vol >= (VACU25 - VACU26)) && (vol < (VDAILYMAX - VACU2))) k: int[0,11], l: int[1,32000]
inp:=k, vol:=l, granted:=(VACU25 - VACU26), VACU2:=(VACU2 + granted), VACU26:=(VACU26 + granted)
k: int[0,11], l: int[1,32000]
(inp == 7) inp:=k, vol:=l, granted:=vol
((inp == 7) && ((freeAmount - ACU23) <= 0) && (SO7 == 0) && (VACU25 <= VACU26) && (VACU2 < VDAILYMAX)) k: int[0,11]
inp:=k, granted:=0
((inp == 7) && (ACU21 <= 0) && ((freeAmount - ACU23) <= 0) && (SO7 == 0) && (VACU25 <= VACU26) && (VACU2 < VDAILYMAX)) k: int[0,11]
inp:=k, granted:=0
LimitExceeded
((inp == 7) && (VACU2 < VDAILYMAX)) inp:=k, granted:=0, trap_ES2:=true
k: int[0,11]
(inp == 5) inp:=k, k: int[0,11] VACU2:=0, ACU21:=bonusAmount, ACU23:=0, VACU26:=0
(inp == 9) k: int[0,11] inp:=k, bonusAmount:=(bonusAmount + BONUS_INCR), ACU21:=(ACU21 + BONUS_INCR)

Constants:	Variables:
const int freeAmount = 372;	int granted;
const bool wapDL = true;	int[0,1] SO7;
const int VACU25 = 20000;	int[0,1000] bonusAmount;
const int VDAILYSTART = 37;	int[0,32000] ACU2;
const int VDAILYMAX = 222;	int[0,32000] ACU21;
const int BONUS_INCR = 10;	int[0,32000] ACU23;
	int[0,32000] ACU26;
	int[0,32000] priced;

Figure 8.3: Billing system fragment

9 CONCLUSIONS AND FUTURE WORK

The goal of the thesis is to provide a method for online test generation based on a model of system under test and a test purpose. The proposed method involves modelling systems with EFSMs, formalization of test purpose by a set of traps associated with the edges of the model, offline symbolic test strategy generation and online test generation using the test strategy.

Model-based test generation for non-deterministic models with data is a difficult task in general. Non-determinism in the model does not permit the generation of fixed test sequences and our way to solve it is by online test generation. On the other hand, data dependencies in the control structure require thorough analysis to find feasible runs that cover the test goal and the efficiency of the online test generation is an important concern. This thesis proposes a computationally feasible method for online test generation.

The chapter summarizes the work and plans for future developments.

SUMMARY

A modelling formalism of output-observable I/O-EFSM as proposed in Chapter 3 provides an efficient test strategy and test data generation by limiting the non-determinism allowed in the model. The rigorous definition and discussion of the benefits and limits of the output-observability assumption is an important contribution of the thesis. The method proposed is independent of the background first order theory used for expressing the guards and updates of the I/O-EFSM model introduced in Chapter 3. It just requires that the SMT problem is decidable for the theory.

The method is designed to check conformance of the IUT to the model in respect to a test purpose. A test engineer is usually interested in testing a specific aspects of the IUT or use some coverage criteria on the structural elements of the model rather than performing random testing. It is shown in Chapter 4 how to formalize several different test purposes by a set of traps and auxiliary variables. The definition of traps and conformance is given and discussed in relation to alternative approaches. The proposed method generates a strategy with the purpose of covering all traps of the test purpose as efficiently as possible.

A symbolic testing strategy is used to guide the online testing process efficiently. It is shown in Chapter 5 how to represent and generate the strategy using backwards symbolic analysis of the model. This is the most resource-intensive part of the proposed method, but it can be carried out offline prior to the actual testing process. In Chapter 6 we demonstrate how the symbolic test strategy is used for efficient online test planning. The trap ordering is chosen based on their distance estimates and inputs are generated by solving the constraints included in the strategy.

The feasibility of the method is demonstrated on three case studies in Chapter 8. The proposed method reduces the problem to solving and simplifying complex algebraic constraints and it is applicable to industrial scale problems only because of the advances made in the SMT solver technology.

MAIN CONTRIBUTIONS

The main contribution of this thesis is a MBT approach for efficient online testing based on a test purpose and EFSM model with limited non-determinism. More specifically:

- Definition of output-observable EFSM (Section 3.2) with limited non-determinism that is a key prerequisite for efficient model-based online testing.

- Test purpose specification using traps (Section 4.3) and auxiliary variables that enable to specify both scenario and structural coverage type of test purposes.

- A method for symbolic test strategy representation and generation (Chapter 5). The computationally expensive strategy generation is carried out offline, making it possible to generate the actual test inputs efficiently online.

- A method for efficient online test generation based on the test strategy for achieving feasible path to the test goals with close to optimal length (Chapter 6).

- Demonstration of the feasibility of symbolic reachability analysis using algebraic simplifications and quantifier elimination procedures that are supported by contemporary SMT solvers.
- Empirical validation of the feasibility of the method using three case-studies including one model designed by an industrial partner.

AUTHOR'S CONTRIBUTION TO THE PUBLICATIONS

The main contribution of this thesis is based on the results of three papers [65, 42, 2] included in Part II of the thesis.

Author contributed to Paper I [65] as a collaborator in the development of the ideas first presented in [66], participating in the development of the case-study environment of Feeder Box Control Unit (FBCU) of the street lighting subsystem, conducting the case-study, and participating in writing different parts of the book chapter.

Author was the main contributor of Paper II [42], both in developing the ideas and writing the paper. This is a concise presentation of the main ideas elaborated in detail in this thesis.

Paper III [2] is the result of a project made in collaboration with a master student Danel Ahman. Author of the thesis proposed the problem, idea of the solution, supervised the project, contributed in developing and performing the case-studies and writing some introductory and general parts of the paper. The details of the method proposed in the paper and implementation of the method was conducted by the co-author.

Some of the results of case-studies included in the thesis are published also in [64]. The copy of the book chapter is not included in the thesis because of the non-conforming classification of the publication and unsolved copyright issues.

FUTURE WORK

There are several ideas and problems that arose during the research that require further investigation.

Output observability is the main restricting assumption in the approach presented. Relaxation of the assumption could give more freedom in modelling systems and is a pre-requisite for other extensions. On the other hand, the complexity of offline analysis and online test generation would rise without the assumption considerably as discussed in Subsection 3.2.3. Thus it requires further investigation to study ways of relaxing the assumption.

Composition is an important means both in modelling systems and test purposes. Extension to composition with a scenario automaton for limiting non-determinism is quite straight forward, but compositional modelling of the IUT is more problematic as discussed in Subsection 3.2.4. Compositional modelling would need the relaxation of output-observability assumption or defining composition operator that preserves output-observability.

Extension of the modelling formalism to support hierarchical automata with a formal mapping from UML Statecharts is an extension that we believe would simplify the evaluation of the method on industrial models. Some preliminary work has been done to extend the test generation to models of a simplified version of Hierarchical Timed Automaton (HTA) [15] without timers. This is a subset of Statecharts with strict and suitably defined semantics. The preliminary results show that one of the benefits from hierarchical modelling would be that the local variables in the component automaton could be eliminated from the strategy constraints on higher levels, thus making reasoning more local and resulting in more compact formulae.

Extending test generation to timed automata or hierarchical timed automata models would be quite a natural direction of further research. That would need adding difference logic [55] to the background theory and including special simplification procedures for efficient handling of timing constraints. That would include lifting the results or representing non-convex regions using Clock Difference Diagrams [49] to the domain of logical formulae.

The general heuristic simplification procedures provided by external tools have been used so far. Special simplification methods for constraints resulting from the test generation could be developed. Also an interface for adding optional domain and example specific simplification rules could be added.

Test purposes are stated in terms of trap variables and auxiliary variables. This enables us to specify quite complex properties that include counters and history variables. It would be useful to give a formal analysis and characterisation of what kind of properties can be encoded in this way. Also the use of additional automaton composed to the model of the IUT can be used for test purpose specification if the parallel composition of automaton is defined.

There are many improvements that could be made in implementation of the method. Simplification and satisfiability checking using SMT solver is implemented with text interface in SMT-LIB format at the moment that causes repeated parsing of the formulae. Implementation using the solver

API with the internal representation of the formulae would make the implementation much more efficient. Also, the use of version 4 of the Z3 solver with several new options of tuning the simplification process in conjunction with the use of strategies and tactics in satisfiability checking and model generation could improve the performance significantly. Such a new version of test strategy generator is under development currently. Many subtasks of the strategy generation could be done in parallel thus reducing the time spent on strategy generation significantly.

Several of the above goals are prerequisites for developing the method to a level where it could be used for test generation in an industrial setting.

BIBLIOGRAPHY

[1] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004. (Cited on pages 19 and 21.)

[2] Danel Ahman and Marko Kääramees. Constraint-based heuristic online test generation from non-deterministic I/O EFSMs. In Alexander K. Petrenko and Holger Schlingloff, editors, *MBT*, volume 80 of *EPTCS*, pages 115–129, 2012. (Cited on pages 79, 91, 92, 99, and 113.)

[3] Bernhard Aichernig, Willibald Krenn, Henrik Eriksson, and Jonny Vinter. D 1.2 - State of the art survey - Part a: Model-based test case generation. Technical report, June 2008. MOGENTES public project deliverable. (Cited on page 24.)

[4] Luca De Alfaro. Game models for open systems. In *Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday, volume 2772 of LNCS*, pages 269–289. Springer, 2004. (Cited on page 54.)

[5] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Symbolic algorithms for infinite-state games. In *Proceedings of the 12th International Conference on Concurrency Theory*, CONCUR '01, pages 536–550, London, UK, 2001. Springer-Verlag. (Cited on pages 37, 67, and 78.)

[6] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *Int. J. Softw. Tools Technol. Transf.*, 11(1):69–83, January 2009. (Cited on page 67.)

[7] Axel Belinfante, Jan Feenstra, René G. de Vries, Jan Tretmans, Nicolae Goga, Loe M. G. Feijs, Sjouke Mauw, and Lex Heerink. Formal test automation: A simple experiment. In *Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems*, pages 179–196, Deventer, The Netherlands, 1999. Kluwer, B.V. (Cited on page 25.)

[8] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, 1999. Springer-Verlag. (Cited on page 67.)

[9] Ed Brinksma and Jan Tretmans. Testing transition systems: An annotated bibliography. In *Proceedings of the 4th Summer School on Modeling and Verification of Parallel Processes*, MOVEP '00, pages 187–195, London, UK, 2001. Springer-Verlag. (Cited on page 24.)

[10] Laura Brandán Briones and Ed Brinksma. A test generation framework for quiescent real-time systems. In *Proceedings of the 4th international conference on Formal Approaches to Software Testing*, FATES'04, pages 64–78, Berlin, Heidelberg, 2005. Springer-Verlag. (Cited on page 25.)

[11] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. (Cited on page 24.)

[12] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005. (Cited on pages 26, 67, and 78.)

[13] Samuel T. Chanson and Jinsong Zhu. A unified approach to protocol test sequence generation. In *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future*, pages 106–114. IEEE, 1993. (Cited on page 25.)

[14] Yanping Chen, Robert L. Probert, and Hasan Ural. Model-based regression test suite generation using dependence analysis. In *Proceedings of the 3rd international workshop on Advances in model-based testing*, A-MOST '07, pages 54–62, New York, NY, USA, 2007. ACM. (Cited on page 21.)

[15] Alexandre David. *Hierarchical Modeling and Analysis of Timed Systems*. PhD thesis, 2003. Uppsala: Mathematics and Computer Science, Department of Information Technology, Sweden. (Cited on page 114.)

[16] Alexandre David, Johann Deneux, and Julien d'Orso. A formal semantics for UML statecharts. Technical Report 2003-010, Department of Information Technology, Uppsala University, 2003. (Cited on page 37.)

[17] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. (Cited on pages 93 and 96.)

[18] René G. de Vries. Towards formal test purposes. In G. J. Tretmans and H. Brinksma, editors, *Formal Approaches to Testing of Software 2001 (FATES'01), Aarhus, Denmark*, volume NS-01-4 of *BRICS Notes Series*, pages 61–76, Aarhus, Denkmark, August 2001. (Cited on page 25.)

[19] Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo. Generating feasible input sequences for extended finite state machines (EFSMs) using genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1081–1082, New York, NY, USA, 2005. ACM. (Cited on pages 88 and 91.)

[20] Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo. Estimating the feasibility of transition paths in extended finite state machines. *Automated Software Engineering*, 17(1):33–56, 2010. (Cited on pages 25, 26, 101, 102, and 109.)

[21] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975. (Cited on page 65.)

[22] Andreas Dolzmann and Thomas Sturm. REDLOG: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, June 1997. (Cited on page 96.)

[23] Ali Y. Duale and M. Ümit Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Trans. Comput.*, 53(5):614–627, 2004. (Cited on page 25.)

[24] Loe M. G. Feijs, Nicolae Goga, and Sjouke Mauw. Probabilities in the TorX test derivation algorithm. In *2000 – 2 nd Workshop on SDL and MSC, pages 173– 188. VERIMAG, IRISA, SDL Forum Society*, pages 173–188, 2000. (Cited on page 25.)

[25] Roger Ferguson and Bogdan Korel. Generating test data for distributed software using the chaining approach. *Inf. Softw. Technol.*, 38(5):343–353, 1996. (Cited on page 25.)

[26] Lars Frantzen, Jan Tretmans, and Tim A. C. Willemse. Test generation based on symbolic specifications. In *FATES 2004, number 3395 in LNCS*, pages 1–15. Springer-Verlag, 2005. (Cited on pages 25, 48, 50, 51, 53, and 54.)

[27] Mario Friske, Bernd-Holger Schlingloff, and Stephan Weißleder. Composition of model-based test coverage criteria. In Holger Giese, Michaela Huhn, Ulrich Nickel, and Bernhard Schätz, editors, *MBEES*, volume 2008-2 of *Informatik-Bericht*, pages 87–94. TU Braunschweig, Institut für Software Systems Engineering, 2008. (Cited on page 22.)

[28] Patrice Godefroid. Compositional dynamic test generation. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '07, pages 47–54, New York, NY, USA, 2007. ACM. (Cited on page 25.)

[29] Patrice Godefroid, Peli de Halleux, Aditya V. Nori, Sriram K. Rajamani, Wolfram Schulte, Nikolai Tillmann, and Michael Y. Levin. Automating software testing using program analysis. *IEEE Software*, 25(5):30–37, 2008. (Cited on page 26.)

[30] Wolfgang Grieskamp. Microsoft's protocol documentation program: A success story for model-based testing. In Leonardo Bottaci and Gordon Fraser, editors, *TAIC PART*, volume 6303 of *Lecture Notes in Computer Science*, page 7. Springer, 2010. (Cited on page 21.)

[31] Grégoire Hamon, Leonardo de Moura, and John Rushby. Generating efficient test sets with a model checker. In *SEFM '04: Proceedings of the Software Engineering and Formal Methods, Second International Conference*, pages 261–270, Washington, DC, USA, 2004. IEEE Computer Society. (Cited on pages 25, 26, and 27.)

[32] Michael R. Hansen and Aske Wiid Brekling. On tool support for duration calculus on the basis of Presburger arithmetic. In Carlo Combi, Martin Leucker, and Frank Wolter, editors, *TIME*, pages 115–122. IEEE, 2011. (Cited on page 96.)

[33] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, June 1987. (Cited on page 37.)

[34] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. MIT Press, 2005. (Cited on pages 91 and 97.)

[35] Anders Hessel, Kim Larsen, Marius Mikučionis, Brian Nielsen, Paul Pettersson, and Arne Skou. Testing Real-Time systems using UPPAAL. In *Formal Methods and Testing*, pages 77–117. 2008. (Cited on pages 25, 26, 50, and 53.)

[36] Charles Antony Richard Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. (Cited on page 50.)

[37] Dieter Hogrefe. OSI formal specification case study: The Inres protocol and service. Technical Report 91-012, University of Bern, Switzerland, 1991. (Cited on page 101.)

[38] Harry Hsieh, Felice Balarin, Luciano Lavagno, and Alberto L. Sangiovanni-Vincentelli. Synchronous approach to the functional equivalence of embedded system implementations. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(8):1016–1033, 2001. (Cited on page 50.)

[39] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, New York, NY, USA, 2004. (Cited on page 37.)

[40] Jonathan Jacky, Colin Campbell, Wolfram Schulte, and Margus Veanes. *Model-Based Software Testing and Analysis with C#*. Cambridge Univ. Press, Leiden, 2008. (Cited on pages 21, 50, 58, and 91.)

[41] Claude Jard and Thierry Jeron. TGV: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Int. J. Softw. Tools Technol. Transf.*, 7(4):297–315, August 2005. (Cited on pages 26, 58, and 91.)

[42] Marko Kääramees, Jüri Vain, and Kullo Raiend. Synthesis of on-line planning tester for non-deterministic EFSM models. In Leonardo Bottaci and Gordon Fraser, editors, *Testing - Practice and Research Techniques*, volume 6303 of *LNCS*, pages 147–154. Springer Berlin / Heidelberg, 2010. (Cited on pages 53, 63, 79, and 113.)

[43] Abdul Salam Kalaji, Robert M. Hierons, and Stephen Swift. A search-based approach for automatic test generation from extended finite state machine (EFSM). In *TAIC-PART '09: Proceedings of the 2009*

Testing: Academic and Industrial Conference - Practice and Research Techniques, pages 131–132, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on pages 24, 25, 37, 88, 91, 101, and 109.)

[44] Bogdan Korel and Ali M. Al-Yami. Assertion-oriented automated test data generation. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 71–80, Washington, DC, USA, 1996. IEEE Computer Society. (Cited on pages 25 and 26.)

[45] Nikolai Kosmatov, Bruno Legeard, Fabien Peureux, and Mark Utting. Boundary coverage criteria for test generation from formal models. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE '04, pages 139–150, Washington, DC, USA, 2004. IEEE Computer Society. (Cited on page 87.)

[46] R. Lai. A survey of communication protocol testing. *J. Syst. Softw.*, 62(1):21–46, 2002. (Cited on page 24.)

[47] Axel van Lamsweerde. Formal specification: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 147–159, New York, NY, USA, 2000. ACM. (Cited on page 22.)

[48] Kim G. Larsen, Marius Mikučionis, Brian Nielsen, and Arne Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 299–306, New York, NY, USA, 2005. ACM. (Cited on page 25.)

[49] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic J. of Computing*, 6(3):271–298, September 1999. (Cited on page 114.)

[50] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997. (Cited on page 99.)

[51] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines – a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996. (Cited on page 24.)

[52] Huaizhong Li and Chiou Peng Lam. Using anti-ant-like agents to generate test threads from the UML diagrams. In Ferhat Khendek and Rachida Dssouli, editors, *TestCom*, volume 3502 of *Lecture Notes in Computer Science*, pages 69–80. Springer, 2005. (Cited on pages 26, 87, 88, 91, and 99.)

[53] Gang Luo, Gregor von Bochmann, and Alexandre Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Trans. Softw. Eng.*, 20(2):149–162, 1994. (Cited on page 101.)

[54] Lev Nachmanson, Margus Veanes, Wolfram Schulte, Nikolai Tillmann, and Wolfgang Grieskamp. Optimal strategies for testing nondeterministic systems. In George S. Avrunin and Gregg Rothermel, editors, *ISSTA*, pages 55–64. ACM, 2004. (Cited on pages 25, 26, 67, and 78.)

[55] Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. Verification of timed automata via satisfiability checking. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2*, FTRTFT '02, pages 225–244, London, UK, 2002. Springer-Verlag. (Cited on page 114.)

[56] Alexandre Petrenko, Sergiy Boroday, and Roland Groz. Confirming configurations in EFSM testing. *IEEE Trans. Softw. Eng.*, 30(1):29–42, January 2004. (Cited on pages 24, 37, and 47.)

[57] Alexandre Petrenko, Nina Yevtushenko, and Jia Le Huo. Testing transition systems with input and output testers. In *Proceedings of the IFIP TC6/WG6.1 XV International Conference on Testing of Communicating Systems*, volume 2644 of *LNCS*, pages 129–145, Heidelberg, 2003. Springer. (Cited on page 43.)

[58] Testcast: a TTCN-3 test development and execution platform, 2012. http://www.elvior.com/testcast/introduction, Retrieved 1. Oct, 2012. (Cited on page 97.)

[59] Jan Tretmans. Model based testing with labelled transition systems. In Robert M. Hierons, Jonathan P. Bowen, and Mark Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008. (Cited on pages 25, 48, 50, 51, 53, 54, and 56.)

[60] Jan Tretmans and Ed Brinksma. Côte de Resyste: Automated model based testing. In M. Schweizer, editor, *3rd PROGRESS Workshop on Embedded Systems*, pages 246–255, Utrecht, 2002. STW Technology Foundation. (Cited on page 25.)

[61] Jan Tretmans and Ed Brinksma. TorX: Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, editors, *First European Conference on Model-Driven Software Engineering, Nuremberg, Germany*, pages 31–43, December 2003. (Cited on page 26.)

[62] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. (Cited on pages 21 and 24.)

[63] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012. (Cited on page 24.)

[64] Jüri Vain, Marko Kääramees, and Maili Markvardt. *Dependability and Computer Engineering : Concepts for Software-Intensive Systems*, chapter Online testing of nondeterministic systems with reactive planning tester, pages 113–150. IGI Global, Hershey, PA, 2011. (Cited on pages 53, 63, 79, 99, and 113.)

[65] Jüri Vain, Andres Kull, Marko Kääramees, Maili Markvardt, and Kullo Raiend. *Model-Based Testing for Embedded Systems*, chapter Reactive testing of nondeterministic systems by test purpose directed tester. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press - Taylor & Francis, 2011. (Cited on pages 27, 37, 53, 58, and 113.)

[66] Jüri Vain, Kullo Raiend, Andres Kull, and Juhan Ernits. Synthesis of test purpose directed reactive planning tester for nondeterministic systems. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 363 – 372. ACM Press, 2007. (Cited on pages 26, 27, 53, 58, 59, and 113.)

[67] Margus Veanes and Nikolaj Bjørner. Alternating simulation and IOCO. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(4):387–405, 2012. (Cited on pages 25, 50, 51, 54, and 56.)

[68] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson. Model-based testing of object-oriented reactive systems with Spec Explorer. In *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 39–76. Springer, 2008. (Cited on page 26.)

[69] Margus Veanes, Colin Campbell, and Wolfram Schulte. Composition of model programs. In John Derrick and Jüri Vain, editors,

FORTE, volume 4574 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2007. (Cited on pages 26 and 50.)

[70] Margus Veanes, Pritam Roy, and Colin Campbell. Online testing with reinforcement learning. In Klaus Havelund, Manuel Núñez, Grigore Rosu, and Burkhart Wolff, editors, *FATES/RV*, volume 4262 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2006. (Cited on pages 25 and 26.)

[71] Mihalis Yannakakis. Testing, optimizaton, and games. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 78–88. IEEE Computer Society, 2004. (Cited on pages 37 and 47.)

[72] Thaise Yano, Eliane Martins, and Fabiano L. de Sousa. MOST: A multi-objective search-based testing from EFSM. In *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '11, pages 164–173, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on pages 25 and 101.)

[73] Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman. *Model-Based Testing for Embedded Systems*. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press - Taylor & Francis, 2011. (Cited on page 24.)

Part II

PUBLICATIONS

Paper I

Jüri Vain, Andres Kull, Marko Kääramees, Maili Markvardt, and Kullo Raiend. *Model-Based Testing for Embedded Systems*, chapter Reactive Testing of Nondeterministic Systems by Test Purpose Directed Tester. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press - Taylor & Francis, 2011. http://dx.doi.org/10.1201/b11321-16

Paper II

Marko Kääramees, Jüri Vain, and Kullo Raiend. Synthesis of On-line Planning Tester for Non-deterministic EFSM models. In Leonardo Bottaci and Gordon Fraser, editors, *Testing - Practice and Research Techniques*, volume 6303 of LNCS, pages 147–154. Springer Berlin / Heidelberg, 2010. http://dx.doi.org/10.1007/978-3-642-15585-7_14

Paper III

Danel Ahman and Marko Kääramees. Constraint-based Heuristic Online Test Generation from Non-deterministic I/O EFSMs. In Alexander K. Petrenko and Holger Schlingloff, editors, Proceedings of 7th Workshop on Model-Based Testing, Tallinn, Estonia, volume 80 of EPTCS, pages 115–129, 2012. http://dx.doi.org/10.4204/EPTCS.80.9

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.

2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.

3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.

4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.

5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.

6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.

7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.

8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.

9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.

10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.

11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.

12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.

13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.

14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.

15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.

16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.

17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.

18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo.** Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components. 2009.

48. **Vineeth Govind**. DfT-Based External Test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.