

# Choice and Chance

*Model-Based Testing of Stochastic Behaviour*

Marcus Gerhold

**Graduation Committee:**

Chairman:

prof. dr. J. N. Kok

Promotors:

prof. dr. M. I. A. Stoelinga

prof. dr. J. C. van de Pol

Members:

dr. ir. H. G. Kerkhoff

University of Twente

prof. dr. N. V. Litvak

University of Twente

prof. dr. M. R. Mousavi

University of Leicester

prof. dr. J. Peleska

University of Bremen

dr. ir. G. J. Tretmans

Radboud University Nijmegen

**DIGITAL SOCIETY  
INSTITUTE**

**DSI Ph.D. Thesis Series No. 18-022**

Institute on Digital Society, University of Twente  
P.O. Box 217, 7500 AE Enschede, The Netherlands



**IPA Dissertation Series No. 2018-20**

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).



**Netherlands Organisation for Scientific Research**

The work in this thesis was supported by the BEAT project (BEtter testing with gAme Theory), funded by NWO grant 612.001.303.

ISBN: 978-90-365-4695-9

ISSN: 2589-7721 (DSI Ph.D. Thesis Series No. 18-022)

DOI: 10.3990/1.9789036546959

Available online at <https://doi.org/10.3990/1.9789036546959>

Typeset with L<sup>A</sup>T<sub>E</sub>X

Printed by Ipskamp Printing, Enschede

Cover design © 2018 by Shaun Hall

Copyright © 2018 Marcus Gerhold

CHOICE AND CHANCE  
MODEL-BASED TESTING OF STOCHASTIC BEHAVIOUR

DISSERTATION

to obtain  
the degree of doctor at the University of Twente,  
on the authority of the rector magnificus  
prof. dr. T. T. M. Palstra,  
on account of the decision of the graduation committee,  
to be publicly defended  
on Wednesday 12<sup>th</sup> of December 2018 at 16:45

by

Marcus Gerhold

born on 30<sup>th</sup> of September 1989  
in Hohenmölsen, Germany

This dissertation has been approved by:

Prof. dr. M. I. A. Stoelinga (promotor)

Prof. dr. J. C. van de Pol (promotor)

---

## Acknowledgements

---

This thesis marks the culmination of a journey that started four years ago with a leap of faith into another country, and an entirely new subject. A journey filled with the entire spectrum ranging from pure excitement and happiness, to uncertainties and doubts of whether my work was relevant at all. During this time I was never alone - I met fellow-journeymen, friends, and mentors, to some of whom I would like to reach out and express my gratitude.

Mariëlle, as my daily supervisor you were first in line to witness the progress of my life as a graduate student. I am thankful to you for always sharing your knowledge, but even more so, your time, no matter how busy you were. From you I learned the tools of the trade – performing research, bringing it down to paper, and conveying it to others. Winning the EASST best paper award at ETAPS in 2016 with our first conference paper showed the relevance of our research, and also the influence of your teaching coming to fruition within me. You taught me the power that simple yet clear language can have. Whenever I knew I did not try my very best, I could rely on you pushing me in the right direction – and I am glad you did. I am sure working with me was not always easy, but I want to express my sincere gratitude to you, for giving me the opportunity to grow as a researcher and as a person in the FMT group. The repercussions of your mentoring can be found in every line of this thesis.

Jaco, someone once told me “Whenever you enter Jaco’s office with a problem, you will leave without it.” After you were my supervisor for more than 4 years I can completely confirm this. In some of our meetings you remarked that you cannot contribute anything of relevance – I wholeheartedly disagree. Your unbiased view on my research topics alongside your excellent apprehension of complex yet unfamiliar topics provided me with new approaches more than once. You always knew the right steps to take for every problem that arose, and the right words to say to make solving it sound trivial. Without your motivating words and your guidance, the clear head needed to write a thesis would likely have been obscured with dark clouds brooding over nonsensical topics.

Arnd, in many ways you are not only the co-author of the papers we wrote together, but also my third supervisor. I sincerely thank you for proofreading my work, helping me solve problems encountered along the way, and the time you took to answer even the most trivial of my questions. Your keen perception enabled you to challenge concepts I took for granted, while being unfamiliar with the topics yourself. This gained me new insights and helped me to escape from

dead-ends I got stuck in. My productivity largely increased when we moved into the same office, and I was able to bounce ideas back and forth with you.

In addition to my supervisors, there are many other people without whom I surely would not be able to write an acknowledgements section in this thesis. First and foremost, I want to thank both Joke Lammerink and Ida den Hamer. I believe there were moments, in which I would have been more than figuratively lost, were it not for you always letting me know what to do next. I sincerely hope that my tendency to organise things rather late than soon did not cause you too many headaches.

I would like to thank my old office roommates Dennis Guck, Waheed Ahmad, and Enno Ruijters for making me feel welcome from the very beginning. Especially Enno, whom I must have bothered countless many times over the last years: Thank you for always taking time to patiently give insightful advice, may it be on mathematics, thesis writing, printers, presentations, or pesky bureaucracy. Rajesh, thank you for the many academic and non-academic conversations – Your boundless interest in every topic made you a wonderful well of inspiration to draw from whenever I wore my academic blinders.

Much appreciation is dedicated towards my new office roommates Tom van Dijk, Jeroen Meijer, Vincent Bloemen, David Huistra and Freark van der Berg. Sharing an office with you made working days so much more enjoyable. I cherish all of you joining in on the occasional non-work related banter and tirades more than you can imagine. It was not all fun and jokes however: Your collective brains were a wonderful encyclopedia for nearly every topic, and whenever I encountered even the tiniest issue, I knew I need but ask either of you. The productive and supporting, yet comfortable working atmosphere which you provided is not self-explanatory and I am grateful to every single one of you.

Of course, I do not want to forget all the other FMT members and FMT alumni with whom I never shared an office. You made the halls of FMT always very welcoming: Arend Rensink, Rom Langerak, Marieke Huisman, Ansgar Fehnker, Axel Belinfante, Gijs Kant, Lesley Wevers, Wytse Oortwijn, Sebastiaan Joosten, Güner Orhan, Buğra Yildiz, Carlos Budde, Mohsen Safari, and the many, many people I forgot to mention here. Stefano Schivo, your talents as screenplay writer and athlete are unparalleled. However, above all, the outcome of a weekend of your talent as a baker always made Mondays a pleasant surprise.

Thank you also to Angelika Mader, who let me help out in teaching Creative Technology students. I benefited greatly from occasionally encountering the academic world from a more playful side. I would also like to thank the many students of Testing Techniques – Preparing the course material each year coupled to your thorough questions made me learn a great deal of the topic I otherwise would have missed. In that way, I hope all of us profited from the course.

I would also like to express my gratitude towards the people that reminded me of the life outside of academia: My dear friends and housemates. Elena Lederer, Adrienn Bors, Moritz Arendt – moving in with you made me feel at home in Enschede for the first time ever since I moved to the Netherlands. Tamara Baas, Ksenija Kosel, Lennart Uffmann, Felix Moritz, Paul Gantzer, Kai Leistner, Nils Maurer, Kevin Wolf and Sara Szekeley, thank you all for making me

look forward to weekends and after-work hours. A special thank you is dedicated to my friends and housemates Greta Seuling, Jandia Melenk, Tim Möller and Karan Raju, who had to endure me during the last months of thesis writing.

My sincere apologies to my non-Enschede based friends. Being tangled up in work more often than I hoped for prevented me from being the good friend you all deserve. Thank you for never holding it against me, Benjamin Dixel, Matthias Gründig, Oliver Moisich, and Robert Wenzl. A particular thank you to my friend Andrew Cowie, whom I have not seen in seven years, but who kept me company on the internet during the days I worked overtime in the office. Shaun Hall, I thank you for lending your creativity to design the one part of this dissertation that is probably the only part that most people will see.

Letztlich möchte ich meiner Familie dafür danken mir so viel mit auf meinen Weg gegeben zu haben. Danke an meine älter Schwester Gina, die schon seit jeher auf mich Acht gibt. Ein ganz besonderer Dank gilt meinen Eltern Cornelia König und Uwe Gerhold, ohne deren ständige Unterstützung ich sicher nicht in der Lage wäre diese Zeilen zu schreiben.

Der wohl grösste Dank gebührt Jill – Dein offenes Ohr und deine endlose Geduld sind der Grund, warum ich mein Ziel nie aus den Augen verloren habe.

Münster  
November 2018





---

## Abstract

---

Probability plays an important role in many computer applications. A vast number of algorithms, protocols and computation methods uses randomisation to achieve their goals. A crucial question then becomes whether such *probabilistic* systems work as intended. To investigate this, such systems are often subjected to a large number of well-designed test cases, that compare the observed behaviour to a requirements specification. These tests are often created manually, and are thus prone to human errors. Another approach is to create these test cases automatically. Model-based testing is an innovative testing technique rooted in formal methods, that aims at automating this labour intense task. By providing faster and more thorough testing methods at lower cost, it has gained rapid popularity in industry and academia alike. Despite all, classic model-based testing methods are insufficient when dealing with inherently stochastic systems.

This thesis introduces a rigorous model-based testing framework, that is capable to automatically test such systems. We provide correctness verdicts for functional properties, discrete probability choices, and hard and soft real-time constraints. First, the model-based testing landscape is laid out, and related work is discussed. From there on out, the framework is constructed in a clear step-by-step manner. We instantiate a model-based testing framework from the literature to illustrate the interplay of its theoretical components like, e.g., a conformance relation, test cases, and test verdicts. This framework is then conservatively extended by introducing discrete probability choices to the specification language. A last step further extends this probabilistic framework by adding hard and soft real time constraints. Classic functional correctness verdicts are thus extended with goodness of fit methods known from statistics. Proofs of the framework's correctness are presented before its capabilities are exemplified by studying smaller scale case studies known from the literature.

The framework reconciles non-deterministic and probabilistic choices in a fully-fledged way via the use of schedulers. Schedulers then become a subject worth studying on their own. This is done in the second part of this thesis: We introduce an equivalence relation based on schedulers for Markov automata, and compare its distinguishing power to notions of trace distributions and bisimulation relations. Lastly, the power of different scheduler classes for stochastic automata is investigated: We compare reachability probabilities of schedulers belonging to such different classes by altering the information available to them. This induces a hierarchy of scheduler classes, which we illustrate alongside simple examples.



---

## Table of Contents

---

<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>ix</b>
<b>Table of Contents</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The Formal Methods Approach . . . . .	3
1.2 Verification and Validation . . . . .	5
1.3 Testing and Properties of Interest . . . . .	6
1.4 Modelling Formalisms and Contributions . . . . .	8
1.5 Structure and Synopsis of this Thesis . . . . .	11
<b>2 The Model-Based Testing Landscape</b>	<b>15</b>
2.1 Overview . . . . .	16
2.2 Components of Model-Based Testing . . . . .	17
2.3 A Taxonomy of Model-Based Testing . . . . .	20
2.4 Classification of the Probabilistic Framework . . . . .	24
<b>3 Model-Based Testing in the ioco Framework</b>	<b>27</b>
3.1 Model and Language-Theoretic Concepts . . . . .	28
3.2 The Conformance Relation $\sqsubseteq_{ioco}$ . . . . .	33
3.3 Testing and Test Verdicts . . . . .	35
3.4 Correctness of the Framework . . . . .	39
3.5 Algorithms and Algorithmic Correctness . . . . .	40
3.6 Summary and Discussion . . . . .	43
<b>4 Model-Based Testing with Probabilistic Automata</b>	<b>45</b>
4.1 Model and Language-Theoretic Concepts . . . . .	48
4.1.1 Probabilistic Input Output Transition Systems . . . . .	48
4.1.2 Paths and Traces . . . . .	52
4.1.3 Schedulers and Trace Distributions . . . . .	53
4.2 Probabilistic Testing Theory . . . . .	58
4.2.1 The Conformance Relation $\sqsubseteq_{pioco}$ . . . . .	59
4.2.2 Test Cases and Test Annotations . . . . .	62

4.2.3	Test Evaluation and Verdicts . . . . .	64
4.2.4	Correctness of the Framework . . . . .	69
4.3	Implementing Probabilistic Testing . . . . .	71
4.3.1	Test Generation Algorithms . . . . .	72
4.3.2	Goodness of Fit . . . . .	74
4.3.3	Probabilistic Test Algorithm Outline . . . . .	77
4.4	Experiments . . . . .	78
4.4.1	Dice programs by Knuth and Yao . . . . .	79
4.4.2	The Binary Exponential Backoff Algorithm . . . . .	81
4.4.3	The FireWire Root Contention Protocol . . . . .	82
4.5	Summary and Discussion . . . . .	84
4.6	Proofs . . . . .	85
<b>5</b>	<b>Model-Based Testing with Markov Automata</b>	<b>97</b>
5.1	Input Output Markov Automata . . . . .	100
5.1.1	Definition . . . . .	101
5.1.2	Abstract Paths and Abstract Traces . . . . .	106
5.1.3	Schedulers and Trace Distributions . . . . .	108
5.2	Markovian Test Theory . . . . .	111
5.2.1	The Conformance relation $\sqsubseteq_{Mar-ioco}$ . . . . .	111
5.2.2	Test Cases and Annotations . . . . .	113
5.2.3	Test Evaluation and Verdicts . . . . .	115
5.2.4	Correctness of the Framework . . . . .	119
5.3	Implementing Markovian Testing . . . . .	122
5.3.1	Goodness of Fit . . . . .	123
5.3.2	Stochastic Delay and Quiescence . . . . .	127
5.3.3	Markovian Test Algorithm Outline . . . . .	129
5.4	Experiments on the Bluetooth Device Discovery Protocol . . . . .	129
5.5	Conclusions . . . . .	134
5.6	Proofs . . . . .	135
<b>6</b>	<b>Stoic Trace Semantics for Markov Automata</b>	<b>143</b>
6.1	Markov Automata . . . . .	145
6.1.1	Definition and Notation . . . . .	145
6.1.2	Language Theoretic Concepts . . . . .	146
6.1.3	Stoic Trace Semantics . . . . .	148
6.1.4	Compositionality . . . . .	151
6.2	A Testing Scenario . . . . .	152
6.2.1	Sampling and Expectations . . . . .	152
6.2.2	Observational Equivalence . . . . .	155
6.3	Relation to other Equivalences . . . . .	156
6.3.1	Trace Distribution Equivalence by Baier et al. . . . .	156
6.3.2	Bisimulation . . . . .	158
6.3.3	Hierarchy . . . . .	162
6.4	Conclusions . . . . .	163
6.5	Proofs . . . . .	163

<b>7</b>	<b>Model-Based Testing with Stochastic Automata</b>	<b>169</b>
7.1	Stochastic Automata . . . . .	171
7.1.1	Definition . . . . .	171
7.1.2	Language Theoretic Concepts . . . . .	175
7.1.3	Schedulers and Trace Distributions . . . . .	177
7.2	Stochastic Testing Theory . . . . .	179
7.2.1	The Conformance Relation $\sqsubseteq_{ioco}^{sa}$ . . . . .	179
7.2.2	Test Cases . . . . .	181
7.2.3	Test Execution and Sampling . . . . .	183
7.2.4	Correctness of the Framework . . . . .	185
7.3	Implementing Stochastic Testing . . . . .	188
7.3.1	Goodness of Fit . . . . .	188
7.3.2	Algorithmic Outline . . . . .	192
7.4	Bluetooth Device Discovery Revisited . . . . .	192
7.5	Conclusions . . . . .	196
7.6	Proofs . . . . .	197
<b>8</b>	<b>Scheduler Hierarchy for Stochastic Automata</b>	<b>203</b>
8.1	Preliminaries . . . . .	206
8.1.1	Closed Stochastic Automata . . . . .	207
8.1.2	Timed Probabilistic Transition Systems . . . . .	208
8.1.3	Semantics of Closed Stochastic Automata . . . . .	209
8.2	Classes of Schedulers . . . . .	211
8.2.1	Classic Schedulers . . . . .	211
8.2.2	Non-Prophetic Schedulers . . . . .	212
8.3	The Power of Schedulers . . . . .	213
8.3.1	The Classic Hierarchy . . . . .	214
8.3.2	The Non-Prophetic Hierarchy . . . . .	219
8.4	Experiments . . . . .	220
8.5	Conclusions . . . . .	222
<b>9</b>	<b>Conclusions</b>	<b>225</b>
9.1	Summary . . . . .	225
9.2	Discussion and Future Work . . . . .	228
	<b>Appendices</b>	<b>229</b>
<b>A</b>	<b>Mathematical Background</b>	<b>231</b>
A.1	Probability Theory . . . . .	231
A.2	Statistical Hypothesis Testing . . . . .	234
A.2.1	Statistical errors. . . . .	235
A.2.2	Two Types of Hypotheses Tests . . . . .	236
A.2.3	Pearson's $\chi^2$ Test . . . . .	238
A.2.4	Kolmogorov-Smirnov Test . . . . .	239
A.2.5	Accumulation of Type I Errors . . . . .	242

<b>B Publications by the Author</b>	<b>245</b>
<b>Bibliography</b>	<b>247</b>
<b>Samenvatting</b>	<b>263</b>

# CHAPTER 1

---

## Introduction

---

On May 3rd in 1997 many of the world's eyes were focussed on an unusual competition. It was chess world grandmaster Garry Kasparov's third match versus IBM's newest iteration of their dedicated chess computer Deep Blue. After the grandmaster handily won the first two proposed matches in 1989, and 1996, IBM improved the hardware of their computer alongside its routines [37], and challenged the uncontested chess world champion again. At that time, the six games long match was conceived as more than a mere chess competition – with the most recent developments in computing and artificial intelligence, it was considered as the representative match of human versus machine [81]. Throughout history, chess was conceived as a measure of intelligence, as it combines complex mathematical and combinatorial decisions, long term strategic planning, and human creativity. Defeating the renowned chess champion solely using immense computing power and refined programming routines thus represented the latest advancements in information technology, and would let the world have a glimpse at the true impending potential of computers.

The challenge consisted of six games to be played on consecutive days. Expert analysts agreed that Kasparov started off the first game strong and decisive [155]. However, towards the end of the game something unusual occurred in the behaviour of Deep Blue's chess moves. While both contestants were roughly at eye level, Deep Blue decided to move one of its rooks to a position where it effectively achieved nothing, neither offensively, defensively or in terms of positional play, cf. Figure 1.1. Understandably, Kasparov was baffled by the loss of posture by his machinery opponent [155]. Even though he was the uncontested champion of chess, capable of thinking 10, or even 15 turns ahead, he then faced simulated mental capabilities beyond his understanding.

Kasparov continued the game and drew a concession from Deep Blue in the very next turn, but the latest move left an impression on the Russian chess player [115]. The formidable 1:0 was quickly followed by Deep Blue equalising to 1:1, after a premature concession by Kasparov. Hindsight analysis showed, had Kasparov played the rest of the game perfectly, he would have been able to force a draw [155]. Games three, four and five ended in draws, before the

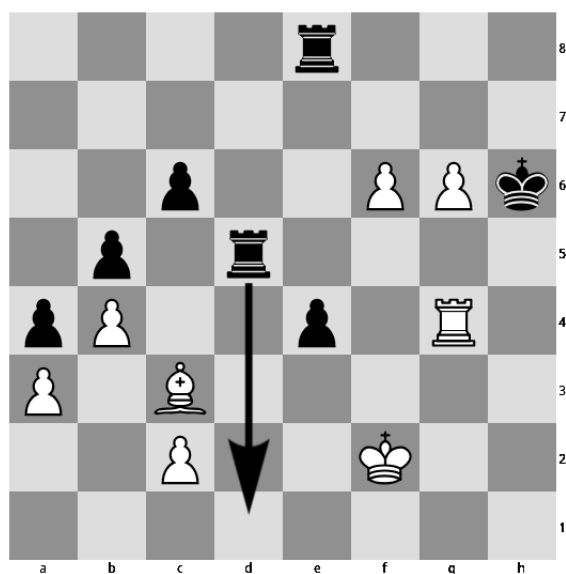


Figure 1.1: Deep Blue’s (black) perplexing rook move from D5 to D1 as a result of a fail safe to exit an infinite loop, yielding a *randomly chosen legal move* [37]. The move results in Deep Blue’s concession in the subsequent turn.

surprising result of game six ended in the match win in favour of Deep Blue.

IBM engineers later found that the baffling rook move performed by Deep Blue in game one, was the outcome of a *fail-safe* built into its programming routines. Deep Blue was stuck in an infinite computation loop – the fail-safe was to perform a *random legal move* to exit. Evidently, there is no assurance whether the overall match win resulted from the strength of Deep Blue’s play, the mental strains on Kasparov’s side as 6 games were played in only few consecutive days, or the overall confusion and perhaps intimidation of facing a superior opponent.

“I’m a human being. When I see something that is well beyond my understanding, I’m afraid.” – Garry Kasparov [193]

The 1997 chess match showed the latest advancements in artificial intelligence, and there is no doubt today, that chess programs rival even the best of their human opponents. While chess allows for seemingly endless possible placements of its pieces on the board, in 2016 AlphaGo [78] for the first time beats South Korean champion Lee Sedol 4:1 in a match of Go [26] – a game of even greater complexity, where pieces are placed on a  $32 \times 32$  square grid with almost no restrictions. As seen by the fail safe performed by Deep Blue, the immense increase of computing power comes at a price: Who is to ensure the *correctness* of a program capable of beating humans in their own game?

Evidently, the problem at hand becomes vastly more grave, if we turn away from *games* to more worldly matters: We live in a world that is almost entirely



percolated by computers, and artificial intelligence aids us in our everyday lives. Consider a simple trip to the airport: Semi-automatic security scanners check the cabin luggage of every passenger, autonomously controlled monorails transport a substantial amount of people to their respective gates, before they enter an aircraft almost exclusively operated by fly-by-wire technology. Autonomous vehicles are not limited to railway systems anymore – the most recent advances suggest that self-driven cars are a widespread realistic mode of transportation of the near future [104]. Even modern healthcare relies on the usage of computer aided methods like the semi-automatic Da Vinci robotic surgical systems for minimally invasive surgeries [192], or dedicated sleep tracking smartphone applications that promise to increase the users well-being [14].

Unfortunately, akin to Deep Blue’s “indifference” for committing a losing move to the chess board, these dedicated algorithms may cause catastrophic losses of lives and aircraft [108], unaccounted for and dangerous behaviour of vehicles in road traffic [80], as well as over-dosage of patients with mortal results [127]. This is to illustrate, that a world, in which nuclear power plants are operated by computer networks demand a *pendent* of the ingenuity in information technology to study their performance, safety, and reliability.

This thesis develops rigorous techniques rooted in mathematics and formal methods, that provide confidence of a system’s correctness. A heavy focus lies on probabilistic systems – systems using algorithms that intrinsically rely on the outcome of probabilistic choices to achieve their goals. In particular, a model-based testing framework for such systems is developed and studied.

## 1.1 The Formal Methods Approach

To counteract uncertainty in the ever-growing advances in information technology, the field of formal methods [44] seeks to develop mathematically rigorous techniques to ensure their safety and reliability. The application of formal methods relies on studying, designing and analysing complex systems based on mathematically profound and unambiguous *models*. This both aids in rationally quantifying the results and findings one encounters upon studying a system, and to go about it in a rigorously structured manner. A model gives engineers the advantage of working in a unifying framework in which inaccuracies can be quickly pointed out, and in which the engineering team shares a common language.

The ingredients of formal methods are presented in Figure 1.2. At their core, formal methods comprise:

- A **design/idea** as the origin of a system to be developed, or maintained,
- an unambiguous **model** describing the behaviour of the system in a mathematically rigorous way,
- a set of **requirements** explicitly describing desired behaviour, and
- a physical **implementation** in the real world.

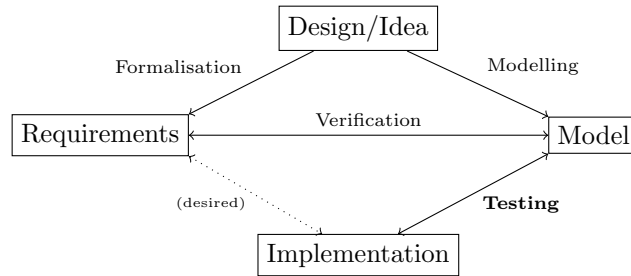


Figure 1.2: Illustration of the formal methods approach after [90]. The interplay of an implementation to its requirements model, i.e. *testing*, comprises the focus of this thesis.

While the nodes in Figure 1.2 illustrate the components of the formal methods approach, the labelled arcs represent various disciplines used therein.

**Formalisation** entails the design step of transforming informal requirements given in a natural language, to an unambiguous description. Upon the development of a system, engineers frequently face ambiguous informal description of stakeholders, e.g. “fast response time is desired for small files”. The formalisation of requirements translate these into explicit statements, such as “A response time of less than 500ms for files smaller than 1MB”. Studies suggest that formalisation in itself prevents propagation of misconceptions early on in the design phase [130, 129], resulting in far less expensive mistakes to be resolved later on [112, 20].

**Modelling** describes the translation of conceptual behaviour phrased in human language, to a mathematical model, e.g. finite state machines (FSM) [25]. The choice of the modelling formalisms depends on the properties of interest (e.g. time, workload etc.), as well as the desired level of abstraction. The process of structurally modelling a system *design* has been shown to be of equal advantage as formalisation with respect to early error prevention [27, 129, 130]

**Verification** comprises techniques to study if a model adheres to a given set of formal requirements. Both the requirements and the model are given on a mathematical level, making verification a problem of algorithmic nature. Depending on the modelling formalism, a *model checker* (e.g. LTSmin [109], PRISM [118], MODEST [22]) receives queries, and continues to explore the state space of a model, to answer if the query is a *true statement*. Other techniques involve static verification [19], or theorem proving [144].

**Testing** encompasses the *validation* of an implementation with respect to a model. It is the interface between real world artefacts and mathematical models. Testing therefore aids in gaining confidence that an actual implementation is correctly reflected by a formal model. It is thus utilized to ensure certain behaviour is realised or avoided in the implementation.

While the *formalisation* of requirements, and *modelling* are crucial to the formal methods approach, they ultimately are a means for *verification* and *validation*. Therefore, the most prevalent advances in research can be observed in verification and testing, due to their direct impact and relevance for today's society.

This thesis focusses on *testing* of probabilistic and stochastic systems.

## 1.2 Verification and Validation

*Verification* and *validation* describe independent techniques in system development, but are frequently used in tandem. The purpose of their application is to gain confidence that a physical system comprises the behaviour encompassed in its original design concept.

**Verification** ensures, that a *model* adheres to given constraints and requirements. Although tool support for *theorem proving* exists, key steps in the procedure remain a manual task requiring human expertise. Hence, tools are frequently referred to as *proof assistants* [12]. In contrast, model checking [9] follows a more streamlined *push-button* approach in that tools are often fully automated. A model checker is provided with a query, i.e. a property of interest, and proceeds to check whether or not the property holds. Those include, but are not limited to, 1. *liveness* properties of concurrent systems, e.g. ensuring that a *deadlock* can never be reached, 2. *reachability* properties, stating that a certain set of goal states is always reachable, and 3. *safety* properties, ensuring that a set of undesirable states can never be reached. All properties may be augmented with aspects like time or probability, e.g. reaching a set of goal states within a certain time, with a minimal probability. A large proportion of research in verification is dedicated towards the avoidance of a *state space explosion* via increasingly more sophisticated algorithms and techniques. That is, a desirable goal is the avoidance to *exhaustively* search all possible system configurations to ensure a property of interest does, or does not hold. As an illustration one may think back to the initial example of a chess board, where the total number of combinations of chess pieces on a board is larger than the number of atoms in the universe. Certainly, checking *all* of them cannot be considered a feasible goal, and sophisticated *state space exploration* algorithms are needed.

**Validation** is the most commonly applied approach in practice to evaluate and certify systems outside of a limited area of highly safety-critical applications. In contrast to verification, validation provides a direct link between an actual implementation and its model. Even medium-sized software development companies deploy testing in a routinely manner, larger-sized companies have a dedicated team of test engineers, and companies that focus only on testing offer their services. However, testing is a time- and money intense task often taking up to 50% of a project's budget [140]. Even though dedicated test companies exist, testing is frequently done manually hinting at the susceptibility and proneness for human errors. Both of these facts demand advances of testing techniques

making 1. *structured testing* feasible for further widespread application, and 2. testing *more effective*, implying that the same budget and time investment, yields more beneficial results for practitioners.

**Model-Based Testing** is an innovative validation technique rooted in formal methods [149, 157], that is developed to both make testing more structured and more efficient. It comprises techniques to automatically *generate*, *execute*, and *evaluate* test cases. This three step approach in combination with the use of a formal model ensures that a real physical implementation can be tested in a mathematically rigorous way. The model encompasses the unambiguously specified behaviour that a system is desired to exhibit. A test *generation* tool (or model-based testing tool) then derives concrete test cases from this specification model, *executes* them on the real implementation, and *evaluates* them by comparing the outcome to the required behaviour. In this way, the error-proneness is transferred from the creation of concrete test cases, to the generation of a formal model. Testing based on a model thus becomes much more streamlined, and incarnates the same *push button* approach exhibited by model-checking. By providing faster and more thorough testing at lower cost, model-based testing has gained rapid popularity in industry [188, 86, 102].

To avoid susceptibility to imprecision, and to enable automation, the work contained in this thesis is concerned with developing novel techniques in the field of *model-based testing*. In particular, the focus of the developed methods lies in probabilistic systems. A formal introduction to the field is given in Chapter 2.

### 1.3 Testing and Properties of Interest

At their very core, every formal method technique relies on models – abstractions from superfluous details, that allow practitioners to focus on, understand and study properties of interest. The wide variety of properties of interest, hence requires a *pendent* in the field of formal methods. To illustrate some: An automated teller system is considered infeasible, if it provides banknotes without prior credential checks, online trading systems that do not realise transactions in certain time constraints are impractical, medical equipment that cannot guarantee a patient’s safety is rightfully regarded as perilous, and a smart phone that empties its battery within an hour of usage is of no use to clients.

**Probability.** To that end, probability plays an increasingly important role in many computer applications, and naturally adds to the properties of interest. A vast number of randomized algorithms, protocols and computation methods use randomization to achieve their goals. Routing in sensor networks, for instance, can be done via random walks [4]; speech recognition is based on hidden Markov models [154]; population genetics use Bayesian computation [13], security protocols use random bits in their encryption methods [41]; control policies in robotics, leading to the emerging field of probabilistic robotics [168], are

concerned with perception and control in the face of uncertainty, and networking algorithms assign bandwidth in a random fashion. More abstractly, service level agreements are formulated in a stochastic fashion, stating that the average uptime should be at least 99%, or that the punctuality of train services should be 95%. The key question whether such systems are correct remains; Is bandwidth distributed fairly among all parties? Is the up-time, packet delay and jitter according to specification? Do the trains on a certain day run punctual enough?

**Related Work.** To investigate the vast variety of properties, model-based testing has matured from its roots in process theory [56] to a wide-ranging research field: functional behaviour of an implementation can automatically be tested by modelling interactions with the system via inputs and outputs, for example with finite state machines [124, 179]. Labelled transition systems [173] additionally cater for today's highly concurrent and cyberphysical systems, by allowing non-determinism and underspecification. To test timing requirements, such as deadlines, a number of timed model-based testing frameworks have been developed [120, 29].

However, a surprisingly small amount of research is dedicated to the testing of probabilistic systems, i.e. systems relying on algorithms that inherently make use of probabilities to achieve their goals. While *verification* of such systems is a well-studied field, putting forth models like probabilistic automata [158], interactive Markov chains [93], or (generalized) stochastic Petri nets [132], and tool support provided by stochastic model checkers like PRISM [118] or Storm [58], only a handful of applicable model-based testing frameworks using probabilities exist. Probabilistic finite state machines are studied in [96, 133], and come with the benefits and caveats of the finite state machine formalism. A *black-box* approach to analyse systems against specifications based on statistics is given in [159]. The approach assumes no interaction with the system is possible – A critical feature, considering that real implementations are frequently exposed to uncertain environments or human agents. Notable work is given by Hierons et al. [98], modelling systems that have physically distributed interfaces, thus causing non-determinism. However, non-determinism is instantiated probabilistically, rather than probabilistic choices being the quantities of interest in the first place. The work by [138, 97] is concerned with stochastic finite state machines that specify soft real-time constraints. Another line of work that uses probabilities is given in *model-based statistical testing* [150, 190]. Here, the behaviour of the tester is modelled, and input sequences are assigned probabilities to maximise the likelihood to achieve certain goals.

All presented frameworks are highly specialised in their respective applications. However, where probabilistic decisions of systems are studied, interaction with their environment is assumed to be minimal, or even non-existent. In particular, the interplay of *non-determinism* and *probabilistic choices* seems to be a challenging one. The work presented in this thesis seeks to take on this challenge, and to establish a unifying framework for *non-determinism*, *discrete probability choices*, and *stochastic time delays*. Specifically:

**Non-determinism** represents the *unquantified* choice between two or more alternative behaviours. A non-deterministic choice is absent of any information about the frequency associated to certain behaviour, as well as the precise influences that determine its outcome. It is utilized 1. to model the *unknown* influence of a system's environment, 2. to allow implementation freedom, 3. to model choices by human agents, or simply 4. as the true absence of knowledge on behalf of the modeller regarding the outcome of choices. Non-determinism is the crucial feature of labelled transition systems (LTS), the fundamental model on which the conceptual framework of this thesis builds upon.

**Functional** behaviour describes precisely the actions and allowed sequences of such actions a system can perform. These may, or may not be visible to an external observer and are frequently referred to as a system's language. The description of functional behaviour is used to 1. allow/enable, or disallow/disable certain behaviour of a system, or 2. enable interaction of multiple modelled components via their parallel composition by characterising certain actions they share.

**Probability** is used to *quantify* the frequency of choices made by the system. Probabilistic choices explicitly describe the outcome of a choice by assigning probabilities to the various alternatives. It is utilized 1. to model the *uncontrollable* actions of a system, or the quantified influence of its environment, or 2. to model the deliberate use of probabilities in various algorithms, e.g. leader election protocols [165]. Probability can be modelled *discretely*, where the outcome of a probabilistic choice is akin to the tossing of a coin, or the roll of a die, or *continuously* where the outcome of a probabilistic choice may for instance be any real number in the interval  $[0, 1]$ . For the remainder of this thesis, we refer to discrete probability choices as *probabilistic*, and to continuous probability choices as *stochastic*.

**Time** describes time constraints in which certain behaviour is expected or allowed. Time constraints are for instance utilized when 1. time is of critical nature, and needs to be accounted for, e.g. extending the wheels of an aircraft and initiate breaking manoeuvres prior to landing, 2. communication with other components may be delayed, and allowed waiting time needs to be quantified, or 3. when studying performance and response rates within a network. Like probability, time can be modelled *discretely* via countable clock ticks, each representing the passage of a singular time unit, or *continuously* via a mechanism akin to a stopwatch.

## 1.4 Modelling Formalisms and Contributions

To provide the reader with a rough overview, we shortly introduce the modelling formalisms used to construct our framework. A formal treatment of the models, as well as individually related work follows in subsequent chapters. We provide a

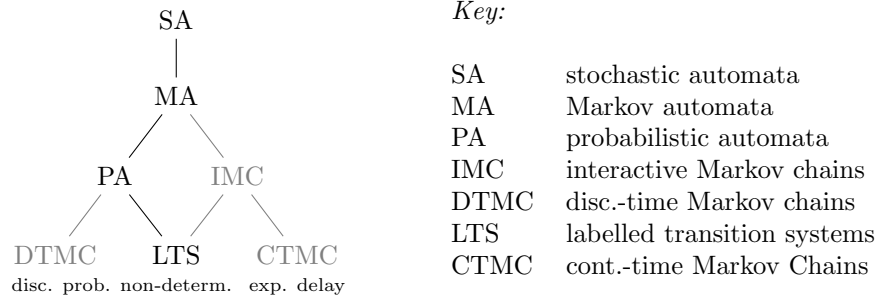


Figure 1.3: Automata modelling-formalisms used in this thesis.

brief overview over the capabilities of the various models to show their coverage of the properties of interest.

**Labelled transition systems** [174] encompass non-deterministic choices and communication among multiple (sub-)systems. States model the configuration of the entire system, while transitions identified by source and target states alongside a label represent the transfer of one system state to another. System events, or actions a user can perform are modelled via a separation of the action alphabet into inputs and outputs.

**Probabilistic Automata** [158] extend labelled transition systems by modifying the *target* of a transition. Instead of a single target, a transition may have multiple targets. The probability to reach a certain state is then quantified by a discrete probability distribution over all its targets. Unlike discrete-time Markov chains, a probabilistic automaton is additionally capable of performing non-deterministic choices. Hence, instead of non-deterministic choices between transitions, a probabilistic automaton has the potential of non-deterministic choices between *distributions*.

**Markov Automata** [67] extend probabilistic automata by adding another type of transition between states. A transition is either *probabilistic*, or *Markovian*. While the first is equivalent to the transitions possible in probabilistic automata, the latter now models stochastic time delay when going from one state to another. The delay of a Markovian transition is associated to a positive real-valued number, representing the parameter of an exponential distribution.

**Stochastic automata** [50] extend Markov automata, by allowing general distributions of time, as opposed to being limited to exponential distributions only. Additionally, transition may now be *guarded* by clock constraints, indicating the passage of time before they become enabled.

These models are by no means novel, but already see wide spread application in academia and industry. Labelled transition systems [169, 174] present a solid

choice of models for concurrent programs, due to their use of non-determinism. Underspecification and implementation freedom makes them ideal for testing, and they have successfully been applied in e.g. testing a Dutch storm surge barrier [177], or electronic passports [136]. Probabilistic automata [158] are a foundational model for stochastic verification, and have seen application in the verification of networking- [165], or security protocols [163]. Markov automata form the semantic foundation of fault- and attack trees [152] and the standardised modelling language AADL [28] analysis. Their exponential delays are an accurate approximation of the true *unknown* delay, if only the average of an activity is known. Lastly, stochastic automata allow for verification of real-time systems in which the time constraints are of purely random nature [90, 51, 50].

The work presented in this thesis follows the hierarchical structure presented in Figure 1.3. First, we recall testing theory for labelled transition systems – The work that this thesis is fundamentally rooted in, and motivate our choice. We then extend the framework by allowing discrete probability choices in the specification model. Lastly, stochastically delayed time is added, and an intermediate step towards Markov automata is made. The testing framework for stochastic automata *formally* supersedes the previous ones, but rather than presenting the final result up front, the work is presented in a step-by-step approach. This is done in an attempt to gradually familiarise the reader with individual components. Hence, a similar structure for these chapters is to be expected.

## Main Results

The main results of this thesis can be summarised as follows

- A mathematically rigorous model-based testing framework based on probabilistic automata is established in Chapter 4. Conformance, test cases, test executions and test verdicts are formally defined, and the framework is proven to be correct, i.e. *sound* and *complete*. Small case studies known from the literature are performed.
- The framework is enhanced by allowing exponentially distributed time delays in specification models in Chapter 5. A case study is performed on the Bluetooth device discovery protocol [161].
- Trace distribution semantics for Markov automata are developed in Chapter 6, that in particular equip *schedulers* with the power to *wait* before scheduling. The power of the semantics is compared with respect to similar approaches and bisimulation.
- General stochastic time delays are added to the model-based testing framework in Chapter 7. We present how practical application shifts from frequency analysis, to additional statistical hypothesis tests to account for time delays.
- A hierarchy for scheduler classes of stochastic automata is established with respect to reachability probabilities. This includes the classic full information view schedulers [30], as well as non-prophetic schedulers [91].



## 1.5 Structure and Synopsis of this Thesis

The nine chapters of the thesis are meant to be read sequentially with each chapter building on its predecessor. For the convenience of the reader, each chapter makes prerequisite knowledge explicit by providing references to earlier occurrences of the material, thus providing an alternative reading approach. While the central theme of the thesis is to establish a unifying model-based testing framework, Chapters 6 and 8 may be read individually. Figure 1.4 provides an overview for a suggested reading flow.

We point out that much of the technical background necessary to understand the work is provided within each chapter, and the thesis seeks to be self-sufficient. However, to maintain overall readability, we provide appendices covering general mathematical preliminaries, probability theory, and statistical hypothesis testing. These are by no means proper introductions to their respective fields, but rather a refresher for the dear reader. Further references to external reading material are given whenever appropriate. In an effort to maintain readability of the text, the mathematical proofs of our theorems are appended to the end of each respective chapter. They may be skipped depending on the scrutiny and interest of the reader.

We briefly summarize the contribution of each chapter:

**Chapter 2** serves as a starting point, and provides a brief overview of (model-based) testing, and the model-based testing (MBT) landscape. Core concepts of MBT are introduced, and a schematic is presented that each established framework in later chapters can fall back on. The taxonomy of MBT by [182] aids us in placing the presented thesis in the context of related work. This chapter is limited to results known from the literature and secondary sources.

**Chapter 3** provides an exemplary MBT framework in the testing theory for labelled transition systems with **ioco** [174], filling in the previously introduced schematic of Chapter 2. Moreover, the presented framework serves as foundation of our own work. This chapter is loosely based on secondary sources [169, 174].

**Chapter 4** introduces the MBT framework for systems specifying probabilities. We recall probabilistic automata, which are used as underlying specification formalism. We present a notion of conformance tailored for probabilistic automata, and show how it conservatively extends the existing theory of Chapter 3. A formal definition of test cases is given, alongside two algorithms that derive them in batch or on-the-fly, before the framework is proven to be correct. The framework is tested on three small-scale case studies.

This chapters' contribution is based on the publications

- Marcus Gerhold and Mariëlle Stoelinga. **ioco** theory for probabilistic automata. In *Proceedings of the 10th Workshop on Model Based Testing, MBT*, pages 23–40, 2015,

- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE*, pages 251–268, 2016,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. *Formal Aspects of Computing*, 30(1):77–106, 2018.

**Chapter 5** extends the MBT framework of Chapter 4 by incorporating stochastic time delays in the form of exponentially delayed transitions. Markov automata are used as underlying formalism to show how tests are generated and executed. We discuss the quiescence observation in the presence of stochastic time delays, and illustrate the framework on a small-scale case study.

This chapters' contribution is based on the publications

- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of stochastic systems with ioco theory. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, A-TEST*, pages 45–51, 2016,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems with stochastic time. In *Proceedings of the 11th International Conference on Tests and Proofs, TAP*, pages 77–97, 2017.

**Chapter 6** establishes trace semantics for Markov automata, incorporating the new notion of *waiting schedulers*. The newly introduced trace semantics are compared to existing ones in the literature, and to several notions of bisimulation relations. We illustrate our findings in a hierarchical overview summarising all considered equivalences, and show implications and strictness.

This chapters' contribution is based on work performed between May 2015 and October 2017 in collaboration with Dennis Guck, Holger Hermanns, Jan Krčál and Mariëlle Stoelinga.

**Chapter 7** culminates the MBT frameworks' capabilities by extending the previously established methods with general stochastic time delays on transitions. Stochastic automata are used as underlying formalism, and benefits and caveats are discussed when our methods are applied in continuous real-time.

This chapters' contribution is based on the publication

- Marcus Gerhold, Arnd Hartmanns, and Mariëlle Stoelinga. Model-based testing for general stochastic time. In *Proceedings of the 10th International Symposium on NASA Formal Methods, NFM*, pages 203–219, 2018.

**Chapter 8** studies schedulers of stochastic automata in their own rights. In particular, a hierarchy of classes of schedulers is established. This is done for classical notions of schedulers, as well as non-prophetic ones. The metric of

choice are unbounded reachability probabilities. The hierarchy is proven via intuitive examples and easy-to-follow proofs. The power of scheduler classes is illustrated via lightweight scheduler sampling.

This chapters' contribution is based on the publication

- Pedro R. D'Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. A hierarchy of scheduler classes for stochastic automata. In *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FOSSACS*, pages 384–402, 2018.

**Chapter 9** summarizes the thesis, presents overall conclusions, and provides a discussion on our work. Additionally, we present some ideas for future work.

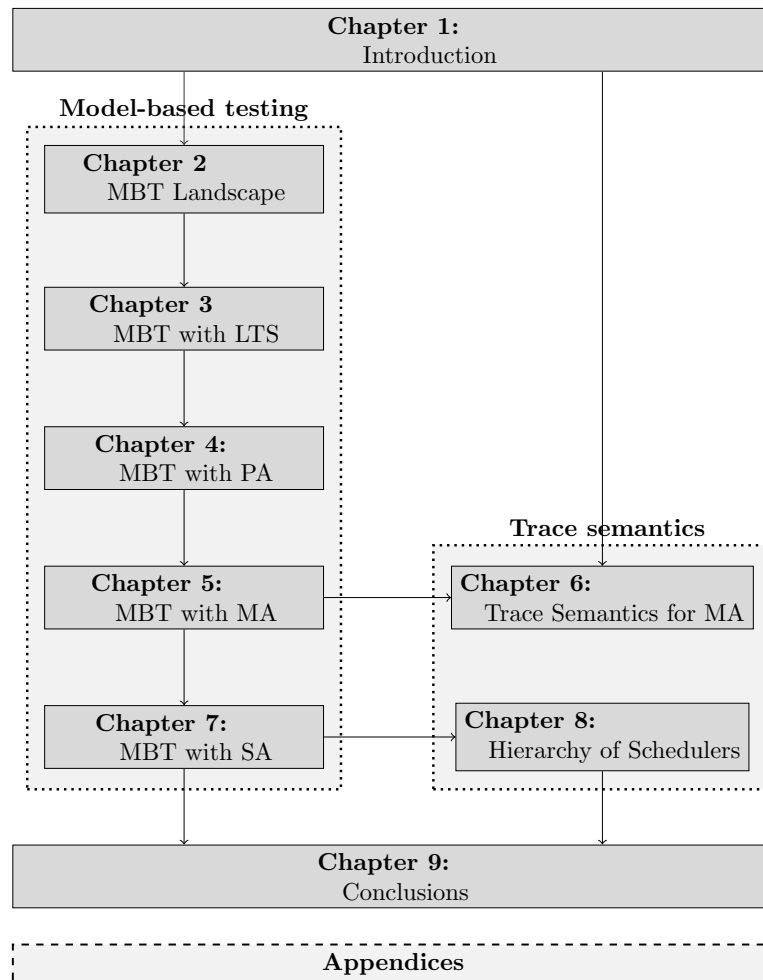


Figure 1.4: Chapter roadmap for suggested reading flow.



## CHAPTER 2

---

### The Model-Based Testing Landscape

---

Testing aims at showing that intended and exhibited behaviour of a system differ, or at gaining confidence, that they do not [194, 53]. Among its main purposes is the detection of failures, i.e. noticeable differences between system requirements and actual observed behaviour. The detection of failures may support future endeavours of developers and programmers in debugging or fault localisation, but it is also used as metric for quality.

It is applicable in various phases of the software development life cycle. A prominent approach is given by the *V-model*, cf. Figure 2.1. Each phase of the software design step has a mirroring component in a parallel testing step. The V-model adds an additional layer of quality assurance to the software development life cycle, thus gaining an edge in comparison to the *waterfall-model*. It prevents the propagation of errors to the lower design levels, where problem solving is more costly and more difficult, via the early use of requirements specifications.

Model-based testing (MBT) is one variant of testing, that aims at automating this otherwise labour intense, and often error prone validation technique. Its origins can be traced back to the seventies [42], but it has gained rapid popularity

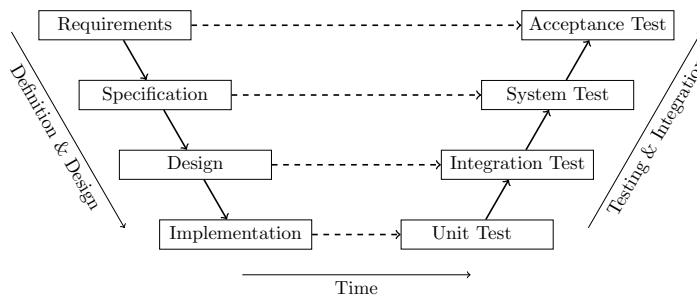


Figure 2.1: The *V-model* used in the software development life cycle. Contrary to the *waterfall-model* comprising a top-to-bottom approach, each phase in the V-model has a corresponding test phase, resulting in the eponymous shape.

in recent years with industry deploying MBT techniques [188, 86, 102]. The benefits of automation are evident for both producers and consumers: Higher quality products are delivered more cost efficiently, and effectively.

Given the increasing use and popularity of MBT, it is a natural consequence that the topic was picked up by academic research and teaching alike. The goal is to develop more sophisticated, powerful and efficient algorithms, tools and theory. Research in MBT techniques has brought forth a plethora of different frameworks, modelling mechanisms (each supporting various system properties), underlying theory, or tools to practically generate and execute tests. We refer to [182] for a formal taxonomy, and [181] for recent advances in the field.

However numerous the frameworks may be, all of them have the same conceptual ingredients, which this section aims to introduce. In later chapters we shall refer back to these ingredients, and compare how they are instantiated.

## 2.1 Overview

The fundamental approach of MBT is depicted in Figure 2.2. It bridges the gaps between the physical world, comprising real artefacts and real systems, and the formal world, where mathematical reasoning is possible, and vice versa.

The first encompasses an actual black-box implementation, e.g. inaccessible code of a programme, or an embedded system, together with its requirements. The intention of formal reasoning is to establish *conformance*. Certain behaviour is desired and expected, while it is preferable that the system does not exhibit other behaviour. This could mean that we eventually expect an ATM to dispense banknotes after credentials were provided, but not before.

However, to avoid ambiguities of what *precisely is desired*, the underlying hypothesis, henceforth referred to as *MBT hypothesis*, is that the behaviour of the black-box implementation can be represented by a particular modelling formalism. To be more specific: We assume that every possible concrete implementation has a unique corresponding object/model in the formal world. This enables

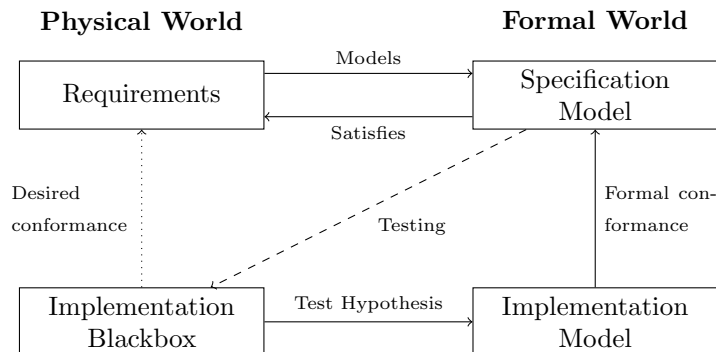


Figure 2.2: The model-based testing (MBT) approach.

us to relate the implementation model to the formal object corresponding to the requirements specification in a purely mathematical demeanour, e.g. via graph isomorphisms, (bi-)simulation, testing pre-orders, etc. Conformance of a black-box is thus unambiguously established its their formal counterpart.

It is here where the purpose of *testing* comes into play; very much like the content of the black-box is hidden from an external observer, the implementation model is not available to the tester. The only means of exploring its behaviour is by executing experiments, i.e. testing it, and consequently observe its reactions.

The goal of any testing method is to give a verdict about the correctness of the observed system. Thus, the specification model formally prescribes *correct* behaviour, and is used as an oracle to generate executable experiments also called *test cases*. The observed behaviour of the system under test (SUT) is then compared to the expected behaviour of the specification, and evaluated.

## 2.2 Components of Model-Based Testing

Even though there exist numerous MBT frameworks, each with their inherent advantages and restrictions, all follow the same basic pattern and have comparable components. The outlaid scheme depicted in Figure 2.2 necessitates each of those cogs in order for the MBT methodology to work as a whole.

In addition to both physical and formal components, all frameworks operate under the MBT hypothesis: It is assumed that each physical implementation has a corresponding model/object in the formal world. Naturally, each MBT methodology strives for correctness of its techniques. Table 2.1 summarizes the ingredients, while we scrutinize its contents below. A schematic overview of the interplay of components can be found in Figure 2.3.

Physical Ingredients:	Formal Ingredients:
<ul style="list-style-type: none"> <li>• Informal requirements</li> <li>• Black-box implementation</li> <li>• Test observations</li> </ul>	<ul style="list-style-type: none"> <li>• Specification model</li> <li>• Conformance relation</li> <li>• Test verdicts</li> </ul>
Tooling:	Objectives:
<ul style="list-style-type: none"> <li>• MBT tool</li> <li>• Test adapter</li> <li>• Test generation method</li> </ul>	<ul style="list-style-type: none"> <li>• Soundness</li> <li>• Completeness/Exhaustiveness</li> </ul>
Assumptions:	
<ul style="list-style-type: none"> <li>• Every physical implementation has a corresponding formal model</li> </ul>	

Table 2.1: Ingredients of a model-based testing framework after [15].

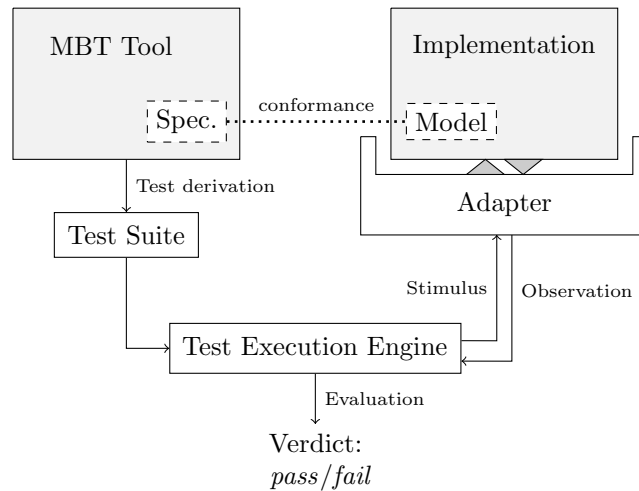


Figure 2.3: Schematic overview of model-based testing components.

**Physical ingredients.** Testing is carried out on real implementations. A trivial requirement to apply the MBT approach is the access to such systems. This includes the capability to interact with them in one way or another in order to perform experiments on them. Occasionally, this requires a *test adapter* mapping generic inputs to concrete implementation inputs, and concrete implementation outputs to more abstract outputs.

Moreover, every testing methodology requires a specification, i.e. a notion of *desired* behaviour. Without it, obviously neither MBT, nor any other test process is capable of establishing a verdict about correctness without the addition of a human oracle.

**Formal ingredients.** The MBT approach dictates, that conformance of the implementation to its requirements is established on a formal level. In order to talk about formal conformance, it is first necessary to translate the requirements into a formal model. This translation serves two purposes: 1. common oversights and ambiguities are detected early on in the model design process. This prevents errors in design to accumulate towards deeper levels of the V-model, cf. Figure 2.1, where they are far more costly to resolve, and 2. it is possible to argue about equivalence or conformance of models on a purely formal level, e.g. via graph isomorphisms, (bi-)simulation or testing pre-orders.

The latter, henceforth simply referred to as *conformance relation*, originates in the purpose of the framework: Does the focus strictly lie on correct functional behaviour? Are we solely interested in timed correctness? Or is the intent to stress test the system, and expose it to a high workload?

Given such a relation and two formal models, it is then a problem of algorithmic nature to check conformance. However, we only have a limited view of the



underlying system model, since its complete behaviour is hidden, i.e. MBT is a black-box method. Thus, it becomes evident that the decision of conformance is limited to parts of the system, that were revealed by executing experiments on them. This necessitates test verdicts, i.e. decision functions on conformance based on the limited view of the system that is accessible, and highlights the inherent incompleteness of testing methodologies.

**Tooling.** Testing, as opposed to formal verification, is a discipline that is carried out on real implementations. Under the assumption that we do not have access to the inner workings of these implementations, i.e. the systems under test are black-boxes. The only way of interacting with them is stimulating them via inputs, and to observe their potential output. The intent to automate this process therefore requires a tool to connect to physical systems. Its objective in conjunction with the implementation is to automatically generate, execute and evaluate experiments. Since the verdict whether an implementation is correct or not is rooted in the underlying formal methods techniques, it is desirable that an MBT tool generates test cases that comply with the same theory.

Another inherent concept of modelling is given by abstraction. Different levels of abstraction require different modelling complexity, e.g. if we intend to model continuous real-time, a discrete model might not suffice. To realize abstraction, the connection of an MBT tool and an implementation is often intercepted by an adapter, cf. Figure 2.3. Its role is twofold: 1. it provides an interface that connects MBT tool to the implementation, and 2. it abstracts, or refines inputs to, or outputs from the system under test.

**Objectives.** The intrinsic goal of the MBT approach is to establish a *correct* framework. Formal correctness comprises *soundness* and *completeness*, also sometimes referred to as *exhaustiveness*. These two properties in tandem necessitate that physical implementations pass a test suite if and only if their underlying model is deemed as conforming.

Specifically, soundness requires that a conforming implementation does indeed pass a test (suite). This logical condition is desirable in every framework, since its absence would invalidate the entire MBT approach. On the contrary, completeness is inherently a theoretical property. Formally, we demand that every non-conforming implementation is detectable by at least one test of a given test suite, or test generation method. However, programmes of infinite size, for instance caused by loops, naturally entail the need for an infinitely sized test suite. While practical completeness, i.e. the detection of *every* fault, is virtually impossible to achieve, it is frequently left as a theoretical result. It is commonly sufficient to show, that a test generation method is capable of generating a test that can reveal every possible misbehaviour. Another approach to provide complete test theories lies in the restriction of assumed implementation behaviour via e.g. fault models and fault domains [63, 100, 147]. The number of tests in a complete test suite can then be reduced to acceptable sizes [101]

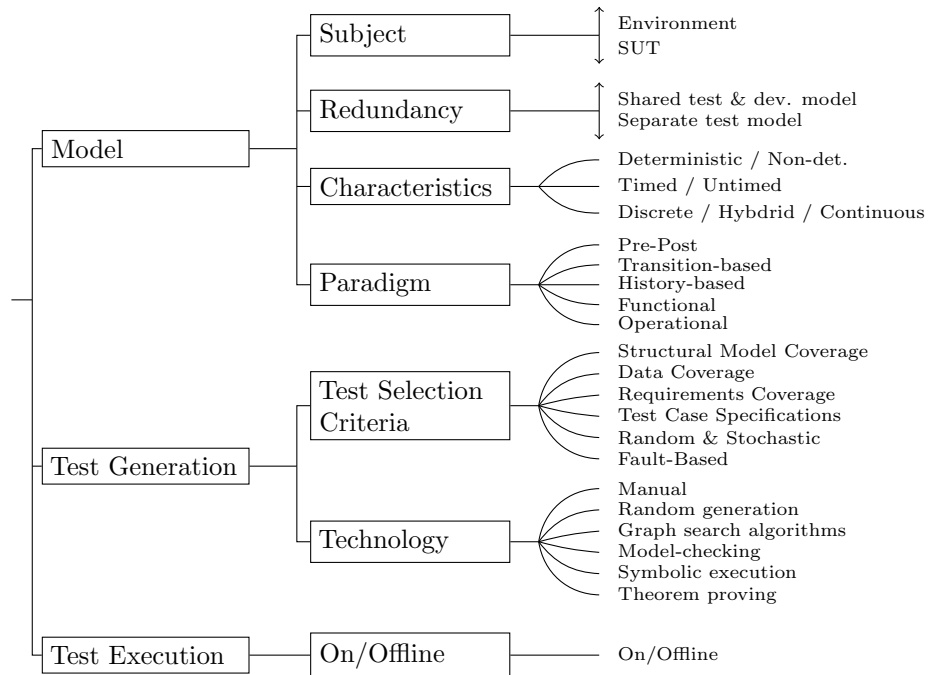


Figure 2.4: Taxonomy of model-based testing frameworks as presented in [182].

## 2.3 A Taxonomy of Model-Based Testing

The idea to perform testing based on a model can be dated back to the late seventies [42]. The flexibility that model-based engineering grants, allows for an equally flexible application domain. It is then no surprise that the usage of model-based testing comes in various shapes and sizes, and entails a vastly heterogeneous landscape. We present a formal classification given in the literature:

In their work, Pretschner et al. [182] establish a modern taxonomy of these different approaches. To capture a broad majority of the multitude of frameworks, the authors define the used terminology in a highly abstract manner. This allows, for instance, to classify typical graphically based approaches, like finite state machines or control flow charts, as well as pre- and postcondition centred approaches, that model a system as a snapshot of its internal variables.

The classification is achieved via seven orthogonal dimensions of model-based testing, as presented in Figure 2.4. Although being orthogonal, Pretschner et al. point out their existing influence among each other. A continuous model, for example, dictates the choice of test selection and test generation criteria.

We paraphrase the seven dimensions, and shortly discuss their implication with respect to a general MBT framework. However, for a complete discussion on the topic, we refer the avid reader to the original [182].

**Subject.** The first dimension is displayed as a continuous choice denoted by the arrows in Figure 2.4, rather than singular items. The general opposing axis are the modelled system behaviour, and the modelled environment behaviour. It is generally more beneficial to provide a mixture of both.

To illustrate, assume a model has *full* knowledge about the environment of the SUT, but has no indication about expected outputs or desired behaviour. Then, evidently usage behaviour is incorporated perfectly, while no verdict about the behaviour of the SUT is possible without the addition of a human oracle.

**Redundancy.** This dimension comprises the intended use of the model, i.e. the purpose of a model is its use for testing versus its simultaneous use for code generation. A prominent example for the latter is given by MATLAB's Simulink [141], allowing for automated code generation. However, models need to be very detailed to enable code generation. This might not be ideal for testing, as this is best done with some layers of abstraction.

**Characteristics.** This criterion encompasses the presence of non-determinism, timing constraints, and the continuous or discrete nature of the model. The particular choice of each of these three properties is naturally mutually exclusive, e.g. a model is either timed or untimed.

Note that this dimension is highly influential with respect to the others. For instance, the choice of a non-deterministic model with continuous real-time necessitates tests to be tree-shaped, rather than single traces, to account for various outcomes of non-deterministic choices of the system caused by jitter, or concurrency.

**Paradigm.** This dimension comprises the paradigm and notation used to model the system. Evidently, the used paradigm directly influences the power of the test generation methods.

The authors adapt the classification of van Lamsweerde [186], which includes 1. state-based notations (e.g. JML), 2. transition-based notations (e.g. FSMs, or I/O automata), 3. history-based notations (e.g. sequence diagrams), 4. functional notations (e.g. first, or higher order logic), 5. operational notations (e.g. Petri-nets or process algebras), 6. stochastic notations (e.g. Markov chains), and lastly 7. data-flow notations (e.g. MATLAB's Simulink [141]).

**Test Selection Criteria.** This dimension contains commonly used test selection criteria. It should be pointed out here, that no “best” criterion exists. It is widely acknowledged that an *optimal* test suite, and methods to generate one, are considered the “holy grail” in the MBT community [146].

The authors mention: 1. *structural model coverage*, which highly depends on the chosen paradigm. For instance, state, or transition coverage in graphical approaches like FSMs are commonly used. 2. *data coverage criteria*, describing how test values are selected from a large data space. Equivalence

partitioning and boundary analysis are two instances. 3. *requirements-based coverage*, in which requirements and tests are directly linked, which thus enable custom coverage criteria. This is frequently referred to as test purposes. 4. *Ad-hoc test case specifications*, which directly describe the pattern in which tests are to be selected, 5. *random and stochastic criteria* are applicable if the environment is modelled, and describe usage patterns of the system. Usage probabilities are modelled, and tests are generated accordingly. And lastly 6. *fault-based criteria*, which directly link system models and system faults. The assumption is the existence of a correlation between faults in the model and the SUT, and between mutations and real world faults. A prominent example is given by mutation testing [107].

**Technology.** The most appealing aspect of MBT is its potential for automation. This dimension encompasses the plethora of dedicated computer aided methods to automatically generate tests. Note that this dimension is heavily influenced by the chosen paradigm and its characteristics.

While manual generation is an option, MBT enables the generation of random tests, or utilize sophisticated graph search algorithms. In turn, these may strive to generate tests covering each edge or node. Complementary, model-checking can be adapted to generate test cases based on a reachability property, e.g. “eventually, a certain state is reached”. On the same note, the authors mention symbolic executions and theorem proving, checking the satisfiability of guards of transitions.

**On/Offline.** The criterion describes the relative timing of test generation and test execution. Generally, there are two approaches to generate, or execute test cases: *on-the-fly* (online), or in *batch* (offline).

On-the-fly testing allows the test generation algorithm to react to system outputs in real-time. This is of immense value in the face of a non-deterministic specification; the test generator sees, which path the system chose, and can thus react appropriately.

Offline testing refers to tests being generated strictly before their execution. Once generated, tests are stored and consequently executed. This allows for high reproducibility, and is of advantage in regression testing.

**Tool support.** The existence and development of tools is a necessity for every MBT framework. Its purpose is the *automation* of testing, and tools are natural means to achieve this goal. Nonetheless, tools are not listed in Figure 2.4, because they arise as a *result* of the interplay of the 7 orthogonal items. That is, their capabilities result from *a priori* design choices of the development team, and they are a means to achieve these goals rather than a goal in themselves.

The short summary of the taxonomy by Pretschner et al. [182] hints at the variety of existing frameworks, and there is no “best” MBT tool, as its purpose vastly differs with its application domain. We provide a brief collection of existing MBT tools in Table 2.2 alongside their underlying modelling formalisms

and a note on their availability as academic-, commercial-, or open source tool. This list is far from complete, but provides a broad overview of the variety and heterogeneity of the field. For a recent survey of MBT tools and their application in various case studies, we refer to [87].

Tool	Modelling Formalism	Availability	Reference
Conformiq	UML, QML	Commercial	[102]
GraphWalker	FSM	Open Source	[110]
JTorX <sup>†</sup>	LTS	Academic	[15]
MaTeLo	Markov Chains	Commercial	[65]
Mathworks MBT	Simulink Model	Commercial	[141]
ModelJUnit	EFSM	Open Source	[180]
SpecExplorer <sup>†</sup>	Model programs in C#	Commercial	[188]
TestCast	Custom	Commercial	[68]
TGV <sup>†</sup>	LOTOS/IOLTS	Academic	[105]
UPPAAL TRON <sup>†</sup>	Timed Automata	Academic	[121]
...	...	...	...

Table 2.2: Small selection of existing MBT tools. Centralised development of tools with the “†” mark has stopped.

**Benefits and drawbacks.** Among others, the model-based approach has three striking benefits: 1. Creating a model necessitates a firm definition of requirement descriptions. This supports the discovery of design flaws early in the development cycle, and serves as a unifying point of reference for a team of engineers, and potentially non-technical staff. 2. An existing model is reusable as part of the requirements specification. Evidently, this is largely beneficial upon the conduction of regression testing. 3. the direct application of the model to generate test cases.

Naturally, there are drawbacks to MBT, and practitioners should be aware of the caveats attached to it: 1. MBT is not an ad-hoc activity, and substantial training of staff is required to make use of its upsides. Obviously, this marks an initial one-time investment that is possibly extended with successional trainings. 2. Modelling costs time and effort. This is not limited to MBT only, but inherited from formal methods. Again, this is a one-time investment with the requirement of occasional model-maintenance. 3. It is not clear when to stop testing. This property is inherited from the natural incompleteness of testing in general. Test selection- and coverage criteria aid in quantifying the confidence of tested behaviour, but complete confidence in the correctness of an implementation is impossible to achieve outside of trivial examples. Apart from incompleteness, the drawbacks are characterised by being temporary – Medium sized to larger long-time projects thus generally warrant the initial time investment of training and modelling, as the relative cost decreases over the long haul.

The core idea of model-based testing is to use some form of abstractions of both the environment and system under test. Different layers and ways of abstracting, make the employment of MBT possible in various application domains. The increasing maturity of MBT frameworks in both the industrial and academic sector led to a plethora of publications. Due to the considerable amount of existing literature and the lack of an overarching unifying framework, it becomes a challenging endeavour to classify the vast majority of frameworks via one taxonomy. However, Pretschner et al. [182] provide an excellent and recent overview over the topic. Therefore, we deem their classification highly appropriate to provide the reader an outlook to techniques that go beyond the scope of this thesis. Other surveys may be found in [62, 87, 160]

## 2.4 Classification of the Probabilistic Framework

The recent years saw a proliferation of probabilistic programming languages [82] like Figaro [148] or probabilistic C [142]. Network algorithms make use of coin flips to establish a connection [106, 165], and routing algorithms based on random exploration see application in everyday household items like robotic lawn mowers or vacuum cleaners [168]. Online banking and security protocols use encryption methods that utilize random bits [41]. While the taxonomy provided in the previous section covers the vast majority of testing frameworks, these items indicate the lack of procedures to test *specified* use of probabilities, i.e. the dedicated use of probabilities in algorithms to achieve certain goals.

The purpose of this thesis is to cover this gap, by providing an MBT framework, capable of testing inherently *probabilistic systems*. We establish a sound overarching framework in a step-by-step manner, covering non-determinism, discrete probabilities, and stochastic time delay. Starting point of our journey is to recall the **ioco** framework [174], which lays the foundations we build upon. We are naturally drawn to it, because it supports non-determinism, thus enabling underspecification, and the composition of various subcomponents. The methodology of **ioco** ensures that *functional behaviour* is implemented correctly.

In a first step we conservatively extend this framework, by equipping specifications with *discrete probability choices*. Requirements may then dictate the quantified choice an implementation has, e.g. a coin shows *heads* 50% of the time. In addition to classic test generation and execution, the test evaluation then necessitates a large *sample* of test observations. This is to establish that probabilistic behaviour was implemented correctly, e.g. the fairness of the coin. Here, we make use of statistical hypothesis tests performed on the sample.

The next step enriches the framework with *stochastic time delays* on the specification side. That is, a specification may then additionally require *delays* between visible actions. This is evidently important in the realm of network connection algorithms, where waiting times dictate the behaviour of various components [106, 165]. We point out the difference to real-time testing [29], where time intervals are specified, in which events occur. Our framework dictates *time distributions*, e.g. the probability to establish connection after 5 seconds is

90%. Rather than checking the occurrence of a single event in a time interval, we thus need to check if *all time values* gathered in a sample follow a concrete probability distribution. This requires additional statistical hypothesis tests.

The previous section formally established a taxonomy of the model-based testing frameworks. To provide the reader with a detailed overview of the content of this thesis, we locate its position in the MBT landscape,

**Subject.** The behaviour of the system under test is modelled foremost. Environmental influences can be taken into account on the modelling level, e.g. parallel composition, or on the level of test selection and generation.

**Redundancy.** Models are primarily used for test generation. This is underlined by case studies [57, 177] for classic **io** theory. As of the time of writing, we are not aware of instances, that use the models in the **io** framework or its extensions for pure development purposes.

**Characteristics.** The classic framework entails non-deterministic, untimed and discrete models. We add discrete probabilities and stochastic time delay on the specification level. This item is not yet captured in Figure 2.4.

**Paradigm.** The methodology utilizes extensions of labelled transition systems, adds distinct input/output label sets, and accounts for *quiescence*. We refer to Figure 1.3 for an overview of modelling formalisms used.

**Test Selection Criteria.** Test selection criteria are based on state space exploration of the specification model. This is achievable via random test selection, or guided tests a.k.a. test purposes. Test selection criteria are mentioned, but not discussed in detail in this thesis.

**Technology.** Test generation in the framework is automatable. The technology is based on state exploration techniques and statistical hypothesis tests.

**On/Offline.** On- and an offline test generation/execution algorithms are provided.

The methodologies described within this thesis add another item to the *characteristics* established by Pretschner et al.[182], i.e. *Probabilistic* versus *Non-probabilistic*. Note, that this characteristic concerns inherent probabilistic choices of the system, e.g. coin flips, random number generators, or stochastic delay. It is not to be confused with random test generation/selection, or random environment models like Markov chains modelling usage behaviour. The latter is an entire research field of its own called *model-based statistical testing* (MBST). In contrast to our work, MBST studies test selection with respect to achieve certain goals, e.g. see [24, 16, 190].





## CHAPTER 3

---

### Model-Based Testing in the *ioco* Framework

---

We instantiate the model-based testing framework by the **input/output conformance** (**ioco**) framework and locate it in the MBT landscape established in the previous chapter. Alongside establishing the various components of an MBT framework summarized in Table 2.1, this framework forms the baseline for the remainder of this thesis. The **ioco** framework [174] lends itself as the *de rigueur* methodology for composable, and concurrent systems, dealing with interleaving, non-determinism and quiescent systems. It originated to test functional correctness of embedded systems, but rapidly caught the interest of other testing disciplines, e.g. timed properties [29], or hybrid frameworks [187].

We first define the very fundamental model, that is used for this MBT approach and add fine nuances to capture additional behaviour like *quiescence*. The underlying models for **ioco** are slight extensions of labelled transition systems (LTSs). LTSs are utilized to formally model the requirements specification, as well as test cases. Moreover, the MBT hypothesis assumes that the behaviour of the black-box implementation is representable by an LTS. In this way, it is possible to compare the black-box LTS to its specification LTS.

To that end, we define the required conformance relation **ioco** (input/output conformance), capturing precisely what *conformance* of models means. Further, we provide algorithms to derive test cases *on-the-fly*, or in *batch*, and show that their verdict corresponds to the conformance relations. Lastly, we show how tests are executed on an implementation and reference their proven correctness.

The framework established in this chapter lays the foundation for later chapters in the thesis. In particular, we extend LTSs with discrete probability distributions to get a probabilistic automaton (PA). They serve as underlying model for Chapter 4. As a next step, PA are extended with stochastic time delay in the form of exponential distributions in Chapter 5. This yields Markov automata (MA), which are an intermediate step towards the usage of *general* time distributions. A last step encompasses said general distributions and brings us to stochastic automata (SA) in Chapter 7.

The final modelling formalism thus incorporates non-determinism, discrete probabilities and stochastic time delay in the sense of general (continuous) probability distributions. Each chapter on MBT roughly exhibits the same structure that we will see in the current one, in that 1. the modelling formalism and its language are introduced, 2. the test theory of MBT with the formalism is established, and 3. applicable methods to use the MBT framework in practice are presented. At the end of each chapter, we will find a table that shows how the MBT ingredients were instantiated with respect to Table 2.1.

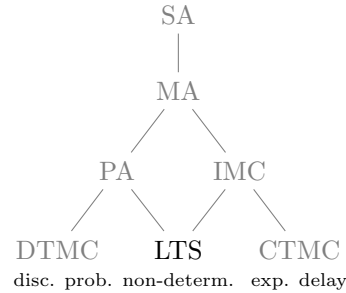


Figure 3.1: Traversing the automata formalism hierarchy shown in Figure 1.3.

### 3.1 Model and Language-Theoretic Concepts

The starting point to establish the framework is the underlying model. The baseline of our modelling language are labelled transition systems (LTSs). They graphically represent the behaviour of a system by defining the various states it can be in, and the eponymous labelled transitions it takes to go from one state to another. Note that the initial state of a system is uniquely determined.

**Definition 3.1** (Labelled transition system). *A labelled transition system (LTS) is a 4-tuple comprising  $\langle S, Act, \rightarrow, s_0 \rangle$ , where  $S$  is a countable, non-empty set of states,  $Act$  is a countable set of labelled actions,  $\rightarrow \subseteq S \times Act \times S$  is the transition relation, and  $s_0 \in S$  is the unique initial state.*

The set  $S$  models the various *states* that a system can be in. The unique starting state is marked by  $s_0$ , and represents the initial configuration of a system. The set  $Act$  is used to model possible actions it can perform via *action labels*. Throughout the thesis, we use *actions* and *labels* interchangeably. Every triple  $(s, a, s') \in \rightarrow$  is interpreted as moving the state of the system from  $s$  to  $s'$  via the action  $a$ . We refer to Figure 3.2 for an example.

In order to make the model comply with our intentions to model observable input/output behaviour, the action alphabet of LTSs is split into three disjoint sets. We write  $Act = Act_I \sqcup Act_O \sqcup Act_H$ , to denote the disjoint union of *input actions*, *output actions* and *internal/hidden actions* of a system, respectively. An LTS exhibiting this split is commonly referred to as an input/output transition system (IOTS).

Note that the **io** framework does account for the *absence of outputs*, commonly referred to as *quiescence*. If an implementation provides no response at all, it is a non-negligible task of the tester to judge the behaviour correct, or incorrect. To illustrate, an ATM should provide no output, if it is not interacted

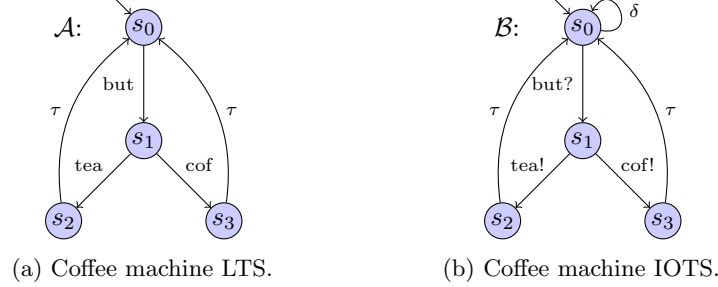


Figure 3.2: Distinguishing example for LTS and IOTS. While both have the same structure, the label set of IOTSs is split into inputs (suffixed by “?”), outputs (suffixed by “!”) and internal actions (ranged over by  $\tau$ ). IOTS  $\mathcal{B}$  has a  $\delta$ -labelled self-loop in state  $s_0$  that denotes the absence of outputs.

with, while no disbursement of banknotes is considered incorrect, after credit card credentials are provided. Quiescence is commonly denoted with the  $\delta$ -label [166], and is considered part of the output label set. Therefore, we always assume  $Act_O$  to contain this unique  $\delta$ -label, unless explicitly stated otherwise.

**Definition 3.2** (Input/Output transition system). *An input/output transition system (IOTS)  $\langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$  is an LTS, whose set of action labels is the disjoint union of input actions  $Act_I$ , output actions  $Act_O$  (incl.  $\delta$ ), and internal/hidden actions  $Act_H$ .*

The set of inputs and outputs, i.e.  $Act_I$  and  $Act_O$ , are referred to as the *visible actions*. They represent the behaviour, that is visible to an external observer and comprise stimuli applied to the system ( $Act_I$ ), or responses of it ( $Act_O$ ). The tuple  $(Act_I, Act_O)$  is called the *action signature*. Intuitively, it provides the interface, that enables communication with the system. On the contrary, the set of internal/hidden actions marks the internal, and thus invisible, progress that a system can make. Throughout this thesis, we let  $\tau$  be a representative of the set  $Act_H$ .

In a testing scenario, a tester should be able to provide any input at any time to the implementation. Formally, this is captured by *input-enabledness*, i.e. every state of an input-enabled system has an outgoing transition for every input action in  $Act_I$ . This notion of input-enabledness is known as *strong input-enabledness* [131], and used here over its weak counterpart [173].

**Example 3.3.** *Figure 3.2 shows a toy illustration of a coffee machine represented in both LTS and IOTS formalism. Both systems encompass four different states, i.e.  $s_0, s_1, s_2, s_3$ , with their respective unique initial state  $s_0$ .*

*While their action label set  $Act = \{but, cof, tea, \tau\}$  looks almost identical (barring  $\delta$ ), there is a distinction of inputs, outputs and internal labels in the IOTS Figure 3.2b. As commonly done in the literature, input actions are suffixed with a “?” , and output actions are suffixed with a “!”. These distinct sets*

provide the interface that enables communication among different components, or with their environment. The label  $\tau$  is a representative of the internal actions throughout this thesis.

Note that the IOTS in Figure 3.2b, has a newly added self-loop in  $s_0$  marked with the  $\delta$  label. This label represents quiescence, or absence of outputs, i.e. “no output” in the initial state is considered correct in a testing scenario.

While Figure 3.2 shows the models visually, the formal notation is given by:

$$\begin{aligned} \mathcal{A} &= \langle \{s_0, s_1, s_2, s_3\}, \{but, cof, tea, \tau\}, \\ &\quad \{(s_0, but, s_1), (s_1, tea, s_2)(s_1, cof, s_3), (s_2, \tau, s_0)(s_3, \tau, s_0)\}, s_0 \rangle \\ \mathcal{B} &= \langle \{s_0, s_1, s_2, s_3\}, \{but\}, \{cof, tea, \delta\}, \{\tau\}, \\ &\quad \{(s_0, \delta, s_0), (s_0, but, s_1), (s_1, tea, s_2)(s_1, cof, s_3), (s_2, \tau, s_0)(s_3, \tau, s_0)\}, s_0 \rangle \end{aligned}$$

We point out that both “?” and “!” are not part of the label, but aid the reader in understanding the visual representation, which is arguably easier to comprehend compared to the set notation.

**Remark 3.4** (A note on quiescence.). *Since its first occurrence in [173] the **io**co framework underwent various overhauls and restructurings, e.g. [174] and [169]. While the treatment of quiescent behaviour has traditionally been a focal point, it is Stokkink et al. [166] that first treat the  $\delta$ -label as a “first class citizen”, by imposing four rules of well-formedness to models.*

*These four rules account for correct treatment of quiescent, as well as divergent behaviour. Both formally treat the absence of system responses, though originate from different causes. While the first treats the sheer absence of output or internal actions in the states of a model (quiescent states, cf. state  $s_0$  in Figure 3.2b), the latter encompasses infinite internal progress (divergent states). Recall that internal actions are deemed invisible for an external observer: Even though a system may (infinitely) progress, an external observer can mistake the absence of visible system responses as quiescent.*

*The formal treatment of divergence is beyond the scope of this chapter, and we focus on non-divergent systems here. Regardless, we deem it important to mention the rules imposed by Stokkink et al. [166]. We paraphrase their rules:*

- R1** Quiescence should be observable,
- R2** the state after a quiescence observation is quiescent,
- R3** there is no new behaviour after the observation of quiescence, and lastly
- R4** continued quiescence preserves behaviour.

*A model obeying these rules is called a well-formed divergent quiescent transition system (DQTS). For the sake of simplicity, we adopt their rules, but simply call the resulting model a well-formed IOTS.*

**Parallel composition.** The popularization of component based design necessitates an equivalent in the modelling world. Individual components are designed separately and integrated later on. This greatly simplifies the modelling step

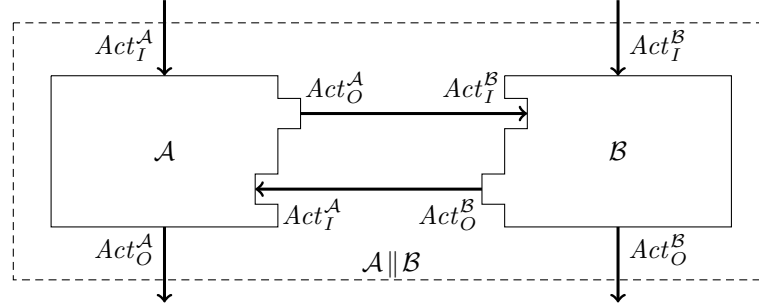


Figure 3.3: Schematic overview of parallel composition. Outputs of unit  $\mathcal{A}$  are regarded as inputs of unit  $\mathcal{B}$  and vice versa. Note that this approach enables further composition with more systems.

of larger scale systems, and adds an additional layer of quality control upon traversing the V-model as can be seen in Figure 2.1.

To that end, IOTSs are naturally equipped to mirror this design step in the formal world: Models of individual components are designed and factored together via parallel composition. The readiness and ease of this operator are two of the big advantages of IOTSs over other modelling formalisms like finite state machines or Petri-nets. The key idea comprises the individual development of system behaviour, and enforcement of synchronizing actions where it is desired. We formally define the parallel composition operator of IOTSs based on the common approach given in the literature, e.g. [9, 169].

However, before any synchronization can take place, we ensure that the systems to be composed, are indeed compatible. That is, we ensure that there is no undesired overlap in action label sets of participating components.

**Definition 3.5** (Compatibility). *Two IOTSs*

$$\begin{aligned} \mathcal{A} &= \langle S^{\mathcal{A}}, Act_I^{\mathcal{A}}, Act_O^{\mathcal{A}}, Act_H^{\mathcal{A}}, \rightarrow^{\mathcal{A}}, s_0^{\mathcal{A}} \rangle, \text{ and} \\ \mathcal{B} &= \langle S^{\mathcal{B}}, Act_I^{\mathcal{B}}, Act_O^{\mathcal{B}}, Act_H^{\mathcal{B}}, \rightarrow^{\mathcal{B}}, s_0^{\mathcal{B}} \rangle \end{aligned}$$

are compatible, if  $Act^{\mathcal{A}} \cap Act_H^{\mathcal{B}} = Act_H^{\mathcal{A}} \cap Act^{\mathcal{B}} = \emptyset$  and  $Act_O^{\mathcal{A}} \cap Act_O^{\mathcal{B}} = \{\delta\}$ .

As usual in the literature, we require internal and output actions of two components not to overlap. Output actions of one model are present as inputs in the respective other and vice versa. It is this overlap, that is synchronised upon. For an illustration, we refer to Figure 3.3.

**Definition 3.6** (Parallel composition). *Given two compatible IOTSs*

$$\begin{aligned} \mathcal{A} &= \langle S^{\mathcal{A}}, Act_I^{\mathcal{A}}, Act_O^{\mathcal{A}}, Act_H^{\mathcal{A}}, \rightarrow^{\mathcal{A}}, s_0^{\mathcal{A}} \rangle, \text{ and} \\ \mathcal{B} &= \langle S^{\mathcal{B}}, Act_I^{\mathcal{B}}, Act_O^{\mathcal{B}}, Act_H^{\mathcal{B}}, \rightarrow^{\mathcal{B}}, s_0^{\mathcal{B}} \rangle, \text{ their} \end{aligned}$$

parallel composition is the IOTS  $\mathcal{A} \parallel \mathcal{B} = \langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$ , where

- $S = S^A \times S^B$ ,
- $s_0 = (s_0^A, s_0^B)$ ,
- $Act_I = (Act_I^A \cup Act_I^B) \setminus (Act_O^A \cup Act_O^B)$ ,
- $Act_O = Act_O^A \cup Act_O^B$ ,
- $Act_H = Act_H^A \cup Act_H^B$ , and lastly
- $\rightarrow$  given by the set

$$\begin{aligned} & \{((s, t), a, (s', t')) \mid (s, a, s') \in \rightarrow^A \wedge (t, a, t') \in \rightarrow^B \wedge a \in Act^A \cap Act^B\} \\ & \cup \quad \{((s, t), a, (s', t)) \mid (s, a, s') \in \rightarrow^A \wedge a \in Act^A \setminus Act^B\} \\ & \cup \quad \{((s, t), a, (s, t')) \mid (t, a, t') \in \rightarrow^B \wedge a \in Act^B \setminus Act^A\}. \end{aligned}$$

The use of parallel composition within the **ioco** framework is twofold: While it still supports the component based design approach, it also facilitates the theoretic foundation for the execution of test cases on an implementation. Tests are designed to be IOTSSs, too. In this way, an implementation is composed with a test case, yielding the model of a *system under test* (SUT).

**Language-theoretic concepts.** We shortly introduce the notations induced by IOTSSs. The semantics of an IOTS  $\mathcal{A}$  are given by its set of *traces*, i.e. the behaviour visible to an external observer. Conversely, behaviour not associated with  $\mathcal{A}$  cannot be found in its set of traces. This lays the foundations of the conformance relation **ioco**.

Let  $\mathcal{A} = \langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$  be an IOTS. A path  $\pi$  in  $\mathcal{A}$  is a (possibly) infinite sequence of alternating elements of  $S$  and  $Act$ , e.g.

$$\pi = s_0 a_1 s_1 a_2 s_2 \dots,$$

such that for all  $i = 1, 2, \dots$ , we have  $(s_{i-1}, a_i, s_i) \in \rightarrow$ . That is, we allow only paths traversing the IOTS via its set of transitions. If  $\pi$  is finite, we require it to end in a state. In that case, we associate the number of visible actions in  $\pi$  with its *length* and denote it as  $|\pi|$  ( $\infty$  if  $\pi$  is infinite). Further, let  $first(\pi)$  and  $last(\pi)$  denote the first and the last states of  $\pi$ , respectively. We summarize all *finite paths* of  $\mathcal{A}$  in the set  $paths^{fin}(\mathcal{A})$ , and *all paths* in the set  $paths(\mathcal{A})$ .

The trace of the path  $\pi$  represents the behaviour, that is visible to an external observer. It is a (possibly) infinite sequence of input and output actions, e.g.

$$tr(\pi) = a_{i_1} a_{i_2} a_{i_3} a_{i_4} \dots,$$

where  $a_{i_j} \in Act_I \cup Act_O$  for  $j = 1, 2, \dots$ . The trace arises, if we omit all states and internal actions from the underlying path. The length of a trace is the number of its actions. Note that its length thus coincides with the length of its paths. We summarize all *finite traces* of  $\mathcal{A}$  in  $traces^{fin}(\mathcal{A})$ , and *all traces* in  $traces(\mathcal{A})$ , respectively. Moreover, we summarize all *complete traces*, i.e. traces that either have an infinite underlying path, or a path ending in a deadlock state, by the set  $traces^{com}(\mathcal{A})$ . For two IOTSSs  $\mathcal{A}$  and  $\mathcal{B}$ , we write  $\mathcal{A} \sqsubseteq_{tr} \mathcal{B}$ , if  $traces(\mathcal{A}) \subseteq traces(\mathcal{B})$ . We then say there is a *trace inclusion* of  $\mathcal{A}$  in  $\mathcal{B}$ .

Let  $\sigma, \sigma' \in (Act_I \cup Act_O)^*$  be a sequence of actions and  $s, s' \in S$  be states in  $\mathcal{A}$ . We denote the *concatenation* of two traces via  $\sigma \cdot \sigma'$ . Trace prefixes are denoted with  $\sqsubseteq$ , e.g.  $\sigma \sqsubseteq \sigma \cdot \sigma'$ . In case  $(s, a, s') \in \rightarrow$  we call action  $a$  *enabled* in state  $s$ . The set  $enabled(s)$  comprises all enabled actions of  $s$ . We write  $s \xrightarrow{\sigma} s'$ , if there is a path  $\pi$  with  $first(\pi) = s$ ,  $last(\pi) = s'$  and  $tr(\pi) = \sigma$ . That is,  $s'$  is reachable from  $s$  via the sequence  $\sigma$ . We write  $s \xrightarrow{a}$ , if  $s \xrightarrow{\sigma} s'$  for some  $s' \in S$ . Further, we denote with  $after_{\mathcal{A}}(s)$  the set of visible actions, that is enabled from state  $s$  of the IOTS  $\mathcal{A}$ , i.e.

$$after_{\mathcal{A}}(s) \stackrel{\text{def}}{=} \left\{ a \in (Act_I \cup Act_O) \mid s \xrightarrow{a} \right\}.$$

Note that this incorporates actions, which may only be reachable via intermediate internal actions. Formally, we overload  $after_{\mathcal{A}}$  for traces, i.e. for a trace  $\sigma$  the set  $after_{\mathcal{A}}(\sigma)$  contains all input and output actions, that are enabled in states reachable from the initial state  $s_0$  via  $\sigma$ . Finally, we define the set of output actions enabled after a trace  $\sigma$  as

$$out_{\mathcal{A}}(\sigma) \stackrel{\text{def}}{=} after_{\mathcal{A}}(\sigma) \cap Act_O$$

This set is the focal point of the conformance relation **ioco**.

### 3.2 The Conformance Relation $\sqsubseteq_{ioco}$

A binary conformance relation pins down mathematically, which implementation model is deemed correct with respect to a given specification model. The **input/output conformance (ioco)** relation has established itself as one of the core testing relations in the literature. It is rooted in the theory of testing equivalences and pre-orders [55, 56], and is highly flexible as it allows implementation freedom, as well as underspecification. Additionally, it is tailored to deal with *quiescent*, or *divergent* system behaviour, cf. Remark 3.4.

It originates in [173]. Since then, the **ioco** relation was the starting point of various MBT frameworks capable of testing more than just functional behaviour, e.g. timed behaviour [29], or hybrid systems [187]. It has received various overhauls, e.g. [169, 174], and is still the focus of ongoing research [183, 143].

**Definition 3.7** (ioco relation). *Let  $\mathcal{S}$  and  $\mathcal{I}$  be two IOTSs with the same action signature, and let  $\mathcal{I}$  be input enabled. We say  $\mathcal{I}$  conforms to  $\mathcal{S}$  with respect to **ioco**, denoted  $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$ , if and only if*

$$\forall \sigma \in traces^{fin}(\mathcal{S}) : out_{\mathcal{I}}(\sigma) \subseteq out_{\mathcal{S}}(\sigma).$$

The intuition behind the relation is straightforward: If system  $\mathcal{I}$  produces an output after executing an experiment derived from  $\mathcal{S}$ , then  $\mathcal{S}$  should account for this output. Conversely, unwarranted output is regarded as non-conforming. Note that **ioco** is not a pre-order, as it is neither reflexive nor transitive *per se* – the symbol  $\sqsubseteq_{ioco}$  indicates the embedding of the sets  $out_{\mathcal{I}}(\sigma)$  and  $out_{\mathcal{S}}(\sigma)$ .

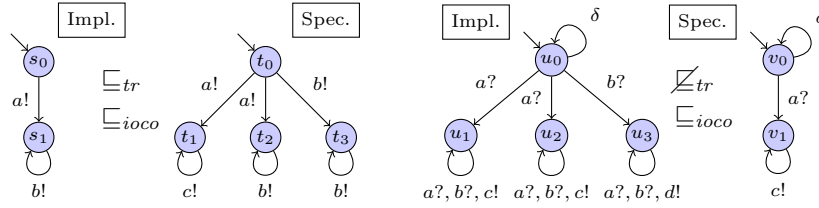


Figure 3.4: Illustrating example of the implementation freedom gained via the **ioco** relation, in comparison to trace inclusion.

At first glance **ioco** seems indistinguishable from trace inclusion  $\sqsubseteq_{tr}$ . However, upon closer inspection, **ioco** grants more freedom to the implementation. It allows behaviour, which is not formally prescribed by the specification. This is the formal mirror of the property, that correctness verdicts can only be given, if the behaviour is accounted for by a requirements specification. Figure 3.4 alongside Example 3.8 show why this is less restrictive than trace inclusion.

**Example 3.8.** Figure 3.4 shows two pairs of an implementation and a specification, respectively. These pairs exemplify the freedom given by the **ioco** relation, as opposed to mere trace inclusion.

The specification on the left hand side prescribes a non-deterministic choice between two outgoing  $a!$  output actions followed by repeated  $c!$  or  $b!$  actions, and an outgoing  $b!$  output followed by repeated  $b!$  actions. An implementation is now free to realize any, or all of the prescribed transitions. The shown implementation on the left has an outgoing  $a!$  followed by the repeated  $b!$  actions.

Obviously, all traces contained in the implementation model can be found in the corresponding specification model. This establishes trace inclusion. According to Definition 3.7, no matter which trace of the requirements we apply on the implementation, the set of outputs is subsumed by the set of outputs of the specification, i.e. disregarding the infinite  $b!$  loops, it is easily verified that:

$$\begin{aligned} out_{Impl.}(\varepsilon) &= \{a!\} \subseteq \{a!, b!\} = out_{Spec.}(\varepsilon) \\ out_{Impl.}(a!) &= \{b!\} \subseteq \{b!, c!\} = out_{Spec.}(a!), \text{ and} \\ out_{Impl.}(b!) &= \emptyset \subseteq \{b!\} = out_{Spec.}(b!). \end{aligned}$$

The pair on the right hand side distinguishes trace inclusion and the **ioco** relation. Note, that the implementation is input-enabled, i.e. all inputs are accounted for in the implementation in every state. The requirements specification only prescribes the output  $c!$  after the input  $a?$  was received. If no input is supplied, the system should do nothing, denoted by the self-loop with label  $\delta$ .

Clearly, the mentioned behaviour is present in the implementation. The trace  $b?d!$ , however, is not present in the specification model. Hence, there is no trace inclusion of the implementation model in the specification. However, since  $b?$  is an unspecified input in the initial state of the specification, the trace  $\sigma = b?$  is not a valid competitor for traces to check, cf. Definition 3.7. Therefore, the implementation is formally conforming with respect to **ioco**.



**Remark 3.9.** We point out that Definition 3.7 requires both models to be defined over the same action signature. Throughout the thesis, we usually represent models visually, like in Figure 3.4. If an action is not visually represented via a transition, we implicitly assume it to be in the action set regardless. To illustrate: even though the specification model on the right hand side does not have a transition with the  $b?$  label, we implicitly assume  $b?$  to be in  $Act_I$ .

### 3.3 Testing and Test Verdicts

We phrase the notion of offline test cases over an action signature  $(Act_I, Act_O)$ . Since the test continuation may depend on the history or outcome of earlier exhibited non-determinism, test cases are formalized as IOTSs in tree shapes. The action signature describes potential interaction of a test case, or tester, with the implementation. During each step of the test process, the test case/tester may decide to supply a stimulus, wait to observe the behaviour of the system, or stop the testing altogether.

In order for an observer to know in which state precisely a test case is, we define them as *internally deterministic*, i.e. there is a bijection between paths and traces of the test IOTS. In that way, no two paths yield the same trace.

**Definition 3.10** (Test case). A test case over an action signature  $(Act_I, Act_O)$  is an IOTS  $t$ , such that  $t$  is a finite internally deterministic, and connected tree. In addition we require for every state  $s \in S$  either:

- $enabled(s) = \emptyset$ , or (stop)
- $enabled(s) = Act_O$ , or (observe)
- $enabled(s) = \{a\}, a \in Act_I$ . (stimulate)

A test suite over an action signature  $(Act_I, Act_O)$  is a set of test cases over  $(Act_I, Act_O)$ . A test case (suite resp.) for an IOTS  $\mathcal{S}$  is a test case (suite resp.) over the action signature of  $\mathcal{S}$ .

Test cases model the behaviour of a tester. At each moment during the test process, a tester may stop testing, observe system output, or provide stimuli. This is represented by the three items in the list of Definition 3.10. The fundamental interactions are thus categorized by input and output labels, respectively. Hence, test cases are formalized as IOTS models. Moreover, the three mutually exclusive items make it impossible to combine a test suite to a single test case – every state either stops, provides input, or waits for output.

Since test cases are the formal representation of potential testing behaviour, they describe conscious choices made by a tester. Therefore it is essential that every observable action trace, can be uniquely identified within the test model. We guarantee this by requiring test cases to be internally deterministic; Given an action trace, we can follow precisely how the tree IOTS was traversed.

We require each test procedure to stop *eventually*. Hence, we have some constraints concerning the finiteness of the test model, i.e. acyclicity and no

infinite path. However, recall that the action alphabet consists of countably many actions. There is no restriction on the *branching* of a test case, and thus Definition 3.10 allows infinite branching in item 2.

**Remark 3.11** (A note on test cases.). *The test cases defined in Definition 3.10 originate in [173]. Note that a newer notion of test cases exist in the **ioco** framework based on more recent work [174]. The new test cases account for priority in practical testing. More specifically, if, upon test execution, a test case attempts to supply input to the implementation, there is a chance that the implementation overwrites this by providing an output beforehand. Tretmans' new test cases, therefore change the third item of tests to  $\text{enabled}(s) = \{a?\} \cup (\text{Act}_O \setminus \{\delta\})$ . This ensures that all outputs are accounted for at any stage in a test case. However, it does not incorporate the quiescence label  $\delta$ . Intuitively, quiescence denotes the absence of responses for an indefinite time; This contradicts the attempt of supplying input eventually, and is therefore not included.*

*We chose the 'old' test cases, because traces can be uniquely identified in the test tree IOTS. The introduction of internal non-determinism has delicate implications in a test environment for probabilistic systems, which we will study in Chapter 4. In particular, internally deterministic test cases play a crucial role to prove the MBT framework correct.*

**Annotations.** Before a test is executed on an implementation, it is crucial to formally define which outcome is deemed correct. Hence, we introduce test annotations; Test annotations are labels attached to each complete trace of the test tree IOTS. These traces are represented as the leaves of the tree. The two labels are *pass* and *fail* respectively. The intuition is straightforward: Desired behaviour is labelled *pass*, while undesired behaviour is labelled *fail*.

**Definition 3.12** (Test annotation). *Let  $t$  be a test case. A test annotation is a function  $\text{ann} : \text{traces}^{\text{com}}(t) \longrightarrow \{\text{pass}, \text{fail}\}$ . The pair  $\hat{t} \stackrel{\text{def}}{=} (t, \text{ann})$  is called an annotated test case, and a set of such pairs  $\hat{T}$  is called an annotated test suite.*

Note that Definition 3.12 formally defines annotations as a function onto the binary set *pass* and *fail*. It does, however, not yet describe how these labels are chosen. The precise choice of verdict labels per trace depends on the conformance relation. An annotation function assigning *pass* and *fail* verdicts according to the **ioco** relation with respect to a specification  $\mathcal{S}$  is given, if for all complete traces  $\sigma \in \text{traces}^{\text{com}}(t)$ , we have

$$\text{ann}_{\text{ioco}}^{\mathcal{S}}(\sigma) = \begin{cases} \text{fail}, & \text{if } \exists \varrho \in \text{traces}^{\text{fn}}(\mathcal{S}), a \in \text{Act}_O : \varrho a \sqsubseteq \sigma \wedge \varrho a \notin \text{traces}(\mathcal{S}) \\ \text{pass}, & \text{otherwise.} \end{cases}$$

We see that  $\text{ann}_{\text{ioco}}^{\mathcal{S}}$  only assigns the *fail* label for unforeseen output actions, which precisely mirrors the **ioco** relation. The encounter of a trace with *fail* label upon test execution gives enough evidence that an implementation is non-conforming with respect to **ioco**, and should consequently not pass the test.

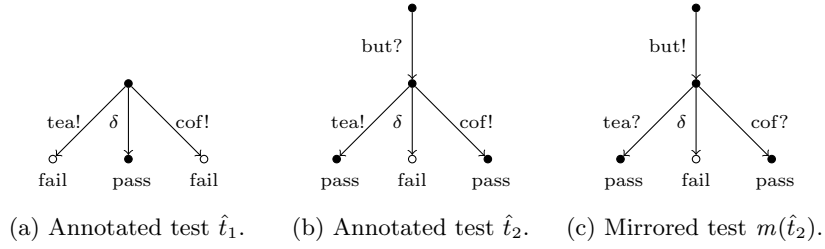


Figure 3.5: Annotated test cases derived from the IOTS in Figure 3.2b.

**Example 3.13.** Figure 3.5 shows two annotated test cases  $\hat{t}_1$  and  $\hat{t}_2$ , together with the mirroring of  $\hat{t}_2$ . Note that a mirrored IOTS has swapped input and output labels in order to allow for synchronisation with respect to parallel composition, cf. Definition 3.6. Mirroring is formally defined in Definition 3.14.

Test cases  $\hat{t}_1$  and  $\hat{t}_2$  are test cases for the specification of the toy coffee machine of Figure 3.2b. While  $\hat{t}_1$  validates the initial state to be quiescent, by only giving the state after  $\delta$ -transition the pass label, test  $\hat{t}_2$  checks the behaviour after the button input *but?* is received. If an implementation provides either of the two hot beverages, the annotation yields the pass label. However, observing no outputs, i.e. observing quiescence  $\delta$ , yields the fail label.

**Test execution.** Executing a test case on an implementation is formally represented by the parallel composition operator. The outputs that a test case provides are regarded as input to the system and vice versa. We refer to Figure 3.3 for an illustration. However, test cases are defined on the same action signature as the specification. To achieve synchronization between models, we introduce *mirroring*.

**Definition 3.14** (Mirroring of IOTSs). *Given an input/output transition system  $\mathcal{A} = \langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$ , we define its mirroring as*

$$m(\mathcal{A}) \stackrel{\text{def}}{=} \langle S, Act_O \setminus \{\delta\}, Act_I \cup \{\delta\}, Act_H, \rightarrow, s_0 \rangle.$$

The mirror operator for IOTSs is straightforward: It switches the input and output label sets, and thus enables synchronisation. For the remainder of this section, parallel composition only takes place between an implementation and a test model. Therefore, whenever we write  $\hat{t}$ , we refer to the mirrored test case  $m(\hat{t})$  implicitly, unless stated otherwise.

Parallel composition enables the interaction of an implementation  $\mathcal{I}$  and an annotated test case  $\hat{t}$ . The resulting system is referred to as *implementation under test (IUT)*. We summarize the set of traces of an IUT in the set  $exec_t(\mathcal{I})$ .

**Definition 3.15** (Test executions). *Let  $\mathcal{I} = \langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$  be an input enabled IOTS, and  $t$  be a test over the action signature  $(Act_I, Act_O)$ . Then the set of observable traces of  $\mathcal{I}$  under  $t$  is defined as*

$$exec_t(\mathcal{I}) \stackrel{\text{def}}{=} traces^{com}(\mathcal{I} \parallel t).$$

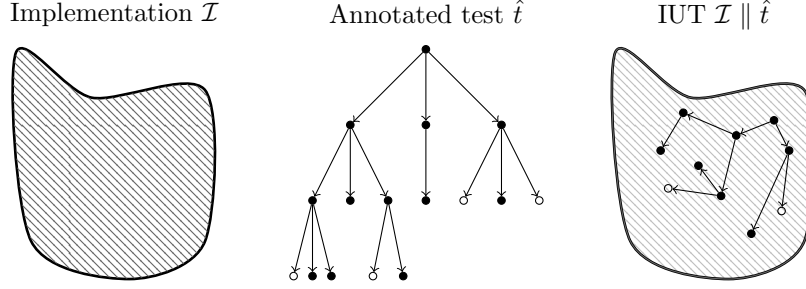


Figure 3.6: Schematic representation of implementation, annotated test and implementation under test. The shaded set on the left entails the complete, but unknown behaviour of the implementation. Hollow leaves of  $\hat{t}$  represent states labelled with *fail*, while filled ones represent *pass* states. Upon test execution, the marked behaviour in  $\mathcal{I} \parallel \hat{t}$  is revealed and summarized in  $exec_t(\mathcal{I})$ .

The set of observable traces of an implementation under test summarizes the behaviour, that an executed test potentially reveals. Figure 3.6 illustrates this. In practice, this may require repeated executions of the same test case due to non-deterministic behaviour and branching. In case no fairness restrictions are made for the implementation, i.e. all executable behaviour is eventually realized, it is possible that some traces are never observed in a practical setting. This shows the inherent incompleteness of testing in practice.

Note that Definition 3.15 crucially depends on the use of complete traces. By construction of annotated test cases, those are the only traces that reach states labelled with *pass* or *fail*, respectively.

**Test verdicts.** We assign a verdict to an implementation under test based on the test annotations. Intuitively, the implementation passes the test, if all encountered traces have the *pass* label according to their annotation. This means, we did not find evidence that the implementation is non-conforming to the specification with respect to **ioco**.

**Definition 3.16** (Test verdict). *Let  $\mathcal{S} = \langle S, Act_I, Act_O, Act_H, \rightarrow, s_0 \rangle$  be an IOTS, and  $\hat{t} = (t, ann_{ioco}^{\mathcal{S}})$  be an annotated test case for  $\mathcal{S}$ . The verdict function is then given as  $v : IOTS \times IOTS \rightarrow \{pass, fail\}$ , such that*

$$v(\mathcal{I}, \hat{t}) \stackrel{\text{def}}{=} \begin{cases} pass & \text{if } \forall \sigma \in exec_t(\mathcal{I}) : ann_{ioco}^{\mathcal{S}}(\sigma) = pass, \\ fail & \text{otherwise.} \end{cases}$$

*For a given test suite  $\hat{T}$  for  $\mathcal{S}$ , we set  $v(\mathcal{I}, \hat{T}) \stackrel{\text{def}}{=} pass$ , iff for all  $\hat{t} \in \hat{T}$  it holds that  $v(\mathcal{I}, \hat{t}) = pass$ .*

In practice, we only have a partial view of the set  $exec_t(\mathcal{I})$ . This reflects that testing is inherently incomplete; Even though no failure was discovered, the system might be faulty regardless. However, the verdict in Definition 3.16 gives

the benefit of the doubt: The implementation is assumed to be conforming, until we gathered sufficient evidence, that it is not.

**Example 3.17.** *We refer to Figure 3.6 for an illustration. The hollow leaves of the annotated test tree  $\hat{t}$  visually represent the label fail, while filled leaves have the pass label. An implementation passes the test if no trace in  $\text{exec}_{\hat{t}}(\mathcal{I})$  ends in a state that is labelled with fail. However, we can see that  $\mathcal{I} \parallel \hat{t}$  contains two traces ending in a failing state. Hence, the implementation fails the test case. Note, that in a practical test scenario, these two traces need to be encountered, in order to give the fail verdict according to Definition 3.16.*

### 3.4 Correctness of the Framework

Correctness comprises *soundness* and *completeness* (a.k.a. exhaustiveness), and constitutes the solid core of any MBT framework. These two properties in tandem necessitate that an implementation passes a test suite if and only if its underlying model is deemed conforming with respect to the conformance relation. It is evident that missing either one is indispensable, and every MBT approach needs to ensure its intrinsic correctness.

*Soundness* guarantees that conforming implementations indeed pass test cases and test suites. Conversely, *completeness* assures that the faults of every non-conforming system are detectable by at least one test of a test suite.

**Definition 3.18** (Soundness and completeness). *Let  $\mathcal{S}$  be an IOTS over the action signature  $(Act_I, Act_O)$ ,  $\hat{T}$  be an annotated test suite for  $\mathcal{S}$ , and  $R$  be a conformance relation. Then we say*

- $\hat{T}$  is sound for  $\mathcal{S}$  with respect to  $R$ , iff for all input enabled IOTS  $\mathcal{I}$  over  $(Act_I, Act_O)$ , it holds that  $(\mathcal{I}, \mathcal{S}) \in R \implies v(\mathcal{I}, \hat{T}) = \text{pass}$ .
- $\hat{T}$  is complete for  $\mathcal{S}$  with respect to  $R$ , iff for all input enabled IOTS  $\mathcal{I}$  over  $(Act_I, Act_O)$ , it holds that  $(\mathcal{I}, \mathcal{S}) \notin R \implies v(\mathcal{I}, \hat{T}) = \text{fail}$ .

The use of a general conformance relation  $R$  in Definition 3.18 is replaced by the **ioco** relation of Definition 3.7 for the purpose of our framework. The framework is correct for these choices. We refer the reader to [169] for the proofs.

**Proposition 3.19** (Soundness). *Let  $\mathcal{S}$  be an IOTS and  $\hat{t} = (t, \text{ann}_{ioco}^{\mathcal{S}})$  be an annotated test case for  $\mathcal{S}$ . Then  $\hat{t}$  is sound for  $\mathcal{S}$  with respect to  $\sqsubseteq_{ioco}$ .*

Completeness, on the other hand, is an inherently theoretical result. Loops in programs, for instance, may result in infinite implementation behaviour, consequently necessitating a test suite of infinite size.

**Proposition 3.20** (Completeness). *Let  $\mathcal{S}$  be an IOTS and  $\hat{T}$  be the set of all annotated test cases for  $\mathcal{S}$ . Then  $\hat{T}$  is complete for  $\mathcal{S}$  with respect to  $\sqsubseteq_{ioco}$ .*

We reiterate that Proposition 3.20 is a theoretical result. Due to inherent non-deterministic behaviour and no *fairness* assumptions, it is not guaranteed to eventually see the entire implementation behaviour, regardless of the test suite. Even though this result sounds trivial at first glance, it shows the interplay of the conformance relation, test cases and their annotations and test verdicts, which is by no means self-evident.

**Remark 3.21.** *We point out that [169] provide an additional criterion, that reduces the size of complete test suites. They show that it is sufficient to include test cases for all canonical traces of the specification, i.e. traces comprising multiple consecutive  $\delta$ -actions are reduced to traces only containing one such  $\delta$ -action. For instance, the canonical form of  $a? \delta \delta \delta b!$  is given by  $a? \delta b!$ .*

*While the size of test suites might still be infinite, this reduces the number of test cases to be executed significantly. Note that this results from the carefully designed well-formedness rules of our systems mentioned in Remark 3.4.*

### 3.5 Algorithms and Algorithmic Correctness

With the theoretical foundation of the test framework in place, we provide two test generation/execution algorithms: the *offline batch generation algorithm* and the *online on-the-fly execution algorithm*. While the first one generates test cases according to Definition 3.10, the latter generates and executes test cases on-the-fly. We present both algorithms taken from [169], discuss their advantages and drawbacks, and provide results guaranteeing their correctness.

**Batch generation.** Algorithm 3.1 presents the recursive procedure `batchGen`, that continuously assembles a test case. It requires a specification IOTS  $\mathcal{S}$ , a history  $\sigma$ , which is initially the empty history  $\varepsilon$ , and a maximal length of traces in the test  $n \in \mathbb{N}$ . The procedure then adds one or multiple branches to the current history. Every execution of the procedure makes a *non-deterministic* choice to return an empty trace and thus terminating a branch (line 3), to observe output of the system (line 5), or to provide a stimulus (line 14). Note that the decision is often implemented via a probabilistic choice in practice, e.g.[15]. The three choices reflect the behaviour a manual tester may exhibit upon interacting with the SUT.

While termination is straightforward (line 2) once a maximal test length is reached, lines 5-13 describe the step of observing the system: A branch is instantiated for every output action in  $Act_O$  (line 7). If an output is foreseen by the specification, it is added to the current branch, and the procedure `batchGen` is called again (lines 8 and 9). In case the output is not present in the specification, it is simply added to the branch (line 11) and is to be labelled *fail* in a consequent annotation sequence not present in the algorithm. Note, that `batchGen` is not called again in this circumstance, as erroneous behaviour is already accounted for. Upon reaching the maximal test length, each branch eventually terminates, and a (sub-)tree is added to the current state.

**Algorithm 3.1:** Batch test generation for **ioco**.

---

**Input:** IOTS  $\mathcal{S}$ , history  $\sigma \in \text{traces}(\mathcal{S})$ , and maximal length  $n \in \mathbb{N}$ .  
**Output:** A test case  $t$  for  $\mathcal{S}$ .

```

1 Procedure batchGen( $\mathcal{S}, \sigma, n$ )
2   if  $|\sigma| < n$  :
3     [true]  $\rightarrow$ 
4     | return  $\{\varepsilon\}$ 
5     [true]  $\rightarrow$ 
6     |  $i := 1$ 
7     | forall  $b! \in \text{Act}_O$  do:
8     |   if  $\sigma b! \in \text{traces}(\mathcal{S})$  :
9     |     | result $[i] := \{b!\sigma' \mid \sigma' \in \text{batchGen}(\mathcal{S}, \sigma b!, n)\}$ 
10    |   else:
11    |     | result $[i] := \{b!\}$ 
12    |     |  $i := i + 1$ 
13    |   return result $[1], \dots, \text{result}[|\text{Act}_O|]$ 
14    [Choose  $a? \in \text{Act}_I$  such that  $\sigma a? \in \text{traces}(\mathcal{S})$ ]  $\rightarrow$ 
15    | result  $:= \{a?\sigma' \mid \sigma' \in \text{batchGen}(\mathcal{S}, \sigma a?, n)\}$ 
16    | return result

```

---

Conversely, lines 14 and 16 corresponds to system stimulation. An input  $a?$  is chosen at random if it is present in the specification, and its branch is added to the tree, before calling **batchGen** again (line 15).

The procedure **batchGen** starts with the empty history  $\varepsilon$  and branches out in lines 9 and 11, thus ensuring the overall tree structure of test cases according to Definition 3.10. It should be pointed out, that no annotation takes place during the assembly of the test. This requires an additional, consequent step once **batchGen** terminates. Since no implementation is expected to run the procedure, its outcomes can be stored for later use. This guarantees the re-usability of the generated test cases.

**Remark 3.22.** *It is infeasible to store any non-trivial test case for a specification over an infinite number of outputs. A single observation for such systems already leads to an infinitely large test case. In this case, Algorithm 3.1 should be considered a pseudo-algorithm, and the next algorithm should be considered.*

**On-the-fly execution.** Algorithm 3.2 presents a sound procedure to execute, and evaluate tests on-the-fly. It requires a specification  $\mathcal{S}$ , an implementation  $\mathcal{I}$  and an upper limit on test length  $n \in \mathbb{N}$ . Initially, the execution history is empty. The history gets appended in every step of the iteration, until it stops when a total length of  $n$  is reached (line 2). Intuitively, the algorithm explores the behaviour of the implementation, while consulting the specification model for permission of the observed behaviour at every step. The algorithm makes a

---

**Algorithm 3.2:** On-the-fly test derivation for **ioco**.

---

**Input:** IOTS  $\mathcal{S}$ , implementation  $\mathcal{I}$  and maximal test length  $n \in \mathbb{N}$ .

**Output:** Verdict: *pass* if  $\mathcal{I}$  was conforming in the first  $n$  steps wrt. **ioco** and *fail* if not.

```

1  $\sigma := \varepsilon$ 
2 while  $|\sigma| < n$  do:
3   [true] $\rightarrow$ 
4   | observe next output  $b!$  (possibly  $\delta$ ) of  $\mathcal{I}$ 
5   |  $\sigma := \sigma b!$ 
6   | if  $\sigma \notin \text{traces}(\mathcal{S})$  : return fail
7   | [Choose  $a? \in \text{Act}_I$  such that  $\sigma a? \in \text{traces}^{fn}(\mathcal{S})$ ] $\rightarrow$ 
8   | | stimulate  $\mathcal{I}$  with  $a?$ 
9   | |  $\sigma := \sigma a?$ 
10 return pass

```

---

non-deterministic choice whether to observe the system (line 3), or to provide a stimulus (line 7) at every iteration.

Observing the system for outputs, or potential quiescence, is reflected in lines 3-6. It adds the observation to the history, in case it is allowed and proceeds to the next step (line 5). In case the output is not foreseen by the specification, the algorithm found sufficient evidence to conclude the implementation to be non-conforming with respect to **ioco** (line 6).

System stimulation is given in lines 7-9. An input available in the requirements model is chosen and applied to the system. The precise method on how an input is chosen is left vague, but a uniform distribution over all available inputs is common practice [15].

The algorithm returns a verdict of whether or not the implementation was found to be conforming with respect to **ioco** in the first  $n$  steps. In case erroneous output is detected, the verdict is *fail*, and *pass* otherwise. Note that the outcome reflects the inherent incompleteness of testing: the sole reason the *pass* verdict is given, is based on the lack of encountered faults.

**Proposition 3.23** (Algorithmic correctness). *Algorithm 3.1 generates test cases according to Definition 3.10. All test cases generated by Algorithm 3.2 are sound.*

**Remark 3.24.** *Quiescence denotes the indefinite absence of outputs. Clearly, this is infeasible in a practical testing scenario, and an alternate solution needs to take care of judging an implementation quiescent. Frequently, a global time out value is chosen in line 4 for the on-the-fly algorithm [15]. The authors of [29] provide the necessary theory by establishing timed-**ioco**. We encounter the dilemma of judging quiescence in Chapter 5, where a specification may require stochastic time delay between two actions. The challenge then becomes when to declare an implementation quiescent without confusing it with the desired delay exhibited by the implementation.*

*Algorithm 3.1 and 3.2 incorporate a non-deterministic choice between stop-*



Physical Ingredients:	Formal Ingredients:
<ul style="list-style-type: none"> <li>• Informal requirements</li> <li>• Black-box implementation</li> <li>• Observations: Def. 3.15, <math>exec_t(\mathcal{I})</math></li> </ul>	<ul style="list-style-type: none"> <li>• Model: Def. 3.2, IOTS</li> <li>• Conformance: Def. 3.7, <math>\sqsubseteq_{ioco}</math></li> <li>• Test verdicts: Def. 3.16</li> </ul>
Tooling:	Objectives:
<ul style="list-style-type: none"> <li>• MBT tool: JTorX [15], TorX [176], TorXakis [175], TGV [105], STG [43]</li> <li>• Test adapter: Implementable</li> <li>• Test generation method: Algorithms 3.1 and 3.2, Random testing</li> </ul>	<ul style="list-style-type: none"> <li>• Soundness: Prop. 3.19</li> <li>• Completeness: Prop. 3.20</li> </ul>
Assumptions:	
<ul style="list-style-type: none"> <li>• Every physical implementation has an underlying IOTS model</li> </ul>	

Table 3.1: The MBT ingredients instantiated by the **ioco** framework.

*ping, stimulating and observing. In practice this choice is frequently implemented probabilistically [15]. Thus, the algorithms are categorized as random testing according to the taxonomy by Pretschner et al. [182] presented in Figure 2.4. In addition, experimental research presented in [69] suggests better results, if stimuli are provided in a ratio of 67% versus observing the system in 33% of the time. While, there is no underlying theory supporting these findings, this might provide a reasonable starting point for future research.*

### 3.6 Summary and Discussion

Recall that Table 2.1 roughly scaffolds both the formal and physical ingredients needed for an MBT framework. In this chapter, we instantiated the ingredients for the **ioco** framework. We refer to Table 3.1 for an overview of the individual components. While some of the instances in the physical realm, such as informal requirements and an artefact implementation, are always assumed to be given, we formally defined test observations as the *test executions*. These represent the (partially) observed behaviour of the implementation upon test execution, and serve as a baseline to form a conformance verdict. The underlying models are labelled transition system with input- and output labels, i.e. IOTSs. These models render it possible to define the conformance relation **ioco**, which is

slightly coarser than trace inclusion. Test verdicts are defined in conjunction with test observations given by the set  $exec_t(\mathcal{I})$ .

Being one of the de rigueur methodologies for functional testing, it is natural that several MBT tools are based on **ioco** theory. While TorX [176] was developed closely in parallel to the underlying theory, JTorX [15] is its successor with heavy focus on academic teaching. Similar tools include STG [43] and TGV [105]. While the test adapter is individual per application, we provide two test generation algorithms in Section 3.5. These encompass offline batch generation, and on-the-fly test methods. Lastly, we provided evidence for the formal correctness of the framework, comprising soundness and completeness.

## CHAPTER 4

---

### Model-Based Testing with Probabilistic Automata

---

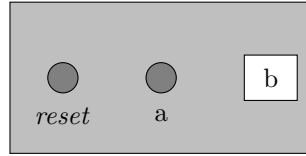
Many systems are inherently probabilistic: they interact with unpredictable environments, or use randomised algorithms. A rather naïve example is given by randomised games like roulette, or one-armed bandit slot machines: the outcome of these games is consciously chosen to be of probabilistic nature. Naturally, the use of probabilistic algorithms is not solely limited to game environments. With the growing popularity of probabilistic programming languages like probabilistic C [142] or Figaro [148] comes their application in various fields. Voice recognition is based on hidden Markov models [154], security protocols rely on random bits in their encryption methods [41], networking algorithms assign bandwidth or priority in a random fashion [106, 165], and the emerging field of probabilistic robotics is concerned with probabilistic routing methods [168].

Evidently, the classical model-based testing techniques described in Chapter 3 are insufficiently equipped to deal with probabilistic aspects. While the theory established therein, is capable of judging functional correctness of an implementation, it is not able to judge *probabilistic correctness*.

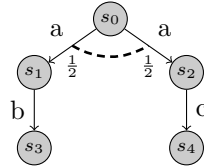
Recall the underlying hypothesis for model-based testing: every physical implementation has an underlying model. In a formal setting, this model can be compared to a specification model on a purely mathematical level. However, since we assume an implementation to be a black-box, an external observer does not have access to this underlying model. The only way for them to infer about the inner workings of the black-box, is to execute experiments, i.e. test cases.

Figure 4.1 illustrates this alongside the example of a fair coin. The implementation is given as a simple *trace machine*, together with a reset button, an action button  $a$ , and an observation window. The latter lets an external observer witness the currently executed action of the implementation. Upon pressing the button  $a$ , the system makes a probabilistic choice over  $b$  or  $c$ .

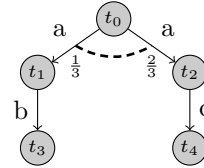
Evidently, the theory we established up to now judges whether the output  $b$  or  $c$  is correctly observed, but does not account for the observed *frequency*. Suppose a requirements specification demands the implementation of a fair coin, i.e. 50% observed  $b$  actions and 50% observed  $c$  actions. When repeating this



(a) Implementation with reset button.



(b) Model of fair coin.



(c) Model of unfair coin.

Figure 4.1: Black-box implementation and two possible underlying automaton models. Dashed arcs represent discrete probability choices.

*push button* experiment 100 times, what precisely is deemed acceptable, and how can we effectively tell the difference between both presented models?

The situation becomes even more interesting in the presence of both non-deterministic *and* probabilistic choices. We refer to Figure 4.2 for an illustration. First an *a* transition (or distribution, to be more precise) is chosen *non-deterministically*, and then the outcome is chosen *probabilistically*. We point out, that the given probabilities of both distributions differ.

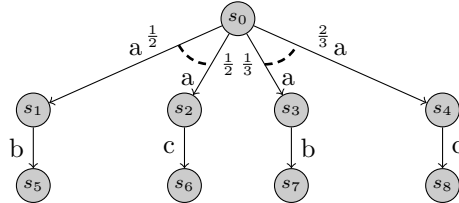


Figure 4.2: Model of non-deterministic coin.

Upon running the implementation of Figure 4.2 multiple times, we can neither guarantee that the observed *trace frequencies* are fair, nor can we guarantee that they are unfair. Thus, undoubtedly, judging the equivalence of two *probabilistic* automata becomes significantly more challenging.

In this chapter, we seek to establish a model-based testing (MBT) framework capable of handling both nondeterminism, and probabilistic choices. This entails all necessary ingredients of Table 2.1: 1. An automaton model capturing both of those properties, 2. a mathematical conformance relation, 3. the notion of test cases and algorithms to automatically generate them, 4. mathematical, and practical methods to give verdicts about correctness, and lastly 5. the proofs of

soundness and completeness of the framework.

Our underlying model of choice are probabilistic input/output transition systems (pIOTSs) - a conservative extension of both labelled transition systems and discrete time Markov chains, alongside a separation of the action alphabet into inputs and outputs. We refer to Figure 4.3 for an overview of the model hierarchy traversed in this thesis. Very much like pIOTSs arise naturally from LTSs, we extend **ioco** theory with probabilistic aspects to establish conformance of probabilistic models. Test cases and their annotations are defined similarly as for LTSs. Moreover, we define mathematical verdict functions, thus relating test observations to the requirements specification, and show how these verdicts are given in a practical scenario. The latter requires statistical analysis in the form of hypothesis testing. The application of the framework is illustrated on classical examples known from the literature.

We summarize the main contributions of this chapter:

- probabilistic input output transition systems, comprising discrete probability distributions and non-determinism,
- their behavioural description in form of trace distribution semantics,
- the probabilistic conformance relation **pioco**,
- definitions of test cases, test annotations, test executions and test verdicts,
- proofs for the correctness of the framework,
- test generation algorithms and applicable methods for probabilistic correctness in the Pearson's  $\chi^2$ -test, and
- three smaller size case studies known from the literature.

**Related Work.** Probabilistic testing pre-orders and equivalences are well-studied [45, 60, 158]. Distinguished work by [122] introduces the concept of probabilistic bisimulation via hypothesis testing. Largely influential work is given by [40], presenting how to observe trace frequencies during a sampling process. Executable probabilistic test frameworks are suggested for probabilistic finite state machines in [96, 103] and Petri nets [23].

**Origins of the chapter.** The work underlying this chapter was performed in collaboration with Mariëlle Stoelinga, and appeared in

- Marcus Gerhold and Mariëlle Stoelinga. ioco theory for probabilistic automata. In *Proceedings of the 10th Workshop on Model Based Testing, MBT*, pages 23–40, 2015,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE*, pages 251–268, 2016,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. *Formal Aspects of Computing*, 30(1):77–106, 2018.

**Organisation of the chapter.** The remainder of this chapter is structured as follows: Section 4.1 introduces the underlying automaton model in pIOTS and their language theoretic concepts. Section 4.2 establishes test cases, the conformance relation and test verdicts, and proves the framework to be correct. We provide applicable methods to apply model based testing according to the framework in practice in Section 4.3. In Section 4.4 we illustrate the frameworks’ applicability on small case studies known from the literature. Lastly, Section 4.5 concludes the chapter with final remarks.

## 4.1 Model and Language-Theoretic Concepts

We introduce the underlying automaton model for the remainder of this chapter in probabilistic input/output transition systems. We illustrate the formalism with easy examples and introduce language theoretic concepts, i.e. paths, traces, and parallel composition. Lastly, we introduce trace distribution semantics by means of schedulers.

### 4.1.1 Probabilistic Input Output Transition Systems

We introduce probabilistic input/output transition systems (pIOTSs) as an extension to labelled transition systems (LTSs). LTSs were studied in Chapter 3, and are mathematical structures that model the behaviour of a system, consisting of states and labelled edges between them. The first represent the status that a system can be in, while the latter models the actions it can perform.

We extend this formalism by allowing edges to be discrete probability distributions<sup>1</sup>; Instead of having one single target, edges may now have multiple target states alongside a discrete distribution describing the probability to reach them. This gives rise to *probabilistic automata* [158]. As such, probabilistic automata may also be seen as an extension to discrete-time Markov chains (not discussed in this thesis), with the addition of non-deterministic choices.

Similar to the step from LTSs to input/output transition systems (IOTSs), we separate the label alphabet in distinct input and output sets. This captures possible communication of a system with its environment, e.g. a tester or other components. Note that the set of outputs contains the distinct label  $\delta$ , explicitly requiring the absence of outputs. Like before, we allow internal

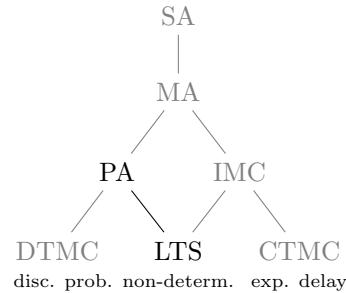


Figure 4.3: Traversing the automata formalism hierarchy shown in Figure 1.3.

<sup>1</sup>We refer the reader to Appendix A.1 for a formal introduction to the basics of probability theory used in this chapter.

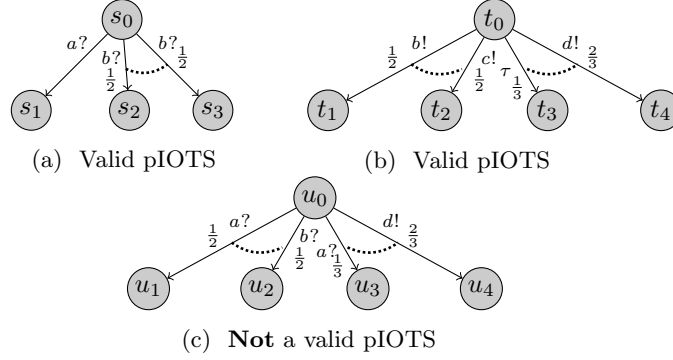


Figure 4.4: Example models to illustrate *input-reactive* and *output-generative* distributions in pIOTSs. We use “?” to denote labels of the set of inputs, and “!” to denote labels of the set of outputs. The symbol  $\tau$  denotes an action from the set of internal labels.

(invisible) actions, to be part of the label set. Those labels describe internal progress of a system, that is not visible to an external observer.

In order for our class of models to be larger than regular PA, we follow [185] in defining pIOTSs to be *input-reactive* and *output-generative*. These are two properties associated with the nature of the discrete distributions. Upon receiving an input action, the pIOTS decides probabilistically which next state to move to. However, upon producing an output, the pIOTS decides both the output and the state probabilistically. Formally, this means that each transition either involves one input action, or possibly several outputs, quiescence or internal actions. A state may enable input and output transitions, albeit not in the same *distribution*. A simple example for illustration purposes of input-reactive and output generative distributions is presented in Figure 4.4.

**Definition 4.1.** A probabilistic input/output transition system (*pIOTS*) is a sextuple  $\mathcal{S} = \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle$ , where

- $S$  is a finite set of states, with  $s_0$  as the unique starting state,
- $Act_I$ ,  $Act_O$ , and  $Act_H$  are disjoint finite sets of input, output and internal/hidden labels respectively, containing the distinct quiescence label  $\delta \in Act_O$ . We write  $Act = Act_I \sqcup Act_O \sqcup Act_H$  for the set of all labels.
- $\Delta \subseteq S \times Distr(Act \times S)$  is a finite transition relation such that for all input actions  $a \in Act_I$  and distributions  $\mu \in Distr(Act \times S)$ :  $\mu(a, s') > 0$  implies  $\mu(b, s'') = 0$  for all  $b \neq a$  and states  $s', s'' \in S$ .

**Example 4.2.** Figure 4.4 presents two valid example pIOTSs and an invalid one. As by common convention we use “?” to suffix input and “!” to suffix output actions. By default, we let  $\tau$  be an internal action. The target distribution of a transition is represented by a densely dotted arc between the edges belonging to it. Throughout this chapter, we commonly use  $\mu$  or  $\nu$  to denote distributions.

In Figure 4.4a there is a non-deterministic choice between the two inputs  $a?$  and  $b?$  modelling the choice that a user has in this state. If  $a?$  is chosen, the automaton moves to state  $s_1$ . In case the user chooses input  $b?$ , there is a 50% chance that the automaton moves to state  $s_2$  and a 50% chance it moves to  $s_3$ . Note that the latter distribution is an example of an input-reactive distribution according to the last item in Definition 4.1.

On the contrary, state  $t_0$  of Figure 4.4b illustrates output-generative distributions. Output actions are not under the control of a user or the environment. Hence, in  $t_0$  the system itself makes two choices: 1. it chooses one of the two outgoing distributions non-deterministically and 2. it chooses an output or internal action and the target state according to the chosen distribution. Note that both distributions are examples of output-generative distributions according to the last item in Definition 4.1.

Lastly, the model in Figure 4.4c is not a valid pIOTS according to Definition 4.1 for two reasons: 1. There are two distinct input actions in one distribution and 2. input and output actions may not share one distribution, as this would violate the last item in Definition 4.1.

**Notation.** By convention, we use the following notations and concepts:

- Elements of the set of input actions are suffixed by “?”, and elements of the set of output actions are suffixed by “!”. By convention, we let  $\tau$  represent an element of the set of internal actions, and  $\delta$  is the distinct output label to denote *quiescence*. Throughout this chapter we let  $\mu$  and  $\nu$  be discrete probability distributions.
- We write  $s \xrightarrow{\mu, a} s'$ , if  $(s, \mu) \in \Delta$  and  $\mu(a, s') > 0$ ,
- We write  $s \rightarrow a$  if there are  $\mu \in \text{Distr}(\text{Act} \times S)$  and  $s' \in S$  such that  $s \xrightarrow{\mu, a} s'$ , and  $s \not\rightarrow a$  if not.
- An action  $a$  is called *enabled* in a state  $s \in S$ , if  $s \rightarrow a$ . The set of all enabled actions in a state  $s \in S$  is denoted *enabled*( $s$ ).
- We write  $s \xrightarrow{\mu, a}_{\mathcal{S}} s'$ , etc. to clarify that a transition belongs to a pIOTS  $\mathcal{S}$  if ambiguities arise.
- We call a pIOTS  $\mathcal{A}$  *input enabled*, if all input actions are enabled in all states, i.e. for all  $a \in \text{Act}_I$  we have  $s \rightarrow a$  for all  $s \in S$ .

**Example 4.3.** Figure 4.5 shows four models of a simple shuffle music player with two songs. The pIOTS in 4.5a models the requirements: pressing the shuffle button enables the two songs with probability 0.5 each. The self-loop in  $s_1$  indicates that after a song is chosen, both are enabled with probability 0.5 each again. Pressing the stop button returns the automaton to the initial state. Note that the system is required to be quiescent in the initial state until the shuffle button is pressed. This is denoted by the  $\delta$  self-loop in state  $s_0$ .

- The pIOTS of Figure 4.5b is subject to a small probabilistic deviation in the distribution over songs. Contrary to the requirements, this implementation



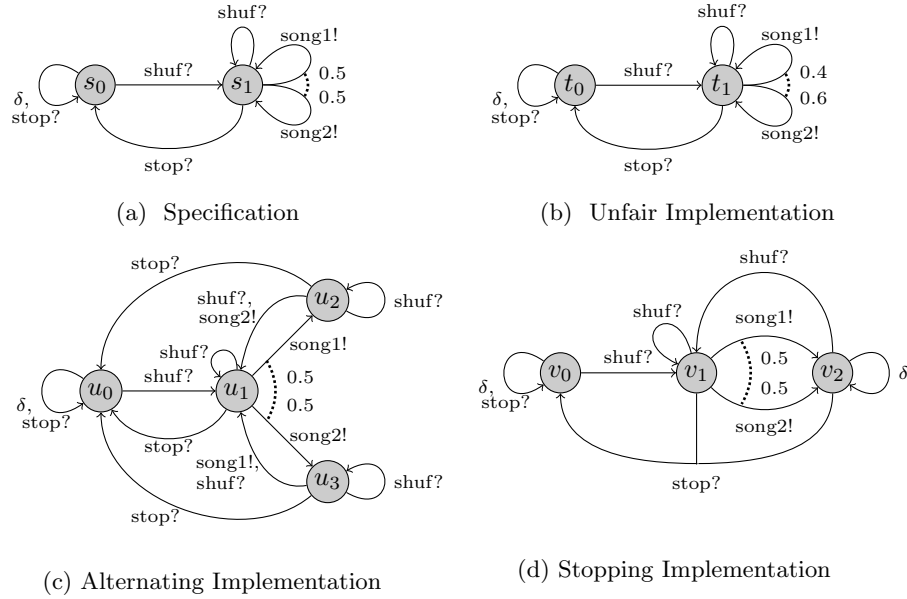


Figure 4.5: Specification and three possible implementation pIOTSs of a shuffle music player. Some actions are separated by commas for readability, indicating that two transitions with different labels are enabled from the same source to the same target states.

*chooses song1 with a probability of 40% and gives a higher probability to song2.*

- *In the pIOTS of Figure 4.5c the same song cannot be played twice in a row without interference of the user. After the shuffle button is pressed, the implementation plays one song and moves to state  $s_2$ , or  $s_3$  respectively. In these states only the respective other song is available. Contrary to the requirements, this pIOTS models the simplified creation of an entire play list upon pressing the shuffle button, that remains unchanged without intervention.*
- *The last pIOTS in Figure 4.5d correctly chooses either song with 50% probability, but stops after one song was played. This is illustrated by the  $\delta$ -labelled self-loop in  $v_2$ , denoting quiescence, or silence of the system.*

*Assuming that all incorrect models are hidden in a black box, the MBT framework presented in this chapter is capable of detecting all flaws.*

**Parallel Composition.** Parallel composition is defined in the standard fashion [9] by synchronizing on shared actions, and evolving independently on others. Since transitions in the component pIOTSs are assumed to be stochastically

independent, we multiply the probabilities when taking shared actions, denoted by  $\mu \otimes \nu$ . To avoid name clashes, we only compose *compatible* pIOTSs.

**Definition 4.4.** *Two pIOTSs*

$$\begin{aligned} \mathcal{S} &= \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle, \text{ and} \\ \mathcal{S}' &= \langle S', s'_0, Act'_I, Act'_O, Act'_H, \Delta' \rangle, \end{aligned}$$

are compatible if  $Act_O \cap Act'_O = \{\delta\}$ ,  $Act_H \cap Act'_H = \emptyset$  and  $Act \cap Act'_H = \emptyset$ .

**Definition 4.5.** Let  $\mathcal{S} = \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle$  and  $\mathcal{S}' = \langle S', s'_0, Act'_I, Act'_O, Act'_H, \Delta' \rangle$  be two compatible pIOTSs. Their parallel composition is the tuple

$$\mathcal{S} \parallel \mathcal{S}' = \langle S'', (s_0, s'_0), Act''_I, Act''_O, Act''_H, \Delta'' \rangle, \text{ where}$$

- $S'' = S \times S'$ ,
- $Act''_I = (Act_I \cup Act'_I) \setminus (Act_O \cup Act'_O)$ ,
- $Act''_O = Act_O \cup Act'_O$ ,
- $Act''_H = Act_H \cup Act'_H$ , and finally the transition relation
- $\Delta'' = \{((s, t), \mu) \in S'' \times Distr(Act'' \times S'') \mid$ 

$$\mu \equiv \begin{cases} \nu_1 \otimes \nu_2, & \text{if } \exists a \in Act \cap Act' \text{ such that } s \rightarrow a \wedge t \rightarrow a \\ \nu_1 \otimes \mathbb{1}, & \text{if } \forall a \in Act \text{ with } s \rightarrow a \text{ we have } t \not\rightarrow a \\ \mathbb{1} \otimes \nu_2, & \text{if } \forall a \in Act' \text{ with } t \rightarrow a \text{ we have } s \not\rightarrow a \end{cases},$$

where  $(s, \nu_1) \in \Delta$ , and  $(t, \nu_2) \in \Delta'$ . Here  $\nu_1 \otimes \nu_2((s', t'), a) = \nu_1(s', a) \cdot \nu_2(t', a)$ ,  $\nu_1 \otimes \mathbb{1}((s', t'), a) = \nu_1(s', a) \cdot 1$  and  $\mathbb{1} \otimes \nu_2((s', t'), a) = 1 \cdot \nu_2(t', a)$ .

The reader is referred to [158] for an interesting discussion on the intricacies for parallel composition of *general probabilistic automata*. The main problems are: when should two distributions synchronise, and how should synchronisation occur. We avoid the pitfalls discussed there by the structure of pIOTSs, and which pIOTSs is synchronised upon. In particular, we compose test cases and implementations to get the set of *observable traces* akin to the set  $exec_t(\mathcal{I})$  as seen in Chapter 3. Like before it is sensible to assume implementations to always be input-enabled. Moreover, input-reactive and output-generative distributions ensure that the handling of discrete probabilities is carried out correctly.

### 4.1.2 Paths and Traces

We recall the usual language concepts for LTSs and adapt them to our needs for pIOTSs. Let  $\mathcal{S} = \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle$  be a pIOTS.

A *path*  $\pi$  of  $\mathcal{S}$  is a (possibly) infinite sequence of the form

$$\pi = s_0 \mu_1 a_1 s_1 \mu_2 a_2 s_2 \mu_3 a_3 s_3 \dots,$$

where  $s_i \in S$ ,  $a_i \in Act$  and  $\mu_i \in Distr(Act \times S)$ , such that each finite path ends in a state, and  $s_i \xrightarrow{\mu_{i+1}, a_{i+1}} s_{i+1}$  for each non-final  $i$ . We use *last* ( $\pi$ ) to denote the last state of a finite path. We write  $\pi' \sqsubseteq \pi$  to denote  $\pi'$  as a *prefix* of  $\pi$ , i.e.  $\pi'$  is finite, ends in a state, and coincides with  $\pi$  on the first finitely many symbols of the sequence. The set of all finite paths of  $\mathcal{S}$  is denoted as  $paths^{fin}(\mathcal{S})$  and all paths by  $paths(\mathcal{S})$ . The set of *complete paths*,  $paths^{com}(\mathcal{S})$ , contains every infinite path, or paths that end in a state that does not enable further actions.

The associated *trace* of a path  $\pi$  is the (possibly infinite) sequence obtained by omitting states, distributions and internal actions from  $\pi$ . Formally, it is a mapping  $tr : paths(\mathcal{S}) \rightarrow (Act_I \cup Act_O)^\omega$ . We overload the mapping, such that it accounts for finite paths, too, i.e.  $tr : paths^{fin}(\mathcal{S}) \rightarrow (Act_I \cup Act_O)^*$ , with

$$tr(\pi) = a_{i_1} a_{i_2} a_{i_3} \dots, \text{ where } a_{i_j} \in Act_I \sqcup Act_O \text{ for } j = 1, 2, \dots$$

$tr^{-1}(\sigma)$  gives the set of all paths, which have trace  $\sigma$ . The *length* of a path is the number of actions on its trace. All finite traces of  $\mathcal{S}$  are summarized in  $traces^{fin}(\mathcal{S})$ , and all traces in  $traces(\mathcal{S})$ . The set of *complete traces*,  $traces^{com}(\mathcal{S})$ , contains every trace based on complete paths.

Note that an IOTS is a pIOTS, in which every element of  $\Delta$  is the *Dirac distribution*, i.e. the distribution with a single target. It is straightforward to extend the definitions of the operators *after*, and *out* used for IOTSs in Section 3.1. Recall that the first describes the available actions in a set of states, whereas the latter characterises the available output actions after a trace.

### 4.1.3 Schedulers and Trace Distributions

Probabilistic input/output transition systems comprise non-deterministic, as well as probabilistic choices. As such, it is not possible to assign probabilities to finite paths, or traces directly. Paths embody the resolution of non-deterministic choices of the states it contains. Since a state may be visited multiple times, and choices may vary, the resolution becomes *history dependent*. That is, the resolution of non-determinism depends both on the current state *and* its prefix.

Lifting the resolution of non-deterministic choices of paths to an entire pIOTSs introduces the concept of *schedulers* [9]. The resolution of the non-determinism via a scheduler leads to a purely probabilistic system, in which probability is assigned to each finite path.

Following the standard theory for probabilistic automata [158], we define the behaviour of a pIOTS via schedulers (a.k.a. policies, adversaries, or strategies) to resolve the non-deterministic choices; in each state of the pIOTS, the scheduler may choose which transition to take, or it may also halt the execution entirely. Given any finite history leading to a state, a scheduler returns a discrete probability distribution over the set of next transitions (or distributions to be precise). In order to model termination, we define schedulers such that they can continue paths with a halting extension  $\perp$ , after which only quiescence is observed.

**Definition 4.6.** A scheduler  $\mathcal{A}$  of a pIOTS  $\mathcal{S} = (S, s_0, Act_I, Act_O, Act_H, \Delta)$  is a function

$$\mathcal{A} : paths^{fn}(\mathcal{S}) \longrightarrow Distr(Distr(Act \times S) \cup \{\perp\}),$$

such that for each finite path  $\pi$ , if  $\mathcal{A}(\pi)(\mu) > 0$ , then  $(last(\pi), \mu) \in \Delta$  or  $\mu \equiv \perp$ . The value  $\mathcal{A}(\pi)(\perp)$  is considered as interruption/halting. A scheduler  $\mathcal{A}$  halts on a path  $\pi$ , if  $\mathcal{A}(\pi)(\perp) = 1$ . We say that a scheduler halts after  $k \in \mathbb{N}$  steps, if it halts for every path of length greater or equal to  $k$  and for every complete path smaller than  $k$ . We denote all such finite schedulers by  $Sched(\mathcal{S}, k)$ . The set of all finite schedulers of  $\mathcal{S}$  is denoted  $Sched(\mathcal{S})$ .

Note that, to account for input-reactive and output-generative distributions instead of scheduling *actions* a scheduler chooses *distributions*. The first clause of Definition 4.6 points out, that only distributions may be scheduled, which are available in the current state.

Moreover, the class of schedulers we consider are *history dependent* and *randomized*. We study the hierarchy of scheduler classes of *stochastic automata* (SA) in Chapter 8, and point out that PA are a subclass of SA.

**Probability Spaces associated to Schedulers.** A scheduler enables the quantification of probability of single paths, since all non-deterministic choices are resolved. To that end, we require the notion of paths of a scheduler, i.e. paths with a non-zero probability.

**Definition 4.7.** A path  $\pi$  of a scheduler  $\mathcal{A}$  is a finite or infinite path

$$\pi = s_0 \mu_1 a_1 s_1 \mu_2 a_2 s_2 \mu_3 a_3 s_3 \dots$$

such that  $\mathcal{A}(s_0 \mu_1 a_1 s_1 \dots s_i)(\mu_i, a_i) > 0$  for each  $i = 1, 2, \dots$ . The maximal paths of a scheduler  $\mathcal{A}$  are the infinite paths of  $\mathcal{A}$  and the finite paths  $\pi$ , such that  $\mathcal{A}(\pi)(\perp) > 0$ . We set  $paths^{max}(\mathcal{A})$  as the set of maximal paths of  $\mathcal{A}$ .

Intuitively, a scheduler rolls a multi-faced and biased die at every step of the computation, telling the system how to proceed. Consecutive scheduler decisions thus induce a path probability distribution on a pIOTS. Hence, the resulting system is a purely probabilistic computation tree.

To construct a proper probability distribution over all finite paths, the probability assigned to a single path  $\pi$  is obtained by the probability of its cone  $C_\pi = \{\pi' \in paths^{max}(\mathcal{A}) \mid \pi \sqsubseteq \pi'\}$ , cf. Figure 4.6. This construction is standard and thus left undiscussed here. Instead we refer to [158, 164] for more details.

**Definition 4.8.** A scheduler  $\mathcal{A}$  of a pIOTS  $\mathcal{S}$  induces a path probability function inductively defined by  $Q^{\mathcal{A}}(s_0) = 1$  and

$$Q^{\mathcal{A}}(\pi \mu a s) = Q^{\mathcal{A}}(\pi) \cdot \mathcal{A}(\pi)(\mu) \cdot \mu(a, s).$$

The probability of  $\mathcal{A}$  generating exactly  $\pi$  equals  $Q^{\mathcal{A}}(\pi) \cdot \mathcal{A}(\pi)(\perp)$ .

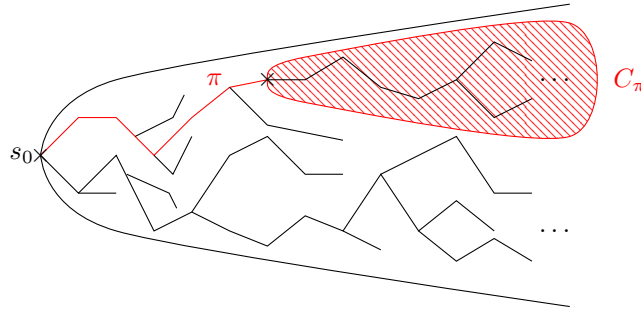


Figure 4.6: Abstract illustration of the cone of a path. The outermost curve describes all possible behaviour of a pIOTS, and lines in it point out possible developments of the system over time. The red curve highlights a specific path  $\pi$ , which is appended by its cone  $C_\pi$ , i.e. all possible paths that have  $\pi$  as prefix.

A scheduler  $\mathcal{A}$  thus defines a unique probability distribution  $P_{\mathcal{A}}$  on the set of (finite) paths. Therefore, the probability of  $\pi$  is  $P_{\mathcal{A}}[C_\pi] \stackrel{\text{def}}{=} Q^{\mathcal{A}}(\pi)$ . Hence, the path probability function enables us to define a unique probability space<sup>2</sup>  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$  associated to a scheduler  $\mathcal{A}$ .

**Definition 4.9.** *The probability space associated to a scheduler  $\mathcal{A}$  of a pIOTS  $\mathcal{S}$ , is the probability space given by  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$ , where*

1.  $\Omega_{\mathcal{A}} = \text{paths}^{\text{max}}(\mathcal{A})$ ,
2.  $\mathcal{F}_{\mathcal{A}}$  is the smallest  $\sigma$ -field generated by the set  $\{C_\pi \mid \pi \in \text{paths}^{\text{fin}}(\mathcal{S})\}$ , where  $C_\pi = \{\pi' \in \Omega_{\mathcal{A}} \mid \pi \sqsubseteq \pi'\}$ , and
3.  $P_{\mathcal{A}}$  is the unique measure on  $\mathcal{F}_{\mathcal{A}}$  such that  $P_{\mathcal{A}}[C_\pi] = Q^{\mathcal{A}}(\pi)$  for all  $\pi \in \text{paths}^{\text{fin}}(\mathcal{S})$ .

Note that  $\Omega_{\mathcal{A}}$  and  $\mathcal{F}_{\mathcal{A}}$  do only indirectly depend on  $\mathcal{S}$ , and are fully determined by  $Q^{\mathcal{A}}$ . The fact that the triple  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$  constitutes a probability space follows from standard measure theory arguments, see e.g. [46, 164].

**Example 4.10.** *Recall the requirements specification of a shuffle music player in Figure 4.5a. Assume there is a scheduler  $\mathcal{A}$  that chooses the distribution containing shuf? in state  $s_0$  with probability 1, and from there on schedules the distribution containing song1 and song2 with probability 1 finitely many times.*

*Asking for the probability of hearing song1 after the shuffle button is pressed is then answered by considering the cone of the path*

$$\pi = s_0, \mu_1, \text{shuf?}s_1, \mu_2, \text{song1!}, s_1.$$

*The cone  $C_\pi$  consists of all paths starting with  $\pi$ , that are appended in all possible*

<sup>2</sup>We refer to Appendix A.1 for a summary of the parts of probability theory needed here.

ways. Thus, the probability to observe exactly  $\pi$  becomes

$$\begin{aligned} P_{\mathcal{A}}[C_{\pi}] &= Q^{\mathcal{A}}(\pi) = Q^{\mathcal{A}}(s_0) \cdot \mathcal{A}(s_0)(\mu_1) \cdot \mu_1(\text{shuffle?}, s_1) \cdot \\ &\quad \mathcal{A}(s_0, \mu_1, \text{shuffle?}, s_1)(\mu_2) \cdot \mu_2(\text{song1}, s_1) \\ &= 1 \cdot 1 \cdot 1 \cdot 1 \cdot 0.5 \\ &= 0.5. \end{aligned}$$

We point out that the usage of cones is essential here as  $\mathcal{A}$  does not stop after scheduling the first song.

**Trace Distributions.** Very much like traces are obtained by first selecting a path and by then removing all information that is invisible to an external observer, we do the same in the probabilistic case. First, we resolve all non-deterministic choices in the pIOTS via a scheduler, and then remove all state information to get its *trace distribution*. This allows us to quantify the probability associated to traces in a pIOTS.

A trace distribution is obtained from (the probability space of) a scheduler by removing all states. Thus, the probability assigned to a set of traces  $X$  is the probability of all paths whose trace is an element of  $X$ .

**Definition 4.11.** The trace distribution  $\mathcal{D}$  of a scheduler  $\mathcal{A} \in \text{Sched}(\mathcal{S})$ , denoted  $\mathcal{D} = \text{trd}(\mathcal{A})$  is the probability space  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$ , where

- $\Omega_{\mathcal{D}} = \text{Act}^* \cup \text{Act}^{\omega}$ ,
- $\mathcal{F}_{\mathcal{D}}$  is the smallest  $\sigma$ -field generated by the set  $\{C_{\varrho} \mid \varrho \in \text{Act}^*\}$ , where the cone is  $C_{\varrho} = \{\varrho' \in \Omega_{\mathcal{D}} \mid \varrho \sqsubseteq \varrho'\}$ ,
- $P_{\mathcal{D}}$  is the unique probability measure on  $\mathcal{F}_{\mathcal{D}}$  such that

$$P_{\mathcal{D}}(X) = P_{\mathcal{A}}(\text{tr}^{-1}(X)) \text{ for } X \in \mathcal{F}_{\mathcal{D}}.$$

The fact that  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$  defines probability spaces is completely standard, and not discussed further here. Additional material and discussions on the topic can be found in [158, 40].

**Trace Distribution Equivalence.** Trace distributions quantify the probability to observe traces in the presence of non-determinism. They are thus a fundamental perspective to relate two automata, i.e. we equate two pIOTSs iff they comprise the same trace distributions. To be more specific: A trace distribution of a pIOTS  $\mathcal{I}$  is said to be contained in the set of trace distributions of a pIOTS  $\mathcal{S}$ , if there exists a scheduler of  $\mathcal{S}$  such that all traces of  $\mathcal{S}$  get assigned the same probability as in  $\mathcal{I}$ .

We formally overload  $\text{trd}(\mathcal{I}, k)$  to denote the set of trace distributions that are based on a scheduler of length  $k \in \mathbb{N}$ . We write  $\text{trd}(\mathcal{I})$  for the set of all finite trace distributions of the pIOTS  $\mathcal{I}$ . Lastly, we write  $\mathcal{I} \sqsubseteq_{TD}^k \mathcal{S}$ , if  $\text{trd}(\mathcal{I}, k) \subseteq \text{trd}(\mathcal{S}, k)$  for  $k \in \mathbb{N}$  and  $\mathcal{I} \sqsubseteq_{TD}^{\text{fin}} \mathcal{S}$  if  $\mathcal{I} \sqsubseteq_{TD}^k \mathcal{S}$  for some  $k \in \mathbb{N}$ .

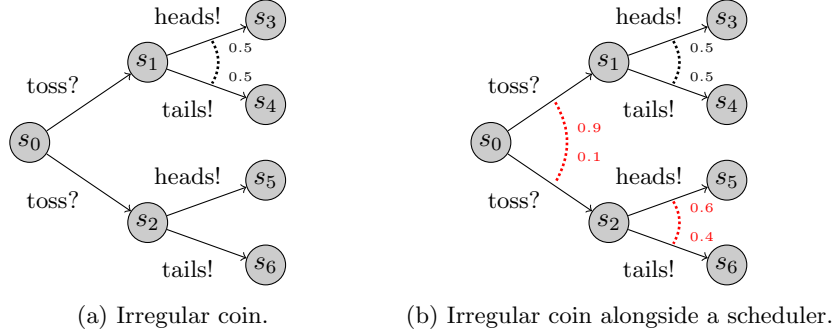


Figure 4.7: The model of an irregular coin toss. The left-hand side shows the model without any augmentation. Note, that there is a non-deterministic choice between the two outgoing *toss?* actions in  $s_0$ . The outcome of this choice results in either tossing a fair coin, or a non-deterministic one. The right-hand side augments the model with an adversary that resolves non-determinism, thus resulting in a purely probabilistic structure.

**Example 4.12.** Consider the example shown in Figure 4.7. The left side shows the model  $\mathcal{S}$  of a coin toss. There is a non-deterministic choice between the outgoing *toss?* actions in  $s_0$ , resulting in either tossing a fair coin, or a non-deterministic coin. Due to the presence of non-determinism in the model, it is not possible to directly assign probabilities to traces.

The right side augments the model with a scheduler  $\mathcal{A} \in \text{Sched}(\mathcal{S}, 2)$ . The probabilistic choices of the scheduler are illustrated in red. In particular we have

$$\begin{aligned} \mathcal{A}(s_0, \mu_1, \text{toss?}, s_1) &= 0.9 & \mathcal{A}(s_0, \mu_1, \text{toss?}, s_1, \mu_3, \text{heads!}, s_5) &= 0.6 \\ \mathcal{A}(s_0, \mu_2, \text{toss?}, s_2) &= 0.1 & \mathcal{A}(s_0, \mu_1, \text{toss?}, s_1, \mu_4, \text{tails!}, s_6) &= 0.4, \end{aligned}$$

where  $\mu_i$  with  $i = 1, 2, 3, 4$  are the unique distributions of  $\mathcal{S}$ . Thus, the scheduler chooses to toss the fair coin in 90% of the cases, and the non-deterministic coin is chosen in 10% of the cases. If the non-deterministic coin is chosen, it resolves the toss with an outcome of 60% heads and 40% tails.

This construction enables us to assign probabilities to traces. Let

$$\begin{aligned} \pi_1 &= s_0, \mu_1, \text{toss?}, s_1, \nu, \text{heads!}, s_3, \text{ and} \\ \pi_2 &= s_0, \mu_2, \text{toss?}, s_2, \mu_3, \text{heads!}, s_5. \end{aligned}$$

Then, obviously  $P_{\mathcal{A}}[C_{\pi_1}] = 0.5 \cdot 0.9 = 0.45$  and  $P_{\mathcal{A}}[C_{\pi_2}] = 0.1 \cdot 0.6 = 0.06$ . Moreover,  $\sigma = \text{tr}(\pi_1) = \text{tr}(\pi_2)$ , and consequently, for the trace distribution of the scheduler  $\mathcal{A}$ , i.e.  $\mathcal{D} = \text{trd}(\mathcal{A})$ , we see  $P_{\mathcal{D}}(\sigma) = P_{\mathcal{A}}[C_{\pi_1}] + P_{\mathcal{A}}[C_{\pi_2}] = 0.51$ .

Thus, the pIOTS  $\mathcal{S}$  alongside scheduler  $\mathcal{A}$  defines a slightly biased coin.

**Remark 4.13** (Compositionality). Working in a model-based testing environment naturally requires our models to be open. That is, we distinguish actions

based on their corresponding label set of inputs, and outputs. This, as opposed to closed systems, enables various modelled components to interact with each other or their environment. Defining schedulers for open systems is inherently more complex over their closed counterparts. The schedulers of Definition 4.6 choose to schedule inputs, as well as outputs. The authors of [39] regard this notion as too powerful and choose to define schedulers for open systems as the combination of an input scheduler, an output scheduler and an arbiter scheduler assigning precedence to one or the other.

The reasoning is as follows; Our trace distribution semantics are not compositional [164], i.e. it generally does not hold that  $\text{trd}(\mathcal{A}) \subseteq \text{trd}(\mathcal{B})$  implies  $\text{trd}(\mathcal{A} \parallel \mathcal{C}) \subseteq \text{trd}(\mathcal{B} \parallel \mathcal{C})$ . Thus, instead of the compose-and-schedule approach used in this thesis, the authors of [39] present a schedule-and-compose formalism. This circumvents the arising hurdles of non-compositionality.

While their framework offers a cleaner, and perhaps more intuitive solution, it comes at the price of increased complexity. In the remainder of this chapter, we shall see that non-compositionality of trace distribution semantics does not impose additional difficulties to deploy our techniques. In particular, we compare the model of a system under test  $\mathcal{I} \parallel t$  to a specification model  $\mathcal{S}$ , as opposed to the model of  $\mathcal{S} \parallel t$ . Moreover, we shall see that the structure of test cases, and the input enabledness of implementations prevents non-compositionality to become a problem. We thus opt for the simpler definition of schedulers in Definition 4.6 to establish an MBT framework for probabilistic systems.

## 4.2 Probabilistic Testing Theory

Model-based testing entails the automatic test case generation, execution and evaluation based on a requirements model. Its ulterior goal is to judge the correctness, or conformance, of an implementation to its requirements automatically. Recall that the MBT approach operates under the assumption that every physical implementation has an underlying formal model. Conformance is then established on a purely mathematical level via testing relations of the implementation model with respect to the specification model. Hence, one of the fundamental concepts to develop an MBT framework is the formal notion of conformance. We do so by conservatively extending **io** theory from Chapter 3 to account for probabilistic systems.

With the conformance relation in place, we proceed to develop methods to infer the inner workings of black-box implementations via experiments, or test cases. We define test cases to be pIOTSs, that only make use of Dirac distributions, i.e. test cases are IOTSs. Further, we proceed by showing how test cases are formally executed on an implementation.

Working in a probabilistic setting requires the test evaluation to be twofold: Functional aspects are evaluated as in the IOTS setting: Observed output is compared to expected output. Probabilistic aspects require a sampling process to take place on which further statistical analysis of trace frequencies is performed. Here, hypothesis testing in the form of Pearson's  $\chi^2$  test is utilized. Lastly, we



present the verdicts used to judge correctness of a system, before we prove the framework to be correct.

#### 4.2.1 The Conformance Relation $\sqsubseteq_{pioco}$

The **pioco** conformance relation is built on the theoretical foundations of **io** theory, cf. Section 3.2, which has established itself as one of the core pillars of testing relations. By conservatively extending upon this established relation, we are guaranteed to keep desirable features like implementation freedom and underspecification. The classical **io** relation (Definition 3.7) states that an implementation conforms to the requirements, if it never provides any unspecified output or quiescence. Thus, the set of output actions after a trace of an implementation should be contained in the set of output actions after the same trace of the specification. However, instead of checking for all possible traces, the **io**-relation only checks traces that were specified in the requirements.

Formally, let  $out_{\mathcal{I}}(\sigma)$  be the set of output actions of an IOTS  $\mathcal{I}$  enabled in the states after trace  $\sigma$ . Then, for two IOTSs  $\mathcal{I}$  and  $\mathcal{S}$ , with  $\mathcal{I}$  input-enabled, we write  $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$ , if and only if

$$\forall \sigma \in traces^{fin}(\mathcal{S}) : out_{\mathcal{I}}(\sigma) \subseteq out_{\mathcal{S}}(\sigma). \quad (4.1)$$

Our intention is to generalize this relation to probabilistic systems. Intuitively, we replace the set of finite traces in (4.1) with the set of finite trace distributions of  $\mathcal{S}$ , and the operator *out* by a trace distribution equivalent. Precisely, we require two auxiliary concepts:

1. The prefix relation for trace distributions  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$  is the analogue of trace prefixes, i.e.  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$  iff  $\forall \sigma \in Act^{<k} : P_{\mathcal{D}}(\sigma) = P_{\mathcal{D}'}(\sigma)$ .
2. The *output continuation* trace distributions; these are the probabilistic counterpart of the set  $out_{\mathcal{S}}(\sigma)$ . For a pIOTS  $\mathcal{S}$  and a trace distribution  $\mathcal{D}$  of length  $k$ , the output continuation of  $\mathcal{D}$  in  $\mathcal{S}$  contains all trace distributions  $\mathcal{D}'$  of length  $k + 1$ , such that  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$ , that assign every trace of length  $k + 1$  ending in input probability 0. We set

$$outcont_{\mathcal{S}}(\mathcal{D}) \stackrel{\text{def}}{=} \left\{ \mathcal{D}' \in trd(\mathcal{S}, k + 1) \mid \mathcal{D} \sqsubseteq_k \mathcal{D}' \wedge \forall \sigma \in Act^k Act_I : P_{\mathcal{D}'}(\sigma) = 0 \right\}.$$

Intuitively, an implementation should conform to a specification, if the probability of every trace in  $\mathcal{I}$  specified in  $\mathcal{S}$ , can be matched. Just like in **io**, we neglect unspecified traces ending in input actions. However, if there is unspecified output in the implementation, there is at least one scheduler that assigns positive probability to this continuation, thus violating the subset relation.

**Definition 4.14.** *Let  $\mathcal{I}$  and  $\mathcal{S}$  be two pIOTSs over the same action signature. Furthermore let  $\mathcal{I}$  be input-enabled, then we say  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$ , if for all  $k \in \mathbb{N}$*

$$\forall \mathcal{D} \in trd(\mathcal{S}, k) : outcont_{\mathcal{I}}(\mathcal{D}) \subseteq outcont_{\mathcal{S}}(\mathcal{D}).$$

**Example 4.15.** In Figure 4.8 we present six toy examples to illustrate the **pioco** relation. Note that the three upper examples do not utilize a probabilistic transition other than the Dirac distribution over the target state. They can therefore be considered as regular IOTSs. The three lower models utilize probabilistic transitions, e.g., there is a probabilistic choice between `song1!` and `song2!` with probability 0.5 each in  $\mathcal{S}_4$ . For the sake of readability in the figure, if an input is not enabled in a state of the implementation, assume that we make it input-enabled by adding a self-loop.

The original **ioco** relation checks whether the output of an implementation, after a specified trace, was expected. To illustrate,  $\mathcal{S}_1$  is **ioco** to both  $\mathcal{S}_2$  and  $\mathcal{S}_3$ . The input `shuf?` yields the output `song1!`, which is a subset of what was specified by the latter two, i.e.  $\{\text{song1!}, \text{song2!}\}$ . In particular, note that it is irrelevant if the non-deterministic choice is over the `shuf?` actions, or the output actions, cf. Figures 4.8b and 4.8c.

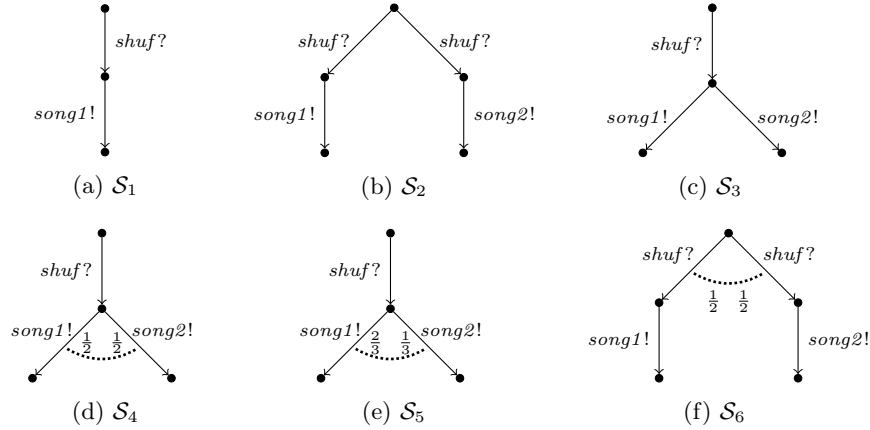


Figure 4.8: Yardstick examples of a simplified shuffle player illustrating **pioco**. The three upper models represent regular IOTSs, while the three lower examples are pIOTSs. See Example 4.15 for more details.

The **pioco** relation combines probabilistic and non-deterministic choices.  $\mathcal{S}_4$  and  $\mathcal{S}_5$  are not **pioco** for using different probabilities attached to the output actions. However, it is  $\mathcal{S}_4 \sqsubseteq_{\text{pioco}} \mathcal{S}_3$ . The requirements specification indicates a choice over `song1!` and `song2!`. If a system implements this choice with a 0.5 choice over the action, there is a scheduler in the specification that assigns exactly those probabilities to the actions `song1!` and `song2!`. Note that the opposite direction does not hold, because any implementation would need to assign probabilities 0.5 to each action, while the non-determinism indicates a free choice of probabilities according to our assumptions.

An intricate example is given in the relation of  $\mathcal{S}_4$  and  $\mathcal{S}_6$ . In particular, it is  $\mathcal{S}_4 \sqsubseteq_{\text{pioco}} \mathcal{S}_6$ , but  $\mathcal{S}_6 \not\sqsubseteq_{\text{pioco}} \mathcal{S}_4$ . Even though, on a surface level, it looks like all traces can get assigned the same probability under a scheduler, it is the probability of halting that the mismatch stems from. Before scheduling either

pIOTS	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$
$\mathcal{S}_1$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	-	-
$\mathcal{S}_2$	-	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	-	-
$\mathcal{S}_3$	-	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	-	-
$\mathcal{S}_4$	-	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	$\sqsubseteq_{pioco}$
$\mathcal{S}_5$	-	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	$\sqsubseteq_{pioco}$	-
$\mathcal{S}_6$	-	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	-	-	$\sqsubseteq_{pioco}$

Table 4.1: Complete table of the **pioco** relation with respect to Figure 4.8. Note that pIOTSs on the left-hand side of the relation are assumed to be input enabled by adding self-loops in their corresponding models.

song in  $\mathcal{S}_6$ , a scheduler may assign different probabilities to halt in both states. A scheduler in  $\mathcal{S}_4$  must now mimic this behaviour in a single state, while scheduling the same probabilities as the other scheduler to both song1 and song2. Obviously, there are combinations of probabilities for which this is impossible.

For a complete list of the conformances in Figure 4.8 we refer to Table 4.1.

The **pioco** relation conservatively extends the **ioco** relation, i.e. both relations coincide for IOTSs. Recall that a pIOTS essentially is an input output transition system with transitions having distributions over states as target. Conversely, an IOTS can be treated as pIOTS where every distribution is the Dirac distribution, i.e. a distribution with a unique target.

**Theorem 4.16.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be two IOTSs and  $\mathcal{A}$  be input-enabled, then*

$$\mathcal{A} \sqsubseteq_{ioco} \mathcal{B} \iff \mathcal{A} \sqsubseteq_{pioco} \mathcal{B} .$$

*Proof sketch.* Conformance of pIOTSs is defined via probabilities of traces, where the probabilities are a result of schedulers and the inherent discrete probability distributions of pIOTSs. The proof follows from the fact that an IOTS does not have inherent probabilities – A scheduler can thus assign the same probabilities to all traces of both  $\mathcal{A}$  and  $\mathcal{B}$  provided they are present in both systems. The conformance relation  $\sqsubseteq_{ioco}$  ensures that relevant traces are indeed present in both  $\mathcal{A}$  and  $\mathcal{B}$ .  $\square$

A tester must be able to apply every input at any given state of the SUT. This is reflected in classic **ioco**-theory by always assuming the implementation to be input enabled (Definition 3.7). If the specification is input-enabled too, then **ioco** coincides with finite trace inclusion. We show that **pioco** coincides with finite trace distribution inclusion in the pIOTS case. Note that input-enabledness of the specification is crucial here. Moreover, our results show that **pioco** is transitive, just like **ioco**.

**Theorem 4.17.** *Let  $\mathcal{A}$ ,  $\mathcal{B}$  and  $\mathcal{C}$  be pIOTSs and let  $\mathcal{A}$  and  $\mathcal{B}$  be input-enabled, then*

$$(i) \mathcal{A} \sqsubseteq_{pioco} \mathcal{B} \text{ if and only if } \mathcal{A} \sqsubseteq_{TD}^{fn} \mathcal{B} .$$

(ii)  $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{pioco} \mathcal{C}$ , then  $\mathcal{A} \sqsubseteq_{pioco} \mathcal{C}$ .

*Proof sketch.* (i) The fact that finite trace distribution inclusion implies  $\sqsubseteq_{pioco}$  is straightforward, since  $\sqsubseteq_{pioco}$  is defined via finite trace distributions. The other direction follows immediately, if we consider that  $\sqsubseteq_{pioco}$  ensures all traces ending in output can get assigned the same probability in both  $\mathcal{A}$  and  $\mathcal{B}$ , as well as both systems being input enabled. Since distributions containing inputs are input-reactive, there cannot be mismatches in probabilities here.

(ii) arises as a consequence of (i).  $\square$

## 4.2.2 Test Cases and Test Annotations

We formalize the notion of test cases over an interface (a.k.a. action signature, or alphabet)  $(Act_I, Act_O)$ . Formally, a test case is a collection of traces that represent possible behaviour of a tester. These are summarized as a pIOTS in tree structure. The action signature describes the potential interaction of the test case with the implementation. In each state of a test, the tester can either provide some stimulus, wait for a response of the system, or stop the overall testing process. When a test is waiting for a system response, it has to take into account all potential outputs including the situation that the system provides no response at all, modelled by  $\delta$ <sup>3</sup>.

**Definition 4.18.** A test or test case over an alphabet  $(Act_I, Act_O)$  is a pIOTS

$$t = \langle S, s_0, Act_I^t, Act_O^t, \emptyset, \Delta \rangle$$

that has the alphabet's outputs as inputs and vice-versa, i.e.  $Act_I^t = Act_O \cup \{\delta\}$  and  $Act_O^t = Act_I \setminus \{\delta\}$ , and that is a finite, internally deterministic and connected tree. In addition, for all  $\mu \in \Delta$  we have  $\mu \equiv \text{Dirac}$ , and for every state  $s \in S$  we require:

- $enabled(s) = \emptyset$ ,
- $enabled(s) = Act_I^t$ , or
- $enabled(s) \in Act_O^t$ .

A test suite  $T$  is a set of test cases. A test case (suite resp.) for a pIOTS  $\mathcal{S} = \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle$ , is a test case (suite resp.) over its alphabet  $(Act_I, Act_O)$ , and if item 2 additionally requires, that

- $\mu(a, s') > 0$  with  $a \in Act_O^t$  implies the existence of  $\sigma \in \text{traces}^{fin}(\mathcal{S})$ , such that  $\sigma a \in \text{traces}^{fin}(\mathcal{S})$ .

For technical reasons, we swap the input and output label sets of a test case. This is to allow for synchronization in the context of parallel composition. In the IOTS case of Chapter 3, this was resolved with the *mirroring* operator, i.e.

<sup>3</sup>Note that in more recent version of *ioco* theory [174], test cases are input-enabled. This enables them to catch possible outputs of the implementation before the test supplied the input. We consciously chose not to incorporate this into our framework for technical reasons.

for an IOTS  $\mathcal{I}$  the IOTS *mirror* ( $\mathcal{I}$ ) switches all its inputs to outputs, and vice versa. However, mirroring an output-generative distribution does not yield an input-reactive distribution, and thus produces an invalid pIOTS. We refer to Figure 4.4 to illustrate this. Since we are only interested in the mirroring of test cases, which only allow the Dirac distribution (i.e. single target), we chose to incorporate mirroring in their definition directly here.

Lastly, we give an intuitive understanding of test cases *for* specifications. In addition to being defined over the same action signature, we require that only inputs are given, which are presents in the specification model. This is referred to as *input-minimal* in the literature [169]. Giving unavailable inputs would leave us in a position, in which we are unable to give a verdict according to **io**co. That is, we may only judge the correctness of specified behaviour, as is inherent in testing.

**Remark 4.19.** *Informally, we define test cases as IOTSSs, as opposed to pIOTSSs. Our earlier work used discrete probability choices in test cases [73, 76], thus paving the way for probabilistic test cases. This fills a gap in that MBT tools frequently use random number generators to resolve the choice of observing or stimulating the system, and the concrete input is often chosen according to a uniform distribution [15]. Using discrete probability distributions in test cases quantifies these choices. We opted to not incorporate these properties in Definition 4.18 to make a distinction between test cases and test selection. The latter is mentioned in the test generation algorithms discussed in the next section.*

**Test Annotation.** In order to pin down the behaviour, which we deem as acceptable/correct, each trace of the test is annotated with *pass* or *fail* verdict, determined by the requirements specification. This allows for automated evaluation of the *functional behaviour* later on. The probabilistic verdict is given via a second step, that requires gathering a large sample of test executions.

We have shown that the **pioco** relation conservatively extends the **io**co one in Theorem 4.16. Since the purpose of annotations is to check for functional correctness only, we directly transfer the annotation of Definition 3.12. Informally, a trace of a test case is labelled as *pass*, if it is present in the system specification and *fail* otherwise.

**Definition 4.20.** *For a given test  $t$  a test annotation is a function*

$$ann : traces^{com}(t) \longrightarrow \{pass, fail\}.$$

A pair  $\hat{t} = (t, ann)$  consisting of a test and a test annotation is called an annotated test. The set of all such  $\hat{t}$ , denoted by  $\hat{T} = \{(t_i, ann_i)_{i \in \mathcal{I}}\}$  for some index set  $\mathcal{I}$ , is called an annotated test suite. If  $t$  is a test for a specification  $\mathcal{S}$  with signature  $(Act_I, Act_O)$ , we define  $ann_{pioco}^{\mathcal{S}} : traces^{com}(t) \longrightarrow \{pass, fail\}$  by

$$ann_{pioco}^{\mathcal{S}}(\sigma) = \begin{cases} fail & \text{if } \exists \varrho \in traces^{fn}(\mathcal{S}), a \in Act_O : \\ & \varrho a \sqsubseteq \sigma \wedge \varrho a \notin traces^{fn}(\mathcal{S}) \\ pass & \text{otherwise.} \end{cases}$$

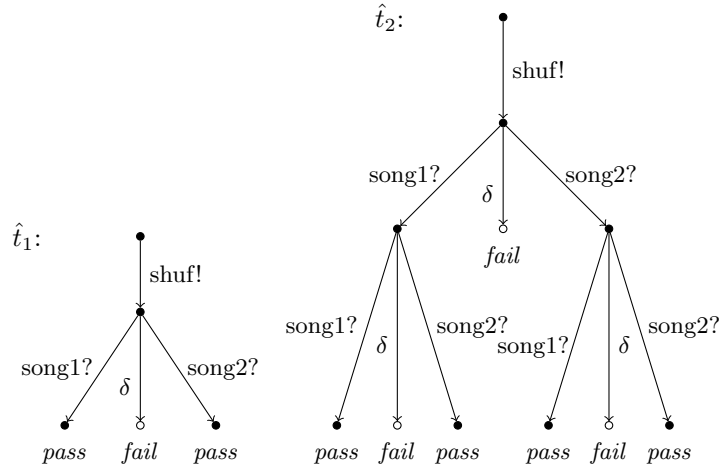


Figure 4.9: Two annotated test cases for the specification of the shuffle music player of Figure 4.5.  $\hat{t}_1$  tests that a song is played after the shuffle button was pressed.  $\hat{t}_2$  ensures that the player keeps on playing music, and does not stop after one song.

**Example 4.21.** Figure 4.9 shows two simple derived tests for the specification of a shuffle music player in Figure 4.5. Note that the interface is mirrored. This is to allow for synchronisation on shared actions according to the definition of parallel composition (Definition 4.5). Outputs of the test case are considered inputs for the implementation and vice versa.

The left model in Figure 4.9 presents an annotated test case  $\hat{t}_1$ . After the shuffle button is pressed, the test waits for a system response. Catching either song assigns the pass verdict, while the absence of outputs, denoted by  $\delta$ , yields the fail verdict. The test  $\hat{t}_2$  on the right-hand side ensures that the player does not stop after one song, but continues to play songs. This is indicated by the pass labels on either song transition, and a fail verdict on the  $\delta$  transitions.

We point out that test annotations do not infer the probabilistic correctness of the implementation. This requires multiple executions of the same test case to gather a sample. Probabilistic correctness is inferred via a statistical hypothesis test and is studied in the next subsection.

### 4.2.3 Test Evaluation and Verdicts

In our framework, we assess functional correctness via the test annotation *ann* precisely how described in Chapter 3. Therefore, this step is not further described here. However, the addition of discrete probability choices on specification level requires further statistical analysis. We present how an implementation can effectively be checked for the correct application of probabilities.

We perform a statistical experiment with various parameters, including

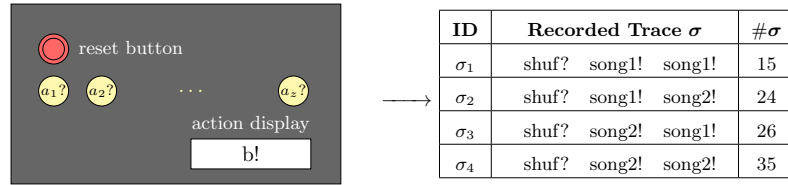


Figure 4.10: Black-box trace machine with input alphabet  $a_1?, \dots, a_z?$ , reset button and action window. Running the machine  $m$  times and observing traces of length  $k$  yields a sample. The ID together with the trace and the respective number of occurrences are noted down in a sample like on the right-hand side.

predefined sample size and level of significance. The requirements specification in tandem with a scheduler defines *expected values* for the frequency of each trace. We proceed by defining sets of possible observations on implementation side. The level of significance then sets a maximum of allowed deviation from the expectation. Any observation within this threshold is deemed acceptable, while observations outside this boundary are rejected.

**Statistical Testing.** In order to reason about probabilistic correctness, a single test execution is insufficient. To illustrate, assume we want to establish the fairness of a coin; A single coin flip is inadequate to draw a conclusion. To overcome this hurdle, we collect a sample via multiple test runs, and perform statistical analysis on it. We refer the reader to Appendix A.2 for the details of statistical testing required in this thesis.

The sampling process consists of a push-button experiment in the sense of [134]. Assume a black-box trace machine is given with input buttons, an observation window, and a reset button as illustrated in Figure 4.10. An external observer records each individual execution before the reset button is pressed and the machine starts again. Note that the observations of an external observer match the notion of *traces*. After a sample of sufficient size was collected, we compare the collected frequencies of traces to their expected frequencies according to the requirements specification. If the empiric observations are close to the expectations, we accept the probabilistic behaviour of the implementation.

**Sampling.** Before the start of the sampling process, or any statistical experiments, we need to set the parameters. We require sample length  $k \in \mathbb{N}$ , sample size  $m \in \mathbb{N}$ , and a level of significance  $\alpha \in (0, 1)$ . In the experimental setting utilized here, this translates to choosing the length of individual traces, how many traces should be observed, and a limit for the statistical *error of first kind*, i.e. the probability of rejecting a correct implementation.

To establish probabilistic correctness, we check if the frequencies of the traces contained in the sample match the probabilities in the specification via statistical hypothesis testing. However, statistical methods can only be directly applied for purely probabilistic systems without non-determinism. To illustrate, recall

Figure 4.7a; The inherent non-deterministic choice between the two *toss?* actions in  $s_0$ , and the non-deterministic coin flip in state  $s_2$  prevents us from directly assigning a probability to trace, say, *toss? head!*.

To overcome this, we check if the observed trace frequencies can be explained, if we resolve non-determinism in the specification according to *some* scheduler. In other words, our hypothesis is the existence of a scheduler that makes the occurrence of the sample likely.

Thus, we assume that each execution of the black-box implementation  $\mathcal{I}$  is governed by an unknown scheduler, resulting in a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{I})$ . In order for any statistical reasoning to work, we assume that  $\mathcal{D}$  is the same in every run. Consequently, the implementation chooses a trace distribution  $\mathcal{D}$  and  $\mathcal{D}$  chooses a trace  $\sigma$  to execute.

**Frequencies and Expectations.** To quantify how *close* a sample is to its expectation, we require a notion of *distance*. Our goal is to evaluate the deviation of a collected sample to the expected distribution. Therefore, let  $\mathcal{M} = (\text{Distr}, \text{dist})$  be the metric space, where

$$\text{dist}(u, v) \stackrel{\text{def}}{=} \sup_{\sigma \in \text{Act}^{\leq k}} |u(\sigma) - v(\sigma)|$$

is the maximal variation distance of two distributions. It is easy to check that *dist* is indeed a metric, and consequently  $\mathcal{M}$  defines a metric space.

We proceed by defining the two distributions that need to be compared; The *empiric distribution* of a sample, and the *expected distribution* based on the specification. The function assessing the frequencies of traces within a sample  $O = \{\sigma_1, \dots, \sigma_m\}$  is given as a mapping  $\text{freq} : \text{Act}^{\leq k \times m} \rightarrow \text{Distr}(\text{Act}^{\leq k})$ , such that

$$\text{freq}(O)(\sigma) = \frac{|\{i=1, \dots, m \wedge \sigma = \sigma_i\}|}{m}.$$

Hence, the discrete distribution *freq* gives the relative frequency of a trace within a sample of size  $m$ .

To calculate the expected distribution according to a specification, we need to resolve all non-deterministic choices to get a purely probabilistic execution tree. Therefore, assume that a trace distribution  $\mathcal{D}$  is given, and  $k$  and  $m$  are fixed. We treat each run of the implementation as a Bernoulli trial. Recall that a Bernoulli trial has two outcomes: *success* with probability  $p$  and *failure* with probability  $1 - p$ . For any trace  $\sigma$ , we say that success occurred at position  $i$  of the sample, if  $\sigma = \sigma_i$ . Therefore, let  $X_i \sim \text{Ber}(P_{\mathcal{D}}(\sigma))$  be Bernoulli distributed random variables for  $i = 1, \dots, m$ . Let  $Z = \frac{1}{m} \sum_{i=1}^m X_i$  be the empiric mean with which we observe  $\sigma$  in a sample. Note that the expected probability under  $\mathcal{D}$  is then calculated as

$$\mathbb{E}^{\mathcal{D}}(Z) = \mathbb{E}^{\mathcal{D}}\left(\frac{1}{m} \sum_{i=1}^m X_i\right) = \frac{1}{m} \sum_{i=1}^m \mathbb{E}^{\mathcal{D}}(X_i) = P_{\mathcal{D}}(\sigma).$$

Hence, the expected probability for each trace  $\sigma$ , is the probability of  $\sigma$  under trace distribution  $\mathcal{D}$ .



**Example 4.22.** *The right hand side of Figure 4.10 shows a potential sample  $O$  that was collected from the shuffle music player of Figure 4.5a when  $\hat{t}_2$  of Figure 4.9 is executed. The sample consists of  $m = 100$  traces of length  $k = 3$ . In total there are four different traces with varying frequencies. We calculate the relative frequencies of the four different traces as*

$$\begin{aligned} \text{freq}(O)(\sigma_1) &= \frac{15}{100} & \text{freq}(O)(\sigma_2) &= \frac{24}{100} \\ \text{freq}(O)(\sigma_3) &= \frac{26}{100} & \text{freq}(O)(\sigma_4) &= \frac{35}{100} \end{aligned}$$

*Together, these frequencies form the empiric sample distribution.*

*Conversely, assume there is a scheduler, that schedules  $\text{shuf?}$  with probability 1 and the distribution consisting of  $\text{song1!}$  and  $\text{song2!}$  with probability 1 in Figure 4.5a. This scheduler then induces a trace distribution  $\mathcal{D}$  on the pIOTS of the shuffle-player. The expected probability of the observed traces under this trace distribution then calculates as  $\mathbb{E}^{\mathcal{D}}(\sigma_i) = 1 \cdot 1 \cdot 0.5 \cdot 0.5 = 0.25$  for  $i = 1, \dots, 4$ .*

*The question we want to answer in practice is, whether there exists a scheduler, such that the empiric sample distribution is sufficiently close to the expected distribution.*

**Acceptable Outcomes.** The intuitive idea is to compare the sample frequency function to the expected distribution. If the observed frequencies do not deviate significantly from our expectations, we accept the sample. How much deviation is allowed depends on the *a priori* chosen  $\alpha \in (0, 1)$ .

We accept a sample  $O$  if  $\text{freq}(O)$  lies within some distance  $r_\alpha$  of the expected distribution  $\mathbb{E}^{\mathcal{D}}$  of the metric space  $\mathcal{M}$ . Recall the definition of a closed ball in a metric space centred at  $x \in \mathcal{M}$  with radius  $r$  as  $B_r(x) = \{y \in \mathcal{M} \mid \text{dist}(x, y) \leq r\}$ . All distributions deviating at most  $r_\alpha$  from the expected distribution are contained within the ball  $B_{r_\alpha}(\mathbb{E}^{\mathcal{D}})$ . To limit the error of accepting an erroneous sample, we choose the smallest radius, such that the error of rejecting a correct sample is not greater than  $\alpha$  by

$$r_\alpha \stackrel{\text{def}}{=} \inf \{r \in \mathbb{R}_0^+ \mid P_{\mathcal{D}}(\text{freq}^{-1}(B_r(\mathbb{E}^{\mathcal{D}}))) \geq 1 - \alpha\}.$$

**Definition 4.23.** *For  $k, m \in \mathbb{N}$  and a pIOTS  $\mathcal{S}$  the acceptable outcomes of  $\mathcal{D} \in \text{trd}(\mathcal{S}, k)$  of significance level  $\alpha \in (0, 1)$  are given by the set*

$$\text{Obs}(\mathcal{D}, \alpha, k, m) = \left\{ O \in (\text{Act}^{\leq k \times m}) \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha \right\}.$$

*We obtain the set of acceptable outcomes of  $\mathcal{S}$  by*

$$\text{Obs}(\mathcal{S}, \alpha, k, m) = \bigcup_{\mathcal{D} \in \text{trd}(\mathcal{S}, k)} \text{Obs}(\mathcal{D}, \alpha, k, m).$$

The set of acceptable outcomes consists of all possible samples we are willing to accept as *close enough* to our expectations, i.e. all samples that are likely

under consideration of *all* trace distributions. Note that, due to non-determinism, the latter is required to define expected values in the first place.

The set of acceptable outcomes of a pIOTS  $\mathcal{A}$  therefore has two properties, reflecting the error of false rejection and false acceptance respectively.

1. If a sample was generated by a truthful trace distribution of the specification, we correctly accept it with probability higher than  $1 - \alpha$ , i.e.

$$P_{\mathcal{D}}(\text{Obs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha.$$

2. If a sample was generated by a trace distribution not admitted by the specification, the chance of falsely accepting it is smaller than some  $\beta_m$ .

Here  $\alpha$  is the predefined level of significance, and  $\beta_m$  is unknown, but minimal by construction. Note that  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ , thus the error of falsely accepting an observation decreases with increasing sample size, as is to be expected.

**Remark 4.24.** *The set of acceptable outcomes comprises samples of the form  $O \in \text{Act}^{\leq k \times m}$ . In order to align observations with the **pioco** relation, we define the set of acceptable output outcomes like follows*

$$\text{OutObs}(\mathcal{D}, \alpha, k, m) = \left\{ O \in (\text{Act}^{\leq k-1} \text{Act}_O)^m \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_{\alpha} \right\}.$$

**Verdict Functions.** With this framework, the following decision process summarizes if an implementation fails a test, or test suite respectively, based on a functional and/or statistical verdict. An overall *pass* verdict is given to an implementation if and only if it passes both verdicts.

**Definition 4.25.** *Given a pIOTS  $\mathcal{S}$ , an annotated test  $\hat{t}$  for  $\mathcal{S}$ ,  $k, m \in \mathbb{N}$  where  $k$  is given by length of the longest trace of  $\hat{t}$ , and  $\alpha \in (0, 1)$ , we define the functional verdict as  $v_{\text{func}} : \text{pIOTS} \times \text{pIOTS} \rightarrow \{\text{pass}, \text{fail}\}$ , with*

$$v_{\text{func}}(\mathcal{I}, \hat{t}) = \begin{cases} \text{pass} & \text{if } \forall \sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t}) : \text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{pass} \\ \text{fail} & \text{otherwise,} \end{cases}$$

*the statistical verdict as the function  $v_{\text{stat}} : \text{pIOTS} \times \text{pIOTS} \rightarrow \{\text{pass}, \text{fail}\}$ , with*

$$v_{\text{prob}}(\mathcal{I}, \hat{t}) = \begin{cases} \text{pass} & \text{if } \forall \mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k) \exists \mathcal{D}' \in \text{trd}(\mathcal{S}, k) : \\ & P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha \\ \text{fail} & \text{otherwise,} \end{cases}$$

*and the overall verdict as the function  $V : \text{pIOTS} \times \text{pIOTS} \rightarrow \{\text{pass}, \text{fail}\}$ , with*

$$V(\mathcal{I}, \hat{t}) = \begin{cases} \text{pass} & \text{if } v_{\text{func}}(\mathcal{I}, \hat{t}) = v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass} \\ \text{fail} & \text{otherwise.} \end{cases}$$

*An implementation passes a test suite  $\hat{T}$ , if it passes all tests  $\hat{t} \in \hat{T}$ .*

The functional verdict is based on the test annotations (Definition 4.20). The execution of a test on the implementation is denoted by their parallel composition. Note that this verdict is similar to the one of Chapter 3 for LTSs.

The statistical verdict is based on a sample. Therefore a test case has to be executed several times to gather a sufficiently large sample. A *pass* verdict is given, if the observation is likely enough under a fitting trace distribution. If no suitable trace distribution exists, the observed behaviour cannot be explained by the requirements specification and the *fail* verdict is given.

Lastly, only if an implementation passes both the functional and statistical test verdicts, it is given the overall verdict *pass*.

#### 4.2.4 Correctness of the Framework

Talking about soundness and completeness when referring to probabilistic systems is not a trivial topic, since one of the main difficulties of statistical analysis is the possibility of *false rejection*, or *false acceptance*, respectively. This means that the application of null hypothesis testing inherently includes the possibilities to erroneously reject a true hypothesis, or to falsely accept an invalid one by chance. We refer to Appendix A.2 for more details.

The former is of interest when we refer to soundness, i.e. what is the probability that we erroneously assign *fail* to a conforming implementation. The latter is important when we talk about completeness, i.e. what is the probability that we assign *pass* to a non-conforming implementation. Thus, a test suite can only fulfil these properties with a guaranteed (high) probability, as reflected in the verdicts we assign, cf. Definition 4.25.

**Definition 4.26.** *Let  $\mathcal{S}$  be a pIOTS over an action signature  $(Act_I, Act_O)$ ,  $\alpha \in (0, 1)$  be the level of significance and  $\hat{T}$  an annotated test suite for  $\mathcal{S}$ . Then*

- $\hat{T}$  is sound for  $\mathcal{S}$  with respect to  $\sqsubseteq_{pioco}$ , if for all input-enabled implementations  $\mathcal{I} \in pIOTS$  and sufficiently large  $m \in \mathbb{N}$  it holds for all  $\hat{t} \in \hat{T}$

$$\mathcal{I} \sqsubseteq_{pioco} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = pass.$$

- $\hat{T}$  is complete for  $\mathcal{S}$  with respect to  $\sqsubseteq_{pioco}$ , if for all input-enabled implementations  $\mathcal{I} \in pIOTS$  and sufficiently large  $m \in \mathbb{N}$ , there is at least one  $\hat{t} \in \hat{T}$ , such that

$$\mathcal{I} \not\sqsubseteq_{pioco} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = fail.$$

Even though the parameters  $k, m$  and  $\alpha$  are not explicitly present in the overall verdict function  $V$ , they are implicitly used by the statistical verdict  $v_{stat}$ , cf. Definition 4.25. They were left out here for readability.

Soundness for a given  $\alpha \in (0, 1)$  expresses that we have a  $1 - \alpha$  chance that a correct system will pass the annotated suite. This relates to false rejection of a correct hypothesis or correct implementation respectively.

**Theorem 4.27.** *Each annotated test for a pIOTS  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{pioco}$ .*

*Proof sketch.* We assume  $\mathcal{I}$  to be an input enabled pIOTS and  $\hat{t}$  to be an annotated test for  $\mathcal{S}$ . The statement is proven by showing that  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{t}) = pass$ , which entails to show that both the functional and the probabilistic verdict yield *pass*.

- Functional correctness requires that all  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$  have the *pass* annotation. This is shown by considering a prefix  $\sigma'$  of  $\sigma$  in the finite traces of  $\mathcal{S}$ . The intuition is showing that  $\sigma'a$  is a trace in  $\mathcal{S}$  for all outputs  $a$  enabled after  $\sigma'$  in  $\mathcal{I}$ . There is a trace distribution  $\mathcal{D}$  of  $\mathcal{S}$  that assigns  $\sigma'$  positive probability. Without loss of generality, we choose a trace distribution in  $outcont_{\mathcal{I}}(\mathcal{D})$  that assigns  $\sigma'a$  positive probability. Together with the  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$  assumption, this shows that  $\sigma'a$  gets assigned positive probability in  $\mathcal{S}$  under this trace distribution. Hence  $\sigma'a$  is in the finite traces of  $\mathcal{S}$  and consequently gets assigned the *pass* annotation. Since this holds for all such prefixes  $\sigma'$  this yields  $v_{func}(\mathcal{I}, \hat{t}) = pass$ .
- Probabilistic correctness requires that all observations of  $\mathcal{I} \parallel \hat{t}$  get assigned a measure greater or equal to  $1 - \alpha$  for a trace distribution of  $\mathcal{S}$ , i.e. they are acceptable outcomes of  $\mathcal{S}$ . The proof encompasses to choose  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, k)$ , and showing that then also  $\mathcal{D} \in trd(\mathcal{S})$ . This is sufficient by merit of the definition of observations (Definition 4.23), i.e. we always have  $P_{\mathcal{D}}(Obs(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$  for any  $\mathcal{D}$ .

This is done in three steps 1.  $\mathcal{D}$  might still schedule positive probability to input actions in the  $k$ -th step; we choose a new scheduler that assigns all this probability mass to halting instead. Note that the measure of *OutObs* is unaffected by this change, since it comprises traces ending in outputs only. 2. We show that  $\mathcal{D}$  is a trace distribution of  $trd(\mathcal{I})$ . Intuitively,  $\mathcal{I} \parallel \hat{t}$  is internally deterministic by the construction of test cases. Hence, there is an injective mapping from paths of  $\mathcal{I} \parallel \hat{t}$  to paths of  $\mathcal{I}$ . We thus construct a scheduler in  $\mathcal{I}$  that copies the behaviour of the scheduler of  $\mathcal{I} \parallel \hat{t}$  step-by-step. Lastly, 3. we apply the assumption  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$  to show  $\mathcal{D} \in trd(\mathcal{S})$ . Finally, this yields  $v_{prob}(\mathcal{I}, \hat{t}) = pass$ .

Since both sub-verdicts were shown to yield *pass* we conclude that the overall verdict is *pass* i.e. all tests for  $\mathcal{S}$  are sound with respect to  $\sqsubseteq_{pioco}$ .  $\square$

Completeness of a test suite is inherently a theoretical result. Since we allow loops, we require a test suite of infinite size. Moreover, there is the chance of falsely accepting an erroneous implementation. However, this is bound from above by construction, and decreases for bigger sample sizes, cf. Definition 4.23.

**Theorem 4.28.** *The set of all annotated test cases for a specification  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{pioco}$  for sufficiently large sample size.*

*Proof sketch.* We assume  $\mathcal{I}$  to be an input enabled pIOTS and  $\hat{T}$  to be the test suite containing *all* annotated test cases for  $\mathcal{S}$ . The statement is proven by showing that  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = \text{fail}$ , i.e. there is a test case  $\hat{t}$  such that either the functional, or the probabilistic verdict fails. Assuming  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$  implies the existence of a trace ending in output in  $\mathcal{I}$ , such that its probability cannot be matched under any trace distribution in  $\mathcal{S}$ . Generally, this can have two causes: 1. the mismatch arises due to the fact that the trace is simply not present in  $\mathcal{S}$ , or 2. all such traces are present in  $\mathcal{S}$ , and the mismatch arises due to different inherent probability distributions in  $\mathcal{I}$  and  $\mathcal{S}$ . We show that the first implies  $v_{\text{func}}(\mathcal{I}, \hat{T}) = \text{fail}$ , and the second implies  $v_{\text{prob}}(\mathcal{I}, \hat{T}) = \text{fail}$ .

- Showing that the functional verdict yields *fail* is straightforward, and requires us to show that there is a test case for which the trace not present in  $\mathcal{S}$  has the *fail* annotation. The proof follows directly from the definition of test cases and test annotations (Definition 4.18 and Definition 4.20), and the fact that  $\hat{T}$  contains *all* test cases for  $\mathcal{S}$ .
- In order to show that the probabilistic verdict yields *fail*, we need to show that there is a test case  $\hat{t}$  and a trace distribution of  $\mathcal{I} \parallel \hat{t}$ , under which all observations get assigned a measure smaller than  $1 - \alpha$  for all trace distributions of  $\mathcal{S}$  for sufficiently large  $m$ . It is clear by the definition of acceptable outcomes (Definition 4.23) that  $P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) < \beta_m$  for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$  whenever  $\mathcal{D} \neq \mathcal{D}'$ . By the initial assumption  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$ , we know this holds for all  $\mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$  with  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$ . This estimation does not change, if we increase the search space to all  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k + 1)$  instead, as the measure of the *OutObs* set is maximised for trace distributions of *outcont*.

It remains to be shown that for a trace distribution  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$ , there is a test case  $\hat{t} \in \hat{T}$ , such that  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t})$ . By assumption, all traces getting assigned positive probability under  $\mathcal{D}$  are traces in  $\mathcal{S}$ . We thus select a test case  $\hat{t}$  containing all such traces. The IOTS structure and absence of internal actions in  $\hat{t}$  imply  $\mathcal{I} \parallel \hat{t}$  to be internally deterministic. In particular, there is no interleaving. This means, there is a scheduler in  $\mathcal{I} \parallel \hat{t}$  that copies the behaviour of the underlying scheduler of  $\mathcal{D}$  step-by-step. A careful construction thus results in  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t})$ , for which all  $\mathcal{D}' \in \text{trd}(\mathcal{S})$  assign all of  $\mathcal{D}$ 's observations a measure smaller than  $1 - \alpha$  for sufficiently large  $m$ . This yields exactly  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$ .

Since the mismatch of probability of a trace has to belong to one of the two cases, we have thus shown that  $V(\mathcal{I}, \hat{T}) = \text{fail}$ .  $\square$

### 4.3 Implementing Probabilistic Testing

The previous section laid the theoretical foundations of our pIOTS-based testing framework. Several aspects were defined rather abstractly, for which we now

provide practical procedures. That is, we link the theory of Section 4.2 to applicable methods and algorithms.

First, we recall the two algorithms of Chapter 3. There, we defined the *batch generation* algorithm, and the *on-the-fly* algorithm. We present basic alterations that instantiate the non-deterministic choices of stimulation, observation or stopping via a uniform probabilistic choice. Albeit their striking similarity, the new algorithms are subsequently proven to be correct.

While the functional evaluation solely depends on test cases and their annotations, the statistical analysis of gathered sample data in MBT is largely unexplored. Since the underlying implementation model is unknown to an external observer, we require concrete procedures to perform the statistical tests of Section 4.2. In particular, we need practical methods to conclude the probabilistic verdict of Definition 4.25. This comprises practical measures to judge deviation of a sample to its expectations. To this end, we utilize hypothesis testing, particularly Pearson's  $\chi^2$ -testing as a measure of distance. Recall that expected probabilities of a pIOTS are only meaningful in tandem with a scheduler. Hence, our interest lies in finding a well-fitting scheduler to minimise the  $\chi^2$  score, that expresses the distance of sample and expectations.

### 4.3.1 Test Generation Algorithms

We present algorithms to generate tests according to Definition 4.18; A *batch generation* algorithm and an *on-the-fly* test algorithm. The first is used to generate and store test cases *before* their execution. This is advantageous when gathering a sample, where a single test case is required to be executed several times. The latter algorithm provides inputs and catches potential system responses during run-time of the implementation. Generally, model-based testing tools possessing an on-the-fly algorithm give the option of manually setting a seed for the random number generator [15], thus reproducibility is covered in this case, too.

**Batch generation.** The recursive procedure `batchGen` in Algorithm 4.1 generates test cases according to Definition 4.18. It requires a specification pIOTS  $\mathcal{S}$ , a history  $\sigma$ , which is initially the empty history  $\epsilon$ , and a maximal length of test cases  $n \in \mathbb{N}$ . The procedure initially starts with the empty history and recursively calls itself to add additional branches to the current history. A branch is terminated, if the procedure returns the empty history  $\epsilon$ . The procedure draws a random number  $p$  from the uniform distribution over  $[0, 1]$ , presenting the choice of stimulation, observation or stopping (line 3). If  $p \in [0, 0.33]$  the procedure stops (line 3), if  $p \in (0.33, 0.66]$  the implementation is observed (line 4), and if  $p \in (0.66, 1]$  a stimulus is sent (line 13). This represents a simple probabilistic implementation of the non-deterministic batch generation algorithm studied in Chapter 3. The additional condition in line 3 ensures eventual termination upon reaching a maximal test length.

Lines 4-12 describe the step of observing the system; A branch is instantiated for every output action in  $Act_O$  (line 6). If a particular output is foreseen in the

**Algorithm 4.1:** Batch test generation for pIOTSs.

---

**Input:** Specification pIOTS  $\mathcal{S}$ , history  $\sigma \in \text{traces}^{\text{fin}}(\mathcal{S})$  and maximal test length  $n \in \mathbb{N}$ .  
**Output:** A test case  $t$  for  $\mathcal{S}$ .

```

1 Procedure batchGen( $\mathcal{S}, \sigma, n$ )
2   Draw random number  $p \in \text{Uni}[0, 1]$ 
3   if  $|\sigma| < n \wedge p \notin [0, 0.33]$  :
4     if  $p \in (0.33, 0.66]$  :
5        $i := 1$ 
6       forall  $b! \in \text{Act}_O$  do:
7         if  $\sigma b! \in \text{traces}^{\text{fin}}(\mathcal{S})$  :
8            $\text{result}[i] := \{b!\sigma' \mid \sigma' \in \text{batchGen}(\mathcal{S}, \sigma b!, n)\}$ 
9         else:
10           $\text{result}[i] := \{b!\}$ 
11         $i := i + 1$ 
12      return  $\text{result}[1], \dots, \text{result}[|\text{Act}_O|]$ 
13    if  $p \in (0.66, 1]$  :
14      Choose  $a? \in \text{Act}_I$  uniformly, such that  $\sigma \cdot a? \in \text{traces}^{\text{fin}}(\mathcal{S})$ 
15       $\text{result} := \{a?\sigma' \mid \sigma' \in \text{batchGen}(\mathcal{S}, \sigma a?, n)\}$ 
16      return  $\text{result}$ 
17    else:
18      return  $\{\varepsilon\}$ 

```

---

specification, it is added to the current branch, and the procedure `batchGen` is called again (line 8). If not, it is simply added to the branch (line 10). In the latter case, the branch of the tree stops and is to be labelled *fail* in a subsequent annotation step not present in the current procedure. Note, that `batchGen` is not called again in this instance, as erroneous output was already observed, and further testing is not needed. Lines 13-16 refer to stimulation of the system, by providing a random input. An input action  $a?$  present in the specification  $\mathcal{S}$  is chosen, added to the branch, and `batchGen` is called again.

A history is concatenated with the result of the procedure `batchGen`, thus a pIOTS in tree shape is continuously constructed. Eventually, the algorithm recursively returns the empty string in line 18, thus ensuring termination. Note that no implementation is needed for the generation of a batch test. We point out, that the procedure is a very basic form of random testing. More sophisticated algorithms are possible by finding different ways to assign concrete values for the stopping, observation and stimulation probability intervals. For instance, [69] claims that assigning a relation of 33% observation and 67% stimulation yielded better test results in practice, i.e. never terminate a branch before maximal test length is reached, stimulate in 67% of the cases, and observe with 33%. While there is no formal reasoning as to why this particular approach is of advantage over other choices, it provides a good starting point for further research.

**On-The-Fly Algorithm** Algorithm 4.2 presents how to evaluate a trace on-the-fly according to Definition 4.20. It requires a specification  $\mathcal{S}$ , an implementation  $\mathcal{I}$  and a maximal length  $n$  as inputs. The algorithm is a basic form of random testing, as opposed to its non-deterministic counterpart of Chapter 3.

Initially, the algorithm starts with the empty history and concatenates an

**Algorithm 4.2:** On-the-fly test derivation for pIOTSs.

---

**Input:** Specification pIOTS  $\mathcal{S}$ , implementation  $\mathcal{I}$  and upper bound for test length  $n \in \mathbb{N}$ .  
**Output:** Verdict *pass* if Impl. was ioco in the first  $n$  steps and *fail* if not.

```

1  $\sigma := \epsilon$ 
2 while  $|\sigma| < n$  do:
3   Draw random number  $p \in \text{Uni}[0, 1]$ 
4   if  $p \in [0, 0.5]$  :
5     observe next output  $b!$  (possibly  $\delta$ ) of  $\mathcal{I}$ 
6      $\sigma := \sigma b!$ 
7     if  $\sigma \notin \text{traces}^{fn}(\mathcal{S})$ : return fail
8   else:
9     Choose  $a? \in \text{Act}_I$  with  $\sigma a? \in \text{traces}^{fn}(\mathcal{S})$  uniformly and stimulate  $\mathcal{I}$  with  $a?$ 
10     $\sigma := \sigma a?$ 
11 return pass

```

---

action label after each step. It terminates after  $n$  steps were executed, or observed (line 2), where  $|\sigma|$  denotes the length of the currently observed trace. Observing the system for outputs is reflected in lines 4-7. In case output (incl. quiescence) is observed, the algorithm checks whether this is allowed by specification. If so, it proceeds with the next iteration, or returns the *fail* verdict otherwise. Lines 8-10 describe the stimulation process. The algorithm applies an input specified in the requirements model.

The algorithm returns whether a *fail* label was encountered in the first  $n$  steps. If erroneous output was detected, the resulting verdict is *fail*, and a *pass* verdict is given otherwise. Note that the probability of observing, or stimulating are arbitrarily chosen with equal probability, i.e. 50%.

**Theorem 4.29** (Algorithmic correctness). (i) *All test cases generated by Algorithm 4.1 are test cases according to Definition 4.18.*

(ii) *Every functionally correct implementation gets assigned the pass verdict by Algorithm 4.2.*

*Proof sketch.* The algorithms are one instantiation of non-deterministic choices as encountered in Algorithms 3.1 and 3.2 via uniform probability choices. Since test cases are *essentially* defined as IOTSs (Definition 4.18), and test annotations are defined equivalently as for Chapter 3 (Definition 4.20) the proofs are similar to the ones for Proposition 3.23 originally presented in [169], and are not further discussed here. For a more formal proof, we refer to the proof section, i.e. Section 4.6.  $\square$

### 4.3.2 Goodness of Fit

The previously presented algorithms provide a practical method to not only automatically generate test cases and annotate them, but to also infer about the functional verdict that an implementation under test should receive. However, we still lack a method to infer about the probabilistic correctness, i.e. a method to check whether discrete probability choices were implemented correctly. We aim to cover the lack thereof in this section.



This is especially relevant, since we do not have access to the underlying implementation model. Thus, having defined formal conformance with the **pioco** relation does not suffice. Instead, a practically applicable method similar to test annotations is needed. In order to overcome this, we consult a fundamental theorem from the literature [40], which we slightly adjusted to fit the canon of this chapter.

**Proposition 4.30** (Cheung, Stoelinga & Vaandrager). *Let  $\mathcal{A}$  and  $\mathcal{B}$  be two pIOTSs,  $\alpha \in (0, 1)$  and  $k \in \mathbb{N}$ . Then*

$$\text{trd}(\mathcal{A}, k) \subseteq \text{trd}(\mathcal{B}, k) \iff \forall m \in \mathbb{N} : \text{Obs}(\mathcal{A}, \alpha, k, m) \subseteq \text{Obs}(\mathcal{B}, \alpha, k, m).$$

The result says, that we have an embedding of finite trace distributions iff we have an embedding of the observation sets. Hence, showing that all observations of the implementation under test are also possible in the specification model implies an embedding on the trace distribution level. As a result, the statistical verdict of Definition 4.25 yields a pass.

However, much like we cannot possibly infer all traces and investigate their respective annotations for a functional verdict, it is impossible to gather *all* possible observations. This is due to the fact, that there is no guarantee of eventually observing every possible sample, since there are potentially uncountably many trace distributions to account for. In a practical scenario, it is much more likely to have *one* sample  $O$  originating from the implementation under test. To cope with the lack of information of the entire observation set, we resort to statistical hypothesis testing<sup>4</sup>.

A sensible null-hypothesis is given by  $O \in \text{OutObs}(\mathcal{S}, \alpha, k, m)$ . The level of significance  $\alpha$  gives us the parameter for the hypothesis test.

We point out two important facts:

- Accepting the null-hypothesis does not imply its correctness; It just means we have not (yet) found enough evidence to prove it wrong. Even though we might commit an error of second kind, we assume that the embedding of observations holds, and alongside Proposition 4.30 also the embedding of trace distributions, resulting in the statistical *pass* verdict.
- Rejecting the null-hypothesis, even though it is true, has a probability of  $\alpha$ . Thus, rejecting it, alongside the implementation under test yields a certain probability to commit an error of first kind. However, the chance of this happening is limited by  $\alpha$ , and is thus in line with the observations of the frameworks' correctness, cf. Definition 4.26

**Remark 4.31.** *The possibility to commit an error of first or second kind implies that the given verdict is not absolute in its correctness. While both quantities are natural in statistical hypothesis testing, an error of second kind is omnipresent in the model-based testing approach.*

*Since testing is inherently incomplete, flaws might still be present, even though the implementation underwent a thorough test process. Thus, what may appear*

<sup>4</sup>We refer the reader to Appendix A.2 for a short introduction to the topic.

as a weakness of the proposed framework, is naturally inherited from the overall MBT approach. Errors of first kind are simply a bi-product of working in a probabilistic environment, and are controllable via the parameter  $\alpha$ .

We remain to investigate the null-hypothesis, i.e. attempting to find a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{S}, k)$ , such that  $O \in \text{OutObs}(\mathcal{S}, \alpha, k, m)$ . Finding said trace distribution implies the acceptance of the null-hypothesis, while failing to find one means its rejection.

To find a fitting trace distribution we resort to Pearson's  $\chi^2$ -hypothesis testing. The empirical  $\chi^2$  score of a sample  $O$  is calculated as

$$\chi^2 = \sum_{i=1}^m \frac{(n(\sigma_i) - m \cdot \mathbb{E}^{\mathcal{D}}(\sigma_i))^2}{m \cdot \mathbb{E}^{\mathcal{D}}(\sigma_i)}, \quad (4.2)$$

where  $n(\sigma)$  is the number with which  $\sigma$  occurred in the sample. The score can be understood as the normalised cumulative sum of deviations from an expected value. Note that this entails a more general analysis of a sample than individual confidence intervals for each trace. The empirical  $\chi^2$  value is compared to critical values of given degrees of freedom and levels of significance. These values can be calculated, or universally looked up in a  $\chi^2$  table (Appendix A.2). If the empiric  $\chi^2$  is below the given threshold, the hypothesis is accepted, and consequently rejected if not.

Since the expected value  $\mathbb{E}^{\mathcal{D}}$  depends on a trace distribution, it is of interest to find a fitting one, i.e. find  $\mathcal{D}$  such that  $\chi^2 \leq \chi_{crit}^2$ . This turns (4.2) into an optimisation or constraint solving problem, i.e.

$$\min_{\mathcal{D} \in \text{trd}(\mathcal{S}, k)} \sum_{i=1}^m \frac{(n(\sigma_i) - m \cdot \mathbb{E}^{\mathcal{D}}(\sigma_i))^2}{m \cdot \mathbb{E}^{\mathcal{D}}(\sigma_i)}. \quad (4.3)$$

The probability of a trace is given by a scheduler and the corresponding path probability function (Definition 4.6). Hence, by construction, we want to optimize the probabilities  $p$  used by a scheduler to resolve non-determinism. This turns (4.3) into a minimisation, or constraint solving, of a rational function  $f(p)/g(p)$  with inequality constraints on the vector  $p$ . As shown in [137], minimizing rational functions of this general form is **NP**-hard.

Optimization naturally finds the best fitting trace distribution. Hence, it gives an indication on the goodness of fit, i.e. how close to a critical value the empirical  $\chi^2$  value is. Alternatively, instead of finding the *best* fitting trace distribution one could turn (4.2) into a satisfaction or constraint solving problem in values of  $p$ . This answers if values of  $p$  exist such that the empirical  $\chi^2$  value lies below the critical threshold.

**Example 4.32.** Recall Example 4.22 and assume we want to find out, if the sample presented on the right in Figure 4.10 is an observation of the specification of the shuffle music player, cf. Figure 4.5a. We already established

$$\text{freq}(O)(\sigma_i) = \begin{cases} \frac{15}{100} & \text{if } i = 1 \\ \frac{24}{100} & \text{if } i = 2 \\ \frac{26}{100} & \text{if } i = 3 \\ \frac{35}{100} & \text{if } i = 4 \end{cases} \quad \text{and} \quad n(\sigma_i) = \begin{cases} 15 & \text{if } i = 1 \\ 24 & \text{if } i = 2 \\ 26 & \text{if } i = 3 \\ 35 & \text{if } i = 4. \end{cases}$$

If we fix a level of significance at  $\alpha = 0.1$ , the critical  $\chi^2$  value is given as  $\chi_{crit}^2 = 6.25$  for three degrees of freedom (Appendix A.2). We have three degrees of freedom, since the frequency of the fourth trace is implicitly given, if we know the rest.

Let  $\mathcal{A}$  be a scheduler, that schedules *shuf?* with probability  $p$  and the distribution consisting of *song1!* and *song2!* with probability  $q$  in Figure 4.5a. We ignore the other choices the scheduler has to make for the sake of this example. We are trying to find values for  $p$  and  $q$  such that the empiric  $\chi^2$  value is smaller than  $\chi_{crit}^2$ , i.e. find  $p, q \in [0, 1]$  such that

$$\frac{(15-100 \cdot p \cdot q \cdot 0.25)^2}{100 \cdot p \cdot q \cdot 0.25} + \frac{(24-100 \cdot p \cdot q \cdot 0.25)^2}{100 \cdot p \cdot q \cdot 0.25} + \frac{(26-100 \cdot p \cdot q \cdot 0.25)^2}{100 \cdot p \cdot q \cdot 0.25} + \frac{(35-100 \cdot p \cdot q \cdot 0.25)^2}{100 \cdot p \cdot q \cdot 0.25} < 6.25.$$

Using MATLABs [86] function `fsolve()` for parameters  $p$  and  $q$  we quickly find the best empiric value as  $\chi^2 = 8.08 > 6.25$ . Hence, the minimal values for  $p$  and  $q$  provide a  $\chi^2$  minimum, which is still greater than the critical value. Therefore, there is no scheduler of the specification *PIOTS* that makes  $O$  a likely sample. Consequently, we reject the null-hypothesis alongside the implementation.

Contrary, assume Figure 4.5b were the requirements specification, i.e. we require *song1!* to be chosen with only 40% and *song2!* with 60%. The satisfaction problem for the same scheduler then becomes: find  $p, q \in [0, 1]$

$$\frac{(15-100 \cdot p \cdot q \cdot 0.16)^2}{100 \cdot p \cdot q \cdot 0.16} + \frac{(24-100 \cdot p \cdot q \cdot 0.24)^2}{100 \cdot p \cdot q \cdot 0.24} + \frac{(26-100 \cdot p \cdot q \cdot 0.24)^2}{100 \cdot p \cdot q \cdot 0.24} + \frac{(35-100 \cdot p \cdot q \cdot 0.36)^2}{100 \cdot p \cdot q \cdot 0.36} < 6.25,$$

because of different specified probabilities. In this case MATLABs [86] `fsolve()` gives the best empiric  $\chi^2$  value as  $\chi^2 = 0.257 < 6.25 = \chi_{crit}^2$  for  $p = 1$  and  $q = 1$ . Hence, we found a scheduler that makes the sample  $O$  most likely and accept the hypothesis, alongside the implementation.

### 4.3.3 Probabilistic Test Algorithm Outline

We summarize all necessary steps to perform model-based testing for probabilistic systems using our framework:

1. Generate a test case (suite resp.) for the specification *PIOTS*.
2. Execute a test case (all test cases of the test suite resp.)  $m$  times. If the functional *fail* verdict is encountered in any of the  $m$  executions, then fail the implementation for *functional* reasons.
3. Perform statistical analysis on the sample of size  $m$  for the test case (all test cases of the test suite resp.) by using optimisation or constraint solving. If a scheduler is found, such that  $\chi^2 \leq \chi_{crit}^2$  accept the implementation. If not, reject the implementation for *probabilistic* reasons.

4. If no *fail* was encountered during the process, accept the implementation.

Note that item 1 only applies in case the batch-generation algorithm in Algorithm 4.1 is used. Upon using the on-the-fly test algorithm, it is preferable to use the same seed for the random number generator. In this way, we guarantee to create the same test in every execution, even though different outcomes for probabilistic choices may occur.

## 4.4 Experiments

We present experimental results of our framework applied to three classical case studies known from the literature:

1. the Knuth and Yao Dice program [114],
2. the binary exponential backoff protocol [106], and
3. the FireWire root contention protocol [165].

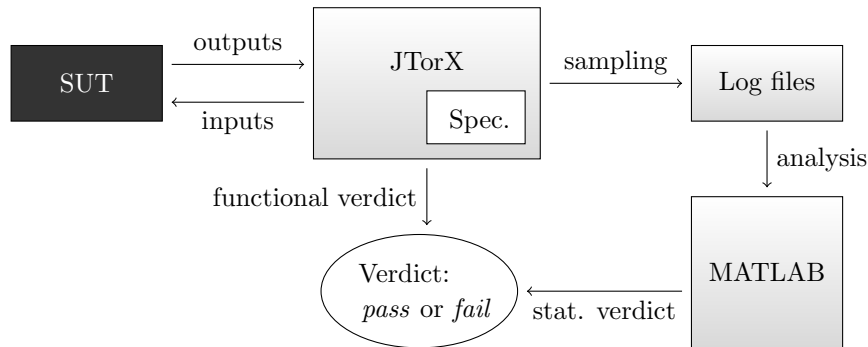


Figure 4.11: Experimental set up entailing the system under test, the MBT tool JTorX [15] and MATLAB [86]. Logs are gathered during the conformance test, and analysed later for a verdict on the implemented probabilities.

Our experimental set up can be seen in Figure 4.11. We implemented the three case studies in Java 7, and connected them to the MBT tool JTorX [15]. JTorX was provided with a specification for each of the three case studies. Note that JTorX is not capable of handling *probabilistic* specifications, and we had to resort to IOTSs. However, it implements **io** theory, which corresponds to the functional verdict. It generated test cases on the fly for each of the applications and the results were saved in log files. For each application we ran JTorX from the command line to initialize the random test generation algorithm. In total we saved  $10^5$  log files for every application.

The statistical analysis used MATLAB [86]. The function `fsolve()` was used for optimisation purposes in the parameters  $p$ , which represent the choices that the scheduler made. The statistical verdicts were calculated based on a

level of significance  $\alpha = 0.1$ . Note that MATLAB gave the *best* fitting scheduler for each application, instead of just *any* scheduler that keeps the  $\chi^2$  score below the threshold. We manually created mutants that implemented probabilistic deviations from the original protocols to analyse if they could be identified. All mutants were correctly given the statistical *fail* verdict, and all supposedly correct implementations yielded in the statistical *pass* verdict.

#### 4.4.1 Dice programs by Knuth and Yao

The dice programs by Knuth and Yao [114] aim at simulating a 6-sided die with multiple fair coin tosses. Formally, the discrete uniform distribution on the numbers 1 to 6 is simulated by repeatedly evaluating the discrete uniform distribution of the numbers 1 and 2. It is a classical example, and simple benchmark in probabilistic model checking [118]. An example specification model for a die using a fair coin, thus resulting in a fair 6 sided die roll, is given in Figure 4.12. Note that there is no non-determinism in this model.

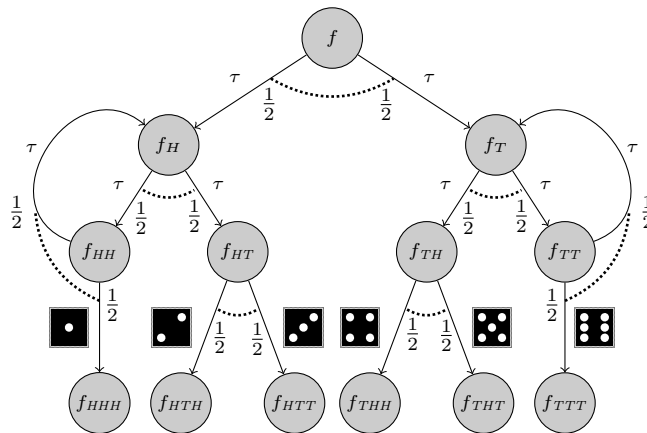


Figure 4.12: Dice program KY1 based on Knuth and Yao [114]. Rolling a 6-sided die is simulated by repeated tosses of a fair coin.

**Set Up.** To incorporate a non-deterministic choice, we implemented a program that chooses between a fair die, and an unfair (weighted) one. The unfair die uses an unfair coin to evaluate the outcome of the die roll. The probability to observe *head* with the unfair coin was set to 0.9. A model of the choice dice program can be seen in Figure 4.13. The action *roll?* represents the non-deterministic choice of which die to roll.

We executed two experiments with this implementation: The first utilizes Figure 4.12 as specification, while the second uses Figure 4.13 as specification. However, both use the same implementation described above. We expect that using a fair die as specification rejects the implementation, while the non-deterministic one does not.

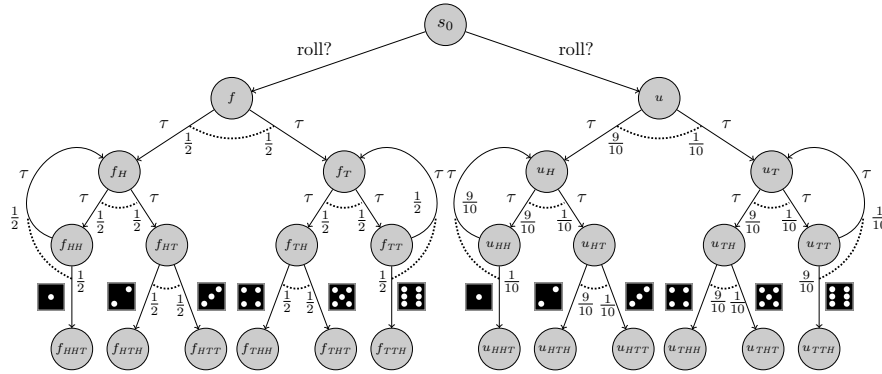


Figure 4.13: Dice program KY2 based on Knuth and Yao [114] version 2. The starting state enables a non-deterministic choice between a fair and an unfair die. The unfair die uses an unfair coin to determine its outcomes. The coin has a probability of 0.9 to yield *head*.

							Total
Observed value	29473	29928	10692	12352	8702	8853	100000
Relative frequency	0.294	0.299	0.106	0.123	0.087	0.088	1
Exp. probability KY1	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	1
Exp. probability KY2	$\frac{p}{6} + \frac{(1-p) \cdot 81}{190}$	$\frac{p}{6} + \frac{(1-p) \cdot 81}{190}$	$\frac{p}{6} + \frac{(1-p) \cdot 9}{190}$	$\frac{p}{6} + \frac{(1-p) \cdot 81}{990}$	$\frac{p}{6} + \frac{(1-p) \cdot 9}{990}$	$\frac{p}{6} + \frac{(1-p) \cdot 9}{990}$	1

Table 4.2: Observation of Knuth’s and Yao’s non-deterministic die implementation, and their respective expected probabilities according to specification KY1 (Figure 4.12) or KY2 (Figure 4.13). The parameter  $p$  depends on the scheduler that resolves the non-deterministic choice on which die to roll in KY2.

**Results.** We chose a level of significance  $\alpha = 0.1$  and gathered a sample of  $10^5$  traces of length 2, i.e. *roll?* followed by the outcome of the die role. We stored the logs for further statistical evaluation. The test process never ended due to erroneous functional behaviour. Consequently we conclude that the implementation is functionally correct.

Table 4.2 presents the statistical results of our simulation and the expected probabilities if

1. Model KY1 of Figure 4.12 is used as specification, and
2. Model KY2 of Figure 4.13 is used as specification.

Note that we left out the *roll?* action in the presented table for readability. Since there is no non-determinism in KY1, we expect each value to have a probability of  $\frac{1}{6}$ . In contrast, there is a non-deterministic choice to be resolved in KY2. Hence, the expected value is given depending on the parameter  $p$ , i.e. the probability with which the fair, or unfair die are chosen.

In order to assess if the implementation is correct with respect to a level

of significance  $\alpha = 0.1$ , we compare the  $\chi^2$  value for the given sample to the critical one given by  $\chi_{0.1,5}^2 = 9.24$ . The empirical  $\chi^2$  value is calculated as the normalised cumulative sum of the squared errors for each trace (Equation (4.2)). The critical value can universally be calculated or looked up in a  $\chi^2$  distribution table (Appendix A.2). We use the critical value for 5 degrees of freedom, because the outcome of the sixth trace is determined by the respective other five.

**KY1 as specification.** The calculated score approximately yields  $\chi_{KY1}^2 = 31120 \gg 9.24 = \chi_{0.1,5}^2$ . The implementation is therefore rightfully rejected, because the observation did not match our expectations.

**KY2 as specification.** The best fitting parameter  $p$  with MATLABs `fsolve()` yields  $p = 0.4981$ , i.e. the implementation chose the fair die with a probability of 49.81%. Consequently, a quick calculation showed  $\chi_{KY2}^2 = 5.1443 < 9.24 = \chi_{0.1,5}^2$ . Therefore, the implementation is assumed to be correct, because we found a scheduler, that chooses the fair and unfair die such that the observation is likely with respect to  $\alpha = 0.1$ .

Our results confirm our expectations: The implementation is rejected, if we require a fair die only (Figure 4.12). However, it is accepted if we *require* a choice between the fair and the unfair die (Figure 4.13).

#### 4.4.2 The Binary Exponential Backoff Algorithm

The Binary Exponential Backoff algorithm is a standardized data transmission protocol in the IEEE 802.3. between  $N$  hosts, trying to send information via one bus [106]. If two hosts try to send at the same time, their messages collide and they pick a waiting time before trying to send their information again. After  $i$  collisions, the hosts randomly choose a new discrete waiting time of the set  $\{0, \dots, 2^i - 1\}$  until no further collisions take place. Note that information thus gets delivered with probability one since the probability of infinitely many collisions is zero.

**Set Up.** We implemented the protocol in Java 7 and gathered a sample of  $10^5$  traces of length 5 for two communicating hosts. Note that the protocol is only executed if a collision between the two hosts arises. Therefore, each trace we collect starts with the *collide!* action. This is due to the fact that the two hosts initially try to send at the same time, i.e. discrete time unit 0. If a host successfully delivers its message it acknowledges this with the *send!* output and resets its clock to 0 before trying to send again.

Our specification of this protocol does not contain non-determinism. Thus, calculations in this example were not subject to optimization, or constraint solving to find the best fitting scheduler/trace distribution.

**Results.** The gathered sample is displayed in Table 4.3. The values of  $n$  show how many times each trace occurred. For comparison, the value  $m \cdot \mathbb{E}(\sigma)$  gives the expected number according to the specification of the protocol. Here,  $m$  is

ID	Trace $\sigma$	$n$	$\approx m\mathbb{E}(\sigma)$	$[l_{0.1}, u_{0.1}]$	$\approx \frac{(n-m\mathbb{E}(\sigma))^2}{m\mathbb{E}(\sigma)}$
1	collide! send! collide! send! send!	18656	18750	[18592, 18907]	0.47
2	collide! send! collide! send! collide!	18608	18750	[18592, 18907]	1.08
3	collide! collide! send! collide! send!	16473	16408	[16258, 16557]	0.26
4	collide! collide! send! send! collide!	12665	12500	[12366, 12633]	2.18
5	collide! send! collide! collide! send!	11096	10938	[10811, 11064]	2.28
6	collide! collide! collide! send! send!	8231	8203	[8091, 8314]	0.10
7	collide! collide! send! send! send!	6108	6250	[6152, 6347]	3.23
8	collide! collide! collide! send! collide!	2813	2734	[2667, 2800]	2.28
9	collide! collide! send! collide! collide!	2291	2344	[2282, 2405]	1.20
10	collide! send! collide! collide! collide!	1538	1563	[1512, 1613]	0.40
11	collide! collide! collide! collide! send!	1421	1465	[1416, 1513]	1.32
12	collide! collide! collide! collide! collide!	100	98	[85, 110]	0.04
$\chi^2 =$					14.84
<b>Verdict:</b>					<i>Accept</i>

Table 4.3: A sample of the binary exponential backoff protocol for two communicating hosts. We collected a total of  $m = 10^5$  traces of length  $k = 5$ . Calculations yield  $\chi^2 = 14.84 < 17.28 = \chi_{crit}^2 = \chi_{0.1,11}^2$ , hence we accept the implementation.

the total sample size and  $\mathbb{E}(\sigma)$  the expected probability. The interval  $[l_{0.1}, r_{0.1}]$  was included only for illustration purposes and represents the 90% confidence interval under the assumption that the traces are normally distributed. It gives a rough estimate on how much values are allowed to deviate for the given level of confidence  $\alpha = 0.1$ , but has no meaning otherwise.

However, we are interested in the *multinomial deviation*, i.e. less deviation of one trace allows higher deviation for another trace, and vice versa. In order to assess the statistical correctness, we compare the critical value  $\chi_{crit}^2$  to the empirical  $\chi^2$  score. The first is given as  $\chi_{crit}^2 = \chi_{0.1}^2 = 17.28$  for  $\alpha = 0.1$  and 11 degrees of freedom. This value can universally be calculated or looked up in a  $\chi^2$  distribution table (Appendix A.2). The empirical value is given by the sum of the entries of the last column of Table 4.3, i.e. the cumulative squared error.

A quick calculation shows  $\chi^2 = 14.84 < 17.28 = \chi_{0.1,11}^2$ . Consequently, we have no statistical evidence that hints at wrongly implemented probabilities in the backoff protocol. In addition, the test process never ended due to a functional fail verdict. Therefore, we conclude that the implementation is correct.

### 4.4.3 The FireWire Root Contention Protocol

The IEEE 1394 FireWire Root Contention Protocol [165] elects a leader between two contesting nodes via coin flips: If *head* comes up, node  $i$  picks a waiting time  $fast_i \in [0.24\mu s, 0.26\mu s]$ , if *tail* comes up, it waits  $slow_i \in [0.57\mu s, 0.60\mu s]$ . After the waiting time has elapsed, the node checks whether a message has



ID	Trace $\sigma$	$\approx m\mathbb{E}^{\mathcal{D}}(\sigma)(p)$	Correct	$M_1$	$M_2$	$M_3$	$M_4$	
			$n_c$	$n_{M_1}$	$n_{M_2}$	$n_{M_3}$	$n_{M_4}$	
1	$c_1? slow_1c_2? slow_2retry!$	$6250 \cdot p$	3148	1113	3091	3055	3161	
2	$c_1? slow_1c_2? slow_2done!$	$18750 \cdot p$	9393	3361	9047	9242	9329	
3	$c_1? slow_1c_2? fast_2done!$	$25000 \cdot p$	12531	40507	18163	15129	12982	
4	$c_1? fast_1c_2? fast_2retry!$	$8333 \cdot p$	4254	1467	4037	4066	4179	
5	$c_1? fast_1c_2? fast_2done!$	$16667 \cdot p$	8227	3048	7858	8474	8444	
6	$c_1? fast_1c_2? slow_2done!$	$25000 \cdot p$	12438	504	7918	10128	11867	
7	$c_2? slow_2c_1? slow_1retry!$	$6250 \cdot (1-p)$	3073	1137	2961	3256	3135	
8	$c_2? slow_2c_1? slow_1done!$	$18750 \cdot (1-p)$	9231	3427	9069	9456	9368	
9	$c_2? slow_2c_1? fast_1done!$	$25000 \cdot (1-p)$	12657	447	8055	9685	11975	
10	$c_2? fast_2c_1? fast_1retry!$	$8333 \cdot (1-p)$	4211	1466	4008	4131	4199	
11	$c_2? fast_2c_1? fast_1done!$	$16667 \cdot (1-p)$	8335	2977	7969	8295	8312	
12	$c_2? fast_2c_1? slow_1done!$	$25000 \cdot (1-p)$	12502	40546	17824	15083	13049	
			$p_{opt} \approx$	0.499	0.498	0.502	0.500	0.499
			$\chi^2 \approx$	9.34	169300	8175	2185	99.22
<b>Verdict</b>			<i>Accept</i>	<i>Reject</i>	<i>Reject</i>	<i>Reject</i>	<i>Reject</i>	

Table 4.4: A sample of length  $k = 5$  and depth  $m = 10^5$  of the FireWire root contention protocol [165]. Calculations of  $\chi^2$  are done after optimization in the parameter  $p$ . It represents which node got to flip its coin first.

arrived: if so, the node declares itself leader. If not, the node sends out a message itself, asking the other node to be the leader. Thus, the four possible outcomes of the coin flips are:  $\{fast_1, fast_2\}$ ,  $\{slow_1, slow_2\}$ ,  $\{fast_1, slow_2\}$  and  $\{slow_1, fast_2\}$ . Since pIOTSs cannot handle time, the check for exact waiting times was excluded.

The protocol contains non-determinism [165] as it is not clear, which node flips its coin first. Further, if different times were picked, e.g.  $fast_1$  and  $slow_2$ , the protocol always terminates. However, if equal times were picked, it either elects a leader, or retries depending on the resolution of the non-determinism.

**Set Up.** We implemented the root contention protocol in Java 7 and created four probabilistic mutants of it. The correct implementation  $C$  utilizes fair coins to determine the waiting time before it sends a message. The mutants  $M_1$ ,  $M_2$ ,  $M_3$  and  $M_4$  were subject to probabilistic deviations giving advantage to the second node via:

**Mutant 1.**  $P(fast_1) = P(slow_2) = 0.1$ ,

**Mutant 2.**  $P(fast_1) = P(slow_2) = 0.4$ ,

**Mutant 3.**  $P(fast_1) = P(slow_2) = 0.45$  and

**Mutant 4.**  $P(fast_1) = P(slow_2) = 0.49$ .

Statistically, the mutants should declare node 1 the leader more frequently. This is due to the fact that node 2 sends a leadership request faster on average.

**Results.** Table 4.4 shows the  $10^5$  recorded traces of length 5, where  $c_1$ ? and  $c_2$ ? denote the coins of node 1 and node 2 respectively. The expected value  $\mathbb{E}^{\mathcal{D}}(\sigma)$  depends on resolving one non-deterministic choice by varying  $p$  (which coin was flipped first). Note that the second non-deterministic choice was not subject to optimization, but immediately clear by the collected trace frequencies. The second column shows how many observations of the respective trace we expect for given sample size  $m$ . The remaining columns show actual trace frequency observations of the various mutant implementations

The empirical  $\chi^2$  score was calculated depending on parameter  $p$  and compared to the critical value  $\chi_{crit}^2$ . The latter is given as  $\chi_{crit}^2 = \chi_{0.1}^2 = 17.28$  for  $\alpha = 0.1$  and 11 degrees of freedom. We used MATLABs `fsolve()` to find the optimal value for  $p$ , such that the empirical value  $\chi^2$  is minimal. The resulting verdicts can be found in the last row of Table 4.4. The only accepted implementation was  $C$ , because  $\chi_C^2 < 17.28$ , whereas  $\chi_{M_i}^2 \gg 17.28$  for  $i = 1, \dots, 4$ .

## 4.5 Summary and Discussion

We defined a sound and complete framework to test probabilistic systems. At the core of our work is the conformance relation in the **ioco** tradition aptly called **pioco**. We presented how to automatically derive, execute and evaluate probabilistic test cases from a requirements model. The evaluation process handles functional and probabilistic behaviour. While the first can be assessed by means of **ioco** theory, we utilize frequency statistics in the latter. Our soundness and completeness results show that the correct verdict can be assigned up to arbitrary precision by means of a level of significance for a sufficiently large sample. We illustrated the application of our framework by means of three case studies from the literature: Knuth's and Yao's dice application, the binary exponential backoff protocol and the FireWire leadership protocol. The test evaluation process found no functional misbehaviour, indicating that the implemented functions were correct. Additionally, all correct implementations were given the statistical *pass* verdict, while all mutants were discovered.

Recall that Table 2.1 laid out a scaffold for an arbitrary MBT framework. All ingredients were instantiated throughout this chapter. We point out, that the MBT tool JTorX is not capable of handling probabilities [15] on its own. We used a shell script to execute it from the command line, and utilized MATLAB for statistical evaluation of the collected log files. Both steps seem easily automatable. However, some steps were performed manually, e.g. the calculation of the parametrised expected probabilities under a given trace distribution. If one dismisses the set up difficulties encountered upon using the MBT in practice, the bottleneck of the approach lies, in fact, in the optimization or SAT solving part. A solution could be the assumption of smaller scheduler classes than history dependent ones.

Physical Ingredients:	Formal Ingredients:
<ul style="list-style-type: none"> <li>• Informal requirements</li> <li>• Black-box implementation</li> <li>• Observations: Definition 4.23, <math>Obs(\mathcal{I} \parallel \hat{t}, \alpha, k, m)</math></li> </ul>	<ul style="list-style-type: none"> <li>• Model: Definition 4.1, pIOTS</li> <li>• Conformance: Definition 4.14, <math>\sqsubseteq_{pioco}</math></li> <li>• Test verdicts: Definition 4.25</li> </ul>
Tooling:	Objectives:
<ul style="list-style-type: none"> <li>• MBT tool: JTorX [15]</li> <li>• Test adapter: Implementable</li> <li>• Test generation method: Algorithms 4.1 and 4.2, Random testing</li> </ul>	<ul style="list-style-type: none"> <li>• Soundness: Theorem 4.27</li> <li>• Completeness: Theorem 4.28</li> </ul>
Assumptions:	
<ul style="list-style-type: none"> <li>• Every physical implementation has an underlying pIOTS model</li> </ul>	

Table 4.5: The MBT ingredients instantiated by the **pioco** framework.

## 4.6 Proofs

We present the proofs of the theorems within this chapter, alongside an additional lemma. Reoccurring theorems are numbered according to their occurrence in the chapter, while additional content is numbered alphanumerically.

**Lemma A.** *Let  $\mathcal{I}$  be a IOTS, and  $\sigma_1, \dots, \sigma_n \in \text{traces}^{fn}(\mathcal{I})$  be pairwise distinct traces of length  $k$ . Further, let  $p_1, \dots, p_n \in [0, 1]$  with  $\sum_{i=1}^n p_i = 1$ . Then there is  $\mathcal{D} \in \text{trd}(\mathcal{I}, k)$ , such that  $P_{\mathcal{D}}(\sigma_i) = p_i$  for all  $i = 1, \dots, n$ .*

*Proof.* Let  $\mathcal{I} = \langle S, s_0, Act_I, Act_O, Act_H, \Delta \rangle$  be an IOTS. Then  $(s, \mu) \in \Delta$  implies  $\mu \equiv \text{Dirac}$ , because  $\mathcal{I}$  is an IOTS. Since  $\sigma_i \in \text{traces}^{fn}(\mathcal{I})$  for  $i = 1, \dots, n$ , there must be at least one  $\pi_i \in \text{tr}^{-1}(\sigma_i)$  for each  $i$ . Without loss of generality, choose one  $\pi_i$  for each  $\sigma_i$ . Note that  $\pi_i \neq \pi_j$  for  $i \neq j$ , since  $\sigma_i \neq \sigma_j$  for  $i \neq j$ .

We proceed with the construction of a scheduler  $\mathcal{A}$ . Let

$$\mathcal{A}(s_0)(\mu) = \sum_{\{i \in \mathbb{N} \mid s_0 \mu a s \sqsubseteq \pi_i\}} p_i.$$

Further we set

$$\mathcal{A}(\pi \mu' a' s')(\mu) = \frac{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \mu a s \sqsubseteq \pi_i\}} p_i}{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \sqsubseteq \pi_i\}} p_i},$$

Observe that  $\mathcal{A}$  is a scheduler according to Definition 4.6, since  $p_i \in [0, 1]$  for

$i = 1, \dots, n$ , and

$$\sum_{(s_0, \mu) \in \Delta} \mathcal{A}(s_0)(\mu) = \sum_{(s_0, \mu) \in \Delta} \sum_{\{i \in \mathbb{N} \mid s_0 \mu a s \sqsubseteq \pi_i\}} p_i = \sum_{i=1}^n p_i = 1,$$

by assumption, as well as

$$\begin{aligned} \sum_{(s', \mu) \in \Delta} \mathcal{A}(\pi \mu' a' s')(\mu) &= \sum_{(s', \mu) \in \Delta} \frac{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \mu a s \sqsubseteq \pi_i\}} p_i}{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \sqsubseteq \pi_i\}} p_i} \\ &= \frac{\sum_{(s', \mu) \in \Delta} \sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \mu a s \sqsubseteq \pi_i\}} p_i}{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \sqsubseteq \pi_i\}} p_i} \\ &= \frac{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \sqsubseteq \pi_i\}} p_i}{\sum_{\{i \in \mathbb{N} \mid \pi \mu' a' s' \sqsubseteq \pi_i\}} p_i} = 1. \end{aligned}$$

The path probability function induced by  $\mathcal{A}$ , yields for the cone of all paths  $\pi_i = s_0 \mu_1 a_1 s_1 \dots \mu_{i_k} a_{i_k} s_{i_k}$  that

$$\begin{aligned} P_{\mathcal{A}}[C_{\pi_i}] &= \sum_{\{j \in \mathbb{N} \mid s_0 \mu_1 a_1 s_1 \sqsubseteq \pi_j\}} p_j \cdot \frac{\sum_{\{j \in \mathbb{N} \mid s_0 \mu_1 a_1 s_1 \mu_2 a_2 s_2 \sqsubseteq \pi_j\}} p_j}{\sum_{\{j \in \mathbb{N} \mid s_0 \mu_1 a_1 s_1 \sqsubseteq \pi_j\}} p_j} \\ &\quad \dots \frac{\sum_{\{j \in \mathbb{N} \mid \pi_j \sqsubseteq \pi_i\}} p_j}{\sum_{\{j \in \mathbb{N} \mid s_0 \mu_1 a_1 s_1 \dots \mu_{i_{k-1}} a_{i_{k-1}} s_{i_{k-1}} \sqsubseteq \pi_j\}} p_j} \\ &= p_i, \end{aligned}$$

as a telescope product. Since all  $\pi_i$  are distinct, it is consequently easy to check that  $P_{\text{trd}(\mathcal{A})}(\sigma_i) = P_{\mathcal{A}}[C_{\pi_i}] = p_i$ .  $\square$

**Theorem 4.16.** *Let  $\mathcal{I}$  and  $\mathcal{S}$  be two IOTs and  $\mathcal{I}$  be input enabled, then*

$$\mathcal{I} \sqsubseteq_{ioco} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{pioco} \mathcal{S}.$$

*Proof.*  $\boxed{\Leftarrow}$  Let  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$  and  $\sigma \in \text{traces}^{fin}(\mathcal{S})$ . Our goal is to show that  $\text{out}_{\mathcal{I}}(\sigma) \subseteq \text{out}_{\mathcal{S}}(\sigma)$ . Assume that there is  $b! \in \text{out}_{\mathcal{I}}(\sigma)$ . Then, we need to show that  $b! \in \text{out}_{\mathcal{S}}(\sigma)$ . For this, assume  $|\sigma| = k \in \mathbb{N}$ , and let  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$  such that  $P_{\mathcal{D}^*}(\sigma) = 1$ , which is possible by Lemma A.

Observe that  $\text{outcont}_{\mathcal{I}}(\mathcal{D}^*) \neq \emptyset$ , because  $\sigma b! \in \text{traces}^{fin}(\mathcal{I})$ . Consequently there is at least one trace distribution  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$  such that  $P_{\mathcal{D}}(\sigma b!) > 0$ . By assumption  $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$ , and therefore  $\mathcal{D} \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$ . We found a trace distribution of  $\mathcal{S}$ , with  $P_{\mathcal{D}}(\sigma b!) > 0$ . This implies that  $\sigma b! \in \text{traces}^{fin}(\mathcal{S})$  by Definition 4.11. Lastly  $b! \in \text{out}_{\mathcal{S}}(\sigma)$  and therefore  $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$ .

$\boxed{\Rightarrow}$  Let  $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$ ,  $k \in \mathbb{N}$  and  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$ . Further, assume  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$ . We want to show  $\mathcal{D} \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$ . Since the transition relations of  $\mathcal{I}$  and  $\mathcal{S}$  are finite, let  $\sigma_1, \dots, \sigma_n \in \text{traces}^{fin}(\mathcal{I})$  be all traces of  $\mathcal{I}$  with  $|\sigma_i| = k$  for  $i = 1, \dots, n$ . Note that  $\sigma_i \neq \sigma_j$  for  $i \neq j$ . Consider the three cases:

1. If  $\sigma_i \notin \text{traces}^{\text{fin}}(\mathcal{S})$ , then  $P_{\mathcal{D}}(\sigma_i a) = 0$  for  $a \in \text{Act}_O$  and  $i = 1, \dots, n$ . This follows from the definition of path probability (Definition 4.8) – schedulers may only assign probability greater than 0 to available paths. Since  $\sigma_i$  is not in  $\text{traces}^{\text{fin}}(\mathcal{S})$ , so are none of its underlying paths, nor their continuations with  $a \in \text{Act}_O$ . As a consequence, output continuations of such trace distributions yield probability 0, too.
2. If  $\sigma_i \in \text{traces}^{\text{fin}}(\mathcal{S})$ , then  $P_{\mathcal{D}}(\sigma_i a) = 0$  for  $a \in \text{Act}_I$  for  $i = 1, \dots, n$ , by definition of output continuations.
3. If  $\sigma_i \in \text{traces}^{\text{fin}}(\mathcal{S})$  and  $P_{\mathcal{D}}(\sigma_i a) > 0$  for  $a \in \text{Act}_O$ , then we know by the assumption  $\mathcal{I} \sqsubseteq_{\text{ioco}} \mathcal{S}$  that  $\sigma'_i a \in \text{traces}^{\text{fin}}(\mathcal{S})$ .

Let  $P_{\mathcal{D}}(\sigma_i a_j) = p_{ij}$  for  $a_j \in \text{Act}_O$ ,  $i = 1, \dots, n$  and  $j = 1, \dots, m$ . By construction of trace distributions it is  $q + \sum_{i,j} p_{ij} = 1$ , where  $q$  is the probability to halt in the  $(k+1)$ -th step. Treating “ $\perp$ ” as output action allows us to apply Lemma A to construct a scheduler  $\mathcal{A} \in \text{Sched}(\mathcal{S}, k+1)$ , such that  $P_{\text{trd}(\mathcal{A})}(\sigma_i a_j) = P_{\mathcal{D}}(\sigma_i a_j)$ . Thus  $\mathcal{D} \in \text{trd}(\mathcal{S}, k+1)$ , and by construction also  $\mathcal{D} \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$ . This overall yields  $\mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}$ .  $\square$

**Theorem 4.17.** *Let  $\mathcal{A}$ ,  $\mathcal{B}$  and  $\mathcal{C}$  be pIOTSSs, and let  $\mathcal{A}$  and  $\mathcal{B}$  be input enabled, then*

- (i)  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$  if and only if  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ .
- (ii)  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{\text{pioco}} \mathcal{C}$  then  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{C}$ .

*Proof.*  $\boxed{(i)} \implies$  Assume  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$ . We need to show  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ , i.e. that  $\mathcal{D} \in \text{trd}(\mathcal{A}, n)$  implies  $\mathcal{D} \in \text{trd}(\mathcal{B}, n)$  for all  $n \in \mathbb{N}$ .

Let  $n \in \mathbb{N}$  and  $\mathcal{D}^* \in \text{trd}(\mathcal{A}, n)$ . We prove the statement for every trace distribution prefix smaller or equal to  $n$  via induction: Assume  $\mathcal{D} \in \text{trd}(\mathcal{A}, 0)$ . Obviously  $\mathcal{D} \sqsubseteq_0 \mathcal{D}^*$ , and consequently  $\mathcal{D} \in \text{trd}(\mathcal{B}, 0)$ .

Now assume that the above statement has been shown for  $m$  with  $m = k-1 \leq n$ . We proceed by showing it holds for  $m = k$ . Let  $\mathcal{D} \in \text{trd}(\mathcal{A}, k)$  with  $\mathcal{D} \sqsubseteq_k \mathcal{D}^*$ . Then take  $\mathcal{D}' \in \text{trd}(\mathcal{A}, k-1)$  with  $\mathcal{D}' \sqsubseteq_{k-1} \mathcal{D}$ . By induction assumption we know  $\mathcal{D}' \in \text{trd}(\mathcal{B}, k-1)$ . With the initial assumption, i.e.  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$ , we know in particular that

$$\text{outcont}_{\mathcal{A}}(\mathcal{D}') \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}').$$

Therefore, we choose  $\mathcal{D}'' \in \text{outcont}_{\mathcal{A}}(\mathcal{D}')$ , such that

$$\forall \sigma \in \text{Act}^{k-1} \text{Act}_O : P_{\mathcal{D}}(\sigma) = P_{\mathcal{D}''}(\sigma), \quad (4.4)$$

i.e.  $\mathcal{D}$  and  $\mathcal{D}''$  assign the same probability of traces of length  $k$  ending in output. Note that  $\mathcal{D}'' \in \text{outcont}_{\mathcal{B}}(\mathcal{D}')$  and thus  $\mathcal{D}'' \in \text{trd}(\mathcal{B}, k)$ .

We are left to show that there is a trace distribution, say  $\mathcal{D}'''$ , that assigns

$$\forall \sigma \in \text{Act}^{k-1} \text{Act}_I : P_{\mathcal{D}}(\sigma) = P_{\mathcal{D}'''}(\sigma),$$

in addition to (4.4), i.e.  $\mathcal{D}'''$  assigns the same probability to *all* traces of length  $k$ . However, the existence of  $\mathcal{D}'''$  is straightforward, since  $\mathcal{A}$  and  $\mathcal{B}$  are input-enabled. That is, all inputs are enabled in every state of both  $\mathcal{A}$  and  $\mathcal{B}$ . We conclude  $\text{trd}(\mathcal{A}, m) \subseteq \text{trd}(\mathcal{B}, m)$  for all  $m \leq n$ , and with it consequently  $\text{trd}(\mathcal{A}, n) \subseteq \text{trd}(\mathcal{B}, n)$ . Hence  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ .

$\boxed{\Leftarrow}$  Let  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ . We need to show  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$ , i.e. for all  $n \in \mathbb{N}$  and for all  $\mathcal{D}^* \in \text{trd}(\mathcal{B}, n)$ , we have  $\text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . Let  $n \in \mathbb{N}$  and  $\mathcal{D}^* \in \text{trd}(\mathcal{B}, n)$ . Choose  $\mathcal{D} \in \text{outcont}_{\mathcal{A}}(\mathcal{D}^*)$ , then we need to show  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ .

By definition of the set *outcont*, we know that  $\mathcal{D} \in \text{trd}(\mathcal{A}, n+1)$ . Together with the initial assumption, i.e.  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ , we conclude  $\mathcal{D} \in \text{trd}(\mathcal{B}, n+1)$ . We are left to show that  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . However, this is straightforward, as for all traces  $\sigma \in \text{Act}^n \text{Act}_I$  it holds that  $P_{\mathcal{D}}(\sigma) = 0$  by construction of *outcont*. Consequently  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ , and therefore  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$ .

$\boxed{(ii)}$  We need to show that  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{\text{pioco}} \mathcal{C}$  imply  $\mathcal{A} \sqsubseteq_{\text{pioco}} \mathcal{C}$ . By initial assumptions we know

1.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{B}, n) : \text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ , and
2.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{C}, n) : \text{outcont}_{\mathcal{B}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .

We need to show

3.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{C}, n) : \text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .

Let  $n \in \mathbb{N}$ , choose  $\mathcal{D}^* \in \text{trd}(\mathcal{C}, n)$ , and assume  $\mathcal{D} \in \text{outcont}_{\mathcal{A}}(\mathcal{D}^*)$ . Obviously,  $\mathcal{D} \in \text{trd}(\mathcal{A}, n+1)$ . Together with (i), and since  $\mathcal{A}$  and  $\mathcal{B}$  are input enabled by assumption, we know  $\mathcal{D} \in \text{trd}(\mathcal{B}, n+1)$ . Note that for all traces  $\sigma \in \text{Act}^n \text{Act}_I$  we have  $P_{\mathcal{D}}(\sigma) = 0$ . With item 2., we know  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . Together with the assumption  $\mathcal{B} \sqsubseteq_{\text{pioco}} \mathcal{C}$ , we conclude  $\mathcal{D} \in \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .  $\square$

**Theorem 4.27** *Each annotated test for a pIOTS  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{\text{pioco}}$ .*

*Proof.* Let  $\mathcal{I}$  be an input enabled pIOTS and  $\hat{t}$  be a test for  $\mathcal{S}$ . Further assume that  $\mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}$ . Then we want to show  $V(\mathcal{I}, \hat{t}) = \text{pass}$ , i.e. we want to show that a **pioco** correct implementation passes an annotated test case. By the definition of verdicts (Definition 4.25) we have  $V(\mathcal{I}, \hat{t}) = \text{pass}$  if and only if

$$v_{\text{func}}(\mathcal{I}, \hat{t}) = v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}.$$

We proceed by showing that  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ , and  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  in two separate steps:

1. In order for  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ , we need to show that

$$\text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{pass} \text{ for all } \sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t}),$$

according to the definition of verdicts (Definition 4.25). Therefore, let  $\sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t})$ . We need to show  $\text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{pass}$  by the definition

of annotations (Definition 4.20). Assume  $\sigma' \in \text{traces}^{\text{fin}}(\mathcal{S})$  and  $a! \in \text{Act}_O$  such that  $\sigma' a! \sqsubseteq \sigma$ .

We observe two things:

- Since  $\varepsilon \in \text{traces}^{\text{fin}}(\mathcal{S})$ , i.e. the empty trace is a trace and is in  $\text{traces}^{\text{fin}}(\mathcal{S})$ ,  $\sigma'$  always exists.
- If no such  $a! \in \text{Act}_O$  exists, then  $\sigma$  is a trace solely consisting of inputs. By definition of annotations (Definition 4.20) consequently  $\text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{pass}$ .

By construction of  $\sigma$  we have  $\sigma' a! \in \text{traces}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$  and therefore also  $\sigma' a! \in \text{traces}^{\text{fin}}(\mathcal{I})$ . We conclude,  $\sigma' \in \text{traces}^{\text{fin}}(\mathcal{I}) \cap \text{traces}^{\text{fin}}(\mathcal{S})$ . Our goal is to show  $\sigma' a! \in \text{traces}^{\text{fin}}(\mathcal{S})$ .

Let  $l = |\sigma'|$  be the length of  $\sigma'$ . Without loss of generality, we can now choose  $\mathcal{D} \in \text{trd}(\mathcal{S}, l)$ , such that  $P_{\mathcal{D}}(\sigma') > 0$ . Note that this, together with the previous observation, yields that  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \neq \emptyset$ . Again, without loss of generality, we choose  $\mathcal{D}' \in \text{outcont}_{\mathcal{I}}(\mathcal{D})$ , such that  $P_{\mathcal{D}'}(\sigma' a!) > 0$ .

Lastly, we assumed  $\mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}$ , hence  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D})$ . We conclude  $\mathcal{D}' \in \text{trd}(\mathcal{S}, l + 1)$ , and  $P_{\mathcal{D}'}(\sigma' a!) > 0$ . By definition of trace distributions (Definition 4.11), this implies that  $\sigma' a! \in \text{traces}^{\text{fin}}(\mathcal{S})$ . If additionally  $\sigma' a! \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t})$ , then  $\sigma = \sigma' a!$ . Consequently  $\text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{pass}$  by definition of annotations (Definition 4.20). This ultimately yields  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ .

2. In order for  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  we need to show that

$$\forall \mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k) \exists \mathcal{D}' \in \text{trd}(\mathcal{S}, k) : P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha,$$

according to the definition of verdicts (Definition 4.25). Therefore, let  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$ . By definition of output-observations (Remark 4.24), we have

$$\text{OutObs}(\mathcal{D}, \alpha, k, m) = \left\{ O \in (\text{Act}^{\leq k-1} \text{Act}_O)^m \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_{\alpha} \right\}.$$

There exists  $\mathcal{D}' \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$  with

$$P_{\mathcal{D}'}(\sigma) = \begin{cases} 0 & \text{if } \sigma \in \text{Act}^{k-1} \text{Act}_I \\ P_{\mathcal{D}}(\sigma) & \text{if } \sigma \in \text{Act}^{\leq k-1} \text{Act}_O. \end{cases} \quad (4.5)$$

To see why, consider the scheduler that assigns all probability to halting instead of inputs for traces of length  $k$ , while assigning the same probability to outputs as the scheduler inducing  $\mathcal{D}$ . By construction of the set  $\text{OutObs}$  (Remark 4.24), observe that

$$\begin{aligned} P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

since only traces ending in output are measured.

It is now sufficient to show that  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k)$ . However, as an intermediate step, we first show that  $\mathcal{D}' \in \text{trd}(\mathcal{I}, k)$ , as this will let us make use of the assumption  $\mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}$ .

Consider the mapping  $f$  from the finite paths of  $\mathcal{I} \parallel \hat{t}$  to the finite paths of  $\mathcal{I}$ , i.e.  $f : \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t}) \rightarrow \text{paths}^{\text{fin}}(\mathcal{I})$ , where for every fragment of the path we have

$$f(\dots(s, q) \mu a (s', q') \dots) = \dots s \nu a s' \dots$$

Since tests are internally deterministic IOTSSs, it holds that  $\mu((s, t), a) = \nu(s, a)$ , i.e. tests do not change discrete probability distributions according to the definition of parallel composition (Definition 4.5). It is easy to see that  $f$  is an injective mapping, i.e.  $f(\pi_1) = f(\pi_2) \Rightarrow \pi_1 = \pi_2$ .

By definition of trace distributions (Definition 4.11) there is a scheduler, say  $\mathcal{A}' \in \text{Sched}(\mathcal{I} \parallel \hat{t}, k)$ , such that  $\text{trd}(\mathcal{A}') = \mathcal{D}'$ . With the help of  $f$  we construct a scheduler  $\mathcal{A}'' \in \text{Sched}(\mathcal{I})$ , such that for all traces  $\sigma$  we have  $P_{\text{trd}(\mathcal{A}')}(\sigma) = P_{\text{trd}(\mathcal{A}'')}(\sigma)$ , i.e.  $\text{trd}(\mathcal{A}'') = \mathcal{D}'$

For every path  $\pi \in \text{paths}^{\text{fin}}(\mathcal{I})$  with  $f^{-1}(\pi) \in \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$ , we define  $\mathcal{A}''$  as

$$\mathcal{A}''(\pi)(\nu) \stackrel{\text{def}}{=} \mathcal{A}'(f^{-1}(\pi))(\mu),$$

where  $\mu$  and  $\nu$  are like above. Note that  $P_{\mathcal{A}''}(\pi) = 0$  if  $\pi \notin \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$ .

The construction of  $\mathcal{A}''$  is straightforward: Due to the construction of test cases,  $\mathcal{I} \parallel \hat{t}$  is internally deterministic. Recall that both  $\mathcal{I}$  and  $\hat{t}$  are defined over the same action alphabet, and  $\hat{t}$  does not contain internal actions. This implies in particular, that there is no interleaving in  $\mathcal{I} \parallel \hat{t}$ . Thus  $\mathcal{A}''$  can copy the behaviour of  $\mathcal{A}'$  step-by-step. We set  $\mathcal{D}'' = \text{trd}(\mathcal{A}'')$  and conclude  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, k)$ . By construction it is easy to check that  $P_{\mathcal{D}''}(\sigma) = P_{\mathcal{D}'}(\sigma)$  for all traces  $\sigma$ . Further, observe

$$\begin{aligned} P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}'', \alpha, k, m)) &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) \\ &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

We proceed to show that  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$ . The proof is by induction over trace distribution length of prefixes of  $\mathcal{D}''$  up to  $k$ . Trivially, if  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, 0)$ , then also  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, 0)$ . Assume this has been shown for length  $n \in \mathbb{N}$ . We proceed by showing that the statement holds for  $n + 1 \leq k$ .

Let  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, n + 1)$  and take  $\mathcal{D}''' \sqsubseteq_n \mathcal{D}''$ . By induction assumption  $\mathcal{D}''' \in \text{trd}(\mathcal{S}, n)$ . Together with the assumption  $\mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}$ , we have

$$\text{outcont}_{\mathcal{I}}(\mathcal{D}''') \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}''').$$



Since  $\mathcal{D}'' \in \text{outcont}_{\mathcal{I}}(\mathcal{D}''')$  (Equation (4.5)) we have  $\mathcal{D}'' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}''')$ , and consequently  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, n+1)$ . We have shown  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$  and conclude  $P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$ . Ultimately, this yields  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  by the definition of verdicts (Definition 4.25).

Both parts together give  $V(\mathcal{I}, \hat{t}) = \text{pass}$ . This means that an annotated test for  $\mathcal{S}$  is sound with respect to  $\sqsubseteq_{\text{pioco}}$  for every  $\alpha \in (0, 1)$ .  $\square$

**Theorem 4.28** *The set of all annotated test cases for a pIOTS  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{\text{pioco}}$  for sufficiently large sample size.*

*Proof.* In order to show the completeness of test suite  $\hat{T}$  consisting of all annotated tests for  $\mathcal{S}$ , assume that  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$  for an input enabled pIOTS  $\mathcal{I}$ . Our goal is to show  $V(\mathcal{I}, \hat{T}) = \text{fail}$ . By the definition of verdicts (Definition 4.25) this holds iff  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{fail}$  or  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$  for some  $\hat{t} \in \hat{T}$ .

Since  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$ , there is  $k \in \mathbb{N}$ , such that there is  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$ , for which

$$\text{outcont}_{\mathcal{I}}(\mathcal{D}^*) \not\sqsubseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}^*).$$

More specifically

$$\exists \mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*) \forall \mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*) \exists \sigma \in \mathfrak{C} : P_{\mathcal{D}}(\sigma) \neq P_{\mathcal{D}'}(\sigma), \quad (4.6)$$

where  $\mathfrak{C} \stackrel{\text{def}}{=} \text{traces}^{\text{fin}}(\mathcal{I}) \cap \text{Act}^k \text{Act}_O$ . Without loss of generality, we can assume  $k$  to be minimal. There are two cases to consider:

1.  $\exists \sigma \in \mathfrak{C} : \sigma \notin \text{traces}^{\text{fin}}(\mathcal{S})$ , or
2.  $\forall \sigma \in \mathfrak{C} : \sigma \in \text{traces}^{\text{fin}}(\mathcal{S})$ ,

We will relate the two cases to the functional and the statistical verdict (Definition 4.25), respectively. We prove that item 1. implies  $v_{\text{func}}(\mathcal{I}, \hat{T}) = \text{fail}$ , and item 2. implies  $v_{\text{prob}}(\mathcal{I}, \hat{T}) = \text{fail}$ . Therefore, let  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$  such that Equation (4.6) holds for all  $\mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$ .

1. In order for  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{fail}$ , we need to show

$$\exists \sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t}) : \text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{fail}$$

for some  $\hat{t} \in \hat{T}$ , according to the definition of verdicts (Definition 4.25). Assume there is  $\sigma \in \mathfrak{C}$ , such that  $\sigma \notin \text{traces}^{\text{fin}}(\mathcal{S})$ . Our goal is to show that there is  $\hat{t} \in \hat{T}$  for which  $\sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t})$ , and  $\text{ann}_{\text{pioco}}^{\mathcal{S}}(\sigma) = \text{fail}$ .

Without loss of generality we can assume  $P_{\mathcal{D}}(\sigma) > 0$ . To see why, assume  $P_{\mathcal{D}}(\sigma) = 0$ . Then we can find a trace distribution in  $\text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$  with an underlying scheduler  $\text{Sched}(\mathcal{S})$  that does not assign probability to the last action in  $\sigma$  to obtain probability 0. This violates the assumption that  $P_{\mathcal{D}}(\sigma) \neq P_{\mathcal{D}'}(\sigma)$ . We conclude  $\sigma = \sigma' a$ , for some  $\sigma' \in \text{Act}^k$  and  $a \in \text{Act}_O$ .

The prefix  $\sigma'$  is in  $traces^{fin}(\mathcal{S})$ , because it is of length  $k$ , and since  $\mathcal{D}^* \in trd(\mathcal{S}, k)$ . Since  $\mathcal{D}$  and all  $\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)$  are continuations of  $\mathcal{D}^*$ , we conclude  $P_{\mathcal{D}^*}(\sigma') = P_{\mathcal{D}}(\sigma') = P_{\mathcal{D}'}(\sigma')$ , i.e. all trace distributions of the respective sets assign every prefix of  $\sigma$  the same probability by merit of *outcont*. We conclude  $\sigma' \in traces^{fin}(\mathcal{S})$ , but  $\sigma' a \notin traces^{fin}(\mathcal{S})$ .

By initial assumption  $\hat{T}$  contains *all* annotated test cases for  $\mathcal{S}$ . Hence, let  $\hat{t} \in \hat{T}$  such that  $\sigma \in traces^{com}(\hat{t})$ . By the definition of annotations (Definition 4.20) we have  $ann_{piooco}^{\mathcal{S}}(\sigma) = fail$ . Since  $\sigma \in traces^{fin}(\mathcal{I})$  and  $\sigma \in traces^{com}(\hat{t})$ , we obviously also have  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ . Ultimately, this yields  $v_{func}(\mathcal{I}, \hat{t}) = fail$ .

2. In order for  $v_{prob}(\mathcal{I}, \hat{t}) = fail$ , we need to show

$$\exists \mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, l) \forall \mathcal{D}' \in trd(\mathcal{S}, l) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, l, m)) < 1 - \alpha,$$

for some  $\hat{t} \in \hat{T}$  and  $l \in \mathbb{N}$ , according to the definition of verdicts (Definition 4.25).

Together with Equation (4.6) and the definition of acceptable outcomes (Definition 4.23), we conclude

$$\forall \mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k + 1, m)) < \beta_m \quad (4.7)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ . Observe that

$$\begin{aligned} & \sup_{\mathcal{D}' \in trd(\mathcal{S}, k+1)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k + 1, m)) \\ &= \sup_{\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k + 1, m)), \end{aligned} \quad (4.8)$$

by definition of *OutObs* (Remark 4.24). *OutObs* only comprises traces ending in output, thus its measure under any trace distribution of  $trd(\mathcal{S}, k + 1)$  cannot be larger than the ones already contained in  $outcont_{\mathcal{S}}(\mathcal{D}^*)$ . Together with Equation (4.7) this yields

$$\forall \mathcal{D}' \in trd(\mathcal{S}, k + 1) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k + 1, m)) < \beta_m \quad (4.9)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ .

We are left to show that  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, k + 1)$  for some  $\hat{t} \in \hat{T}$ . Let  $\mathfrak{R} = \{\sigma \in traces^{fin}(\mathcal{I}) \mid P_{\mathcal{D}}(\sigma) > 0\}$ , i.e. all traces getting assigned positive probability under  $\mathcal{D}$ . Obviously  $\mathfrak{C} \subseteq \mathfrak{R}$ . By initial assumption we know that all  $\sigma \in \mathfrak{C}$  are contained in  $traces^{fin}(\mathcal{S})$ . Hence implicitly all  $\sigma \in \mathfrak{R}$  are necessarily in  $traces^{fin}(\mathcal{S})$ . That means, there is a test case  $\hat{t}$  for  $\mathcal{S}$ , such that all  $\sigma \in \mathfrak{R}$  are in  $traces^{com}(\hat{t})$ . In particular, observe that all  $\sigma$  end in output by assumption. Hence, the last stage of every test case is the second bullet in the definition of test cases (Definition 4.18). We now construct a scheduler  $\mathcal{A}' \in Sched(\mathcal{I} \parallel \hat{t}, k + 1)$  such that  $trd(\mathcal{A}') = \mathcal{D}$ .

Consider the mapping  $f : tr^{-1}(\mathfrak{R}) \rightarrow paths^{fin}(\mathcal{I} \parallel \hat{t})$ , where for every path fragment

$$f(\dots s \mu a s' \dots) = \dots (s, q) \nu a (s', q') \dots$$

The state  $q$  is uniquely determined, because tests are internally deterministic. In particular  $q = q'$  iff  $a = \tau$ . Since every discrete distribution in test cases is the Dirac distribution it is  $\mu(s, a) = \nu((s, q), a)$ . It is then easy to see that  $f$  is an injection, i.e.  $f(\pi_1) = f(\pi_2) \Rightarrow \pi_1 = \pi_2$ .

We now construct a scheduler  $\mathcal{A}' \in \text{Sched}(\mathcal{I} \parallel \hat{t})$ , such that  $\mathcal{D} = \text{trd}(\mathcal{A}')$ . Let  $\mathcal{A} \in \text{trd}(\mathcal{I})$  be the underlying scheduler that induces  $\mathcal{D}$  by definition of trace distributions (Definition 4.11). For every  $\pi \in \text{tr}^{-1}(\hat{\mathfrak{K}})$  we define

$$\mathcal{A}'(\pi)(\nu) \stackrel{\text{def}}{=} \mathcal{A}(f^{-1}(\pi))(\mu),$$

where  $\mu$  and  $\nu$  are as above. Observe  $P_{\text{trd}(\mathcal{A}')}(\sigma) = 0$  for  $\sigma \notin \hat{\mathfrak{K}}$ . The construction of  $\mathcal{A}'$  is straightforward: Since  $\hat{t}$  is internally deterministic, and does not contain internal actions, there is no interleaving in  $\mathcal{I} \parallel \hat{t}$ . Hence, a scheduler of  $\mathcal{I} \parallel \hat{t}$  may copy the decisions of  $\mathcal{A}$  step-by-step. Note in particular that every of  $\hat{t}$ 's discrete distributions is the Dirac distribution, and hence  $P_{\text{trd}(\mathcal{A}')}(\sigma) = P_{\mathcal{D}}(\sigma)$  for all  $\sigma \in \hat{\mathfrak{K}}$ . We conclude  $\text{trd}(\mathcal{A}') = \mathcal{D}$  and therefore  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k + 1)$ .

Together with Equation (4.8), we have found a scheduler  $\mathcal{A}'$  such that  $\text{trd}(\mathcal{A}') \in \text{trd}(\mathcal{I} \parallel \hat{t}, k + 1)$ , and for all  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k + 1)$  we have

$$P_{\mathcal{D}'}(\text{OutObs}(\text{trd}(\mathcal{A}'), \alpha, k + 1, m)) < \beta_m. \quad (4.10)$$

Now iff  $\alpha \leq 1 - \beta_m$ , we estimate (4.10) further to

$$P_{\mathcal{D}'}(\text{OutObs}(\text{trd}(\mathcal{A}'), \alpha, k + 1, m)) < \beta_m \leq 1 - \alpha.$$

However, the inequality  $\alpha \leq 1 - \beta_m$  always holds for sufficiently large  $m$ , since  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$  by the definition of acceptable outcomes (Definition 4.23). Ultimately, this yields  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$ .

Together, the two cases yield  $\mathcal{I} \not\sqsubseteq_{\text{pioco}} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = \text{fail}$ .  $\square$

#### Theorem 4.29

- (i) All test cases generated by Algorithm 4.1 are test cases according to Definition 4.18.
- (ii) Every functionally correct implementation gets assigned the pass verdict by Algorithm 4.2.

*Proof.*  $\boxed{(i)}$  Let  $\mathcal{S} = \langle \mathcal{S}, s_0, \text{Act}_I, \text{Act}_O, \text{Act}_H, \Delta \rangle$  be a pIOTS. Furthermore, let  $t = \text{batchGen}(\mathcal{S}, \varepsilon, n)$  for some  $n \in \mathbb{N}$ . In order to show that  $t$  is a test case according to Definition 4.18, we need to show that  $t$  is 1. a pIOTS, 2. internally deterministic, and 3. a finite and connected tree. We prove each property separately in the following. Let the states of  $t$  be given by the history  $\sigma$ .

1. The interface  $(Act_I, Act_O)$  of  $t$  is finite and directly transferred from the specification  $\mathcal{S}$  by the algorithm, i.e. line 13. The procedure initially starts with the empty history  $\varepsilon$ , thus the starting state  $s_0 = s_\varepsilon$  is uniquely determined. Line 3 ensures termination: every call of the procedure adds at least one action to history  $\sigma$  (line 8, 10 and 15), thus  $|\sigma|$  is increasing. When  $|\sigma| = n$ , i.e. the maximally specified test length is reached, the procedure returns  $\varepsilon$  (line 3 and 18). Since this holds for every branch, the recursive calls of the procedure inevitably stop in the states  $s_{\sigma.\varepsilon}$ , and the algorithm terminates. Additionally, infinite cycles are never introduced, since line 3 contains the only loop. As a consequence, we conclude that the number of states of  $t$  is finite. Additionally, a state  $s_\sigma$  traverses to state  $s_{\sigma.a}$ , with  $a \in Act_I \cup Act_O \cup \{\varepsilon\}$ , with probability 1. Even though the choice parameter  $p$  is determined probabilistically (line 2), no such choice is given for the added action  $a$  and all distributions are Dirac. Since  $Act_I$  and  $Act_O$  are finite, the transition relation of each state is finite. This means that  $t$  is a pIOTS according to Definition 4.1.
2. The procedure recursively assembles a pIOTS. Note that it is only called again, once an externally visible action from the sets  $Act_I$  and  $Act_O$  is chosen (lines 8 and 15). The procedure stops if  $p \in [0, 0.33]$  or the maximal test length  $n$  is reached (line 3). This means, the only way non-determinism is possibly introduced, is if the same action is added twice to a branch in *one* execution of the procedure. However, the algorithm cycles through the entire output alphabet exactly once, while line 13 chooses one input. Hence, a state  $s_{\sigma.a}$  with  $a \in Act_O \cup Act_I \cup \{\varepsilon\}$  is uniquely determined, and there is no possibility of introducing internal non-determinism. Consequently,  $t$  is internally deterministic.
3. Each state  $s_{\sigma.a}$  with  $a \in Act_I \cup Act_O \cup \{\varepsilon\}$  has a uniquely determined parent node  $s_\sigma$ . Recursively calling `batchGen` connects source  $s_\sigma$  with target  $s_{\sigma.a}$ , i.e. transitions are of the form  $s_\sigma \xrightarrow{Dirac,a} s_{\sigma.a}$ . Line 3 ensures termination if  $|\sigma| = n$ . Hence,  $t$  is finite, connected, and every state  $s_\sigma$  is reachable.

We showed that  $t$  is a finite, internally deterministic and connected tree pIOTS. Consequently,  $t$  is a test case according to Definition 4.18.

(ii) In order to show that every functionally correct implementation gets assigned the *pass* verdict by the `on-the-fly` test derivation algorithm, let  $\sigma$  be a generated trace by the `on-the-fly` algorithm. We need to show:

$$\mathcal{I} \sqsubseteq_{ioco} \mathcal{S} \implies \mathcal{I} \text{ passes } \sigma.$$

We prove the statement by contraposition.

Assume  $\mathcal{I}$  *fails*  $\sigma$ , i.e.  $ann_{pio}^{\mathcal{S}}(\sigma) = fail$ . Note that  $\sigma$  keeps track of the trace exhibited by the implementation thus far in every iteration of the `while` loop. The only way for the algorithm to return *fail*, is when  $\sigma \notin traces^{fm}(\mathcal{S})$  in line 7.

In this case, we can always write  $\sigma = \sigma' b!$  with  $b! \in Act_O$  (line 6). It is easy to check, that up to the point of returning *fail*, only inputs were provided, or correct outputs were observed, i.e. output present in  $\mathcal{S}$ . Since only input present in  $\mathcal{S}$  is provided (line 9), and the algorithm terminates as soon as unexpected output is observed (line 7), it follows that  $\sigma' \in traces^{fin}(\mathcal{S})$ .

If the *fail* verdict is returned, we evidently encountered  $b! \in out_{\mathcal{I}}(\sigma')$ , but  $b! \notin out_{\mathcal{S}}(\sigma')$  because  $\sigma' b! \notin traces(\mathcal{S})$  (line 7). Consequently, if an implementation fails a test case generated by the **on-the-fly** test generation algorithm, it is functionally incorrect, i.e.  $\mathcal{I} \not\equiv_{ioco} \mathcal{S}$ .  $\square$



# CHAPTER 5

---

## Model-Based Testing with Markov Automata

---

The role of computer-based systems is ever increasing: robots, drones and autonomous cars will soon pervade our lives. Attuning to this progress, verification and validation techniques of these systems have grown to a field of crucial importance. They provide methods that show whether the actual and the intended behaviour of a system differ, or give confidence that they do not. As a consequence, the progressively intricate design of embedded systems continuously brings new challenges to the field of verification engineers. The key question of whether a system works as intended therefore has a variety of angles: Was the functional behaviour correctly implemented? Does the system continue to operate under a work overload? Is the average lifetime within safety regulations? Can requirements be met on time?

As seen in Chapter 4 probabilistic aspects in many computer applications naturally add one of those angles. This chapter builds on the established test methodology for probabilistic automata, and ensures that *soft real-time constraints* are added to the capabilities of our test framework. The requirements specification now additionally prescribes stochastic time delay between actions. There is a large body of different MBT frameworks in the literature that accommodate a variety of requirements aspects, like functional properties [173], or real-time constraints [21, 29, 120]. Surprisingly, only few papers are concerned with the testing of probabilistic systems. As a result, an even smaller fraction of the literature deals with the addition of stochastically delayed real-time.

Albeit strikingly similar in the use of continuous time, we point out that *real-time testing* and *stochastically timed testing* are fundamentally different. The first specifies an *exact* time interval in which actions are required to take place. The latter prescribes time *distributions*, e.g. deviations are allowed, as long as overall usage follows a certain distribution shape. To illustrate, assume a file is requested from a server. Workload of the server and multiple requests, as well as the file size may cause a significant delay until the file is actually delivered. Naturally, the circumstances of each file request may differ, and with it will the time until it arrives. Rather than prescribing a fixed time in which its arrival is expected, we ask for an appropriate delay *on average*. Thus, we are

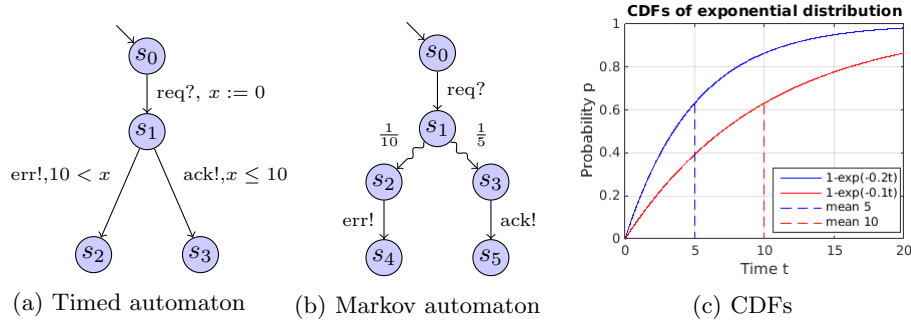


Figure 5.1: Example models illustrating the intent of MBT based on timed automata [6] versus Markov automata [67]. Timed automata specify hard time constraints, while Markov automata require stochastic delay. The delay is distributed according to the CDFs of an exponential distribution with parameter  $\frac{1}{10}$  and  $\frac{1}{5}$ , respectively. Note that the models specify entirely different behaviour.

willing to accept a server that supplies some files after the specified average, as long as it also supplies some amount before the average.

While the goal of this thesis is to establish a framework which handles non-determinism, discrete probability choices, and general stochastic time distributions, this chapter is an intermediate step in that we first consider a specific kind of distribution; the exponential distribution. It is of particular interest because of its memorylessness, i.e. the waiting time until a certain event happens, does not depend on the already elapsed time. Due to this property, and due to its simplicity, it is used to model real life phenomena whose true distribution of time is not known, but the average is, e.g.

- the time between receiving two phone calls,
- the degradation rate of construction components and machines,
- radioactive decay, or
- the arrival of a new item in a queue.

This makes the exponential distribution an interesting *intermezzo* on the voyage to general continuous distributions. Hence, in this chapter we present an applicable framework in an MBT setting, that allows to specify three key properties in tandem: 1. non-deterministic choices, 2. discrete probability choices, and 3. exponentially distributed time delay. The foundation of our methodology are Markov automata (MAs), since they incorporate all three properties. Non-determinism is utilized to model choices, that are not under the systems direct control. Conversely, discrete probability choices represent choices made by the system (e.g. coin tosses or random seeds) or the environment (e.g. failure probabilities). Lastly, the exponential distributions model stochastic delay between two actions. These give an appropriate approximation, if only the mean duration of an activity is known, as is often the case in a practical setting.



Mathematically, MAs arise as the conservative extension of both probabilistic automata (PAs) [158] and interactive Markov chains (IMCs) [93]. We refer to Figure 5.2 for an hierarchical overview. As such, it is a natural consequence that the major part of our testing techniques of Chapter 4 carries over to the Markov automata framework. In this chapter, we recall the formal definition of Markov automata, and their corresponding language theoretic concepts. We define trace distribution semantics that resolve all non-deterministic choices. This allows us to talk about *the probability of a trace*. Furthermore, test cases and their annotations are defined according to a conformance relation akin to **io**. Naturally, the framework is subsequently proven as correct.

The modelling capabilities of Markov automata compared to probabilistic automata come at a price. We require novel techniques to cope with real-time both in theory, and in practical application of the framework. While counting trace frequencies sufficed in the PA setting, it generally does not work when considering time. To illustrate, is receiving a file after five seconds the same *behaviour* as receiving the file after ten seconds? And what about 5.1 seconds?

We summarize the key contributions of this chapter:

1. The input output Markov automata model, comprising non-deterministic choices, discrete probability distributions, and exponentially delayed transitions,
2. a behavioural description for Markov automata based on trace distribution semantics,
3. definitions of test cases, test execution and verdicts,
4. the soundness and completeness results of our framework,
5. the treatment of quiescence in a setting containing stochastically delayed time, and
6. a small case study in the Bluetooth device discovery protocol.

**Related Work.** There is a large body of work on testing real-time systems [21, 116, 120]. Brandán-Briones et al. [29] extend existing frameworks to incorporate the imperative notion of quiescence, i.e. the absence of outputs.

Conversely, probabilistic testing pre-orders and equivalences are well-studied [45, 60, 158]. Distinguished work by [122] introduces the concept of probabilistic bisimulation via hypothesis testing. Largely influential work is given by [40], presenting how to observe trace frequencies during a sampling process. Executable

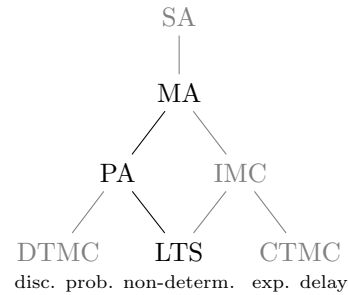


Figure 5.2: Traversing the automata formalism hierarchy shown in Figure 1.3.

probabilistic test frameworks are suggested for probabilistic finite state machines in [96, 103] and Petri nets [23].

Closely related to our work is the study of Markovian bisimulation. The foundation of an observational equivalence is presented in [67] in the form of weak bisimulation for Markov automata, and was refined by introducing late-weak bisimulation [59, 162] and branching bisimulation [171]. The relating bisimulation relations for Markov automata are studied in Chapter 6 in more detail.

**Origins of the chapter.** The work underlying this chapter was performed in collaboration with Mariëlle Stoelinga, and appeared in

- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of stochastic systems with ioco theory. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, A-TEST*, pages 45–51, 2016.
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems with stochastic time. In *Proceedings of the 11th International Conference on Tests and Proofs, TAP*, pages 77–97, 2017.

**Organisation of the chapter.** Section 5.1 introduces the underlying automaton model in Markov automata, and recalls language theoretic concepts. Section 5.2 describes the theory of using Markov automata in the test process, and shows that it is correct. Conversely, a practical approach utilizing Markovian test theory, and an algorithmic outline are shown in Section 5.3. We bring the framework to application by investigating the Bluetooth device discovery protocol in Section 5.4. Lastly, Section 5.5 concludes the chapter with a summary and discussion.

## 5.1 Input Output Markov Automata

We introduce the underlying model for the remainder of the next two chapters in (input/output) Markov automata. We reiterate the step from a closed system like LTSs or PAs, to an open input output system, like IOTSs or pIOTSs. We do so by first recalling closed Markov automata from the literature, before taking the step to input/output Markov automata. Working in the latter formalism is of priority when considering a testing scenario where interaction is desired.

Following the basic structure of Chapters 3 and 4 we illustrate the modelling formalism alongside easy examples, and introduce relevant language theoretic concepts in paths and traces, and their time abstract counterparts *abstract paths* and *abstract traces*. Further, we define parallel composition to allow communication of a system’s subcomponents, as well as its interaction with test cases. Lastly, we extend the trace distribution semantics, previously seen in Chapter 4 to Markov automata.

### 5.1.1 Definition

The foundation of the methodology in this chapter are Markov automata (MAs). As a conservative extension of both probabilistic automata (PAs) and interactive Markov chains (IMCs), cf. Figure 5.2, they are equipped with both probabilistic and non-deterministic choices. The first represent choices made by the system or the environment. The latter model choices that are not under its control. Complementary, Markov automata are of particular interest because of their memoryless exponential distributions modelling time delay. These give a highly appropriate approximation, if only the mean duration of an activity is known, as is often the case in practice.

In the context of this thesis, MAs arise by equipping the PAs of Chapter 4 with the *Markovian transition relation*. Markovian transitions are defined via one positive real valued parameter, representing the mean duration of a delay to take place.

**Definition 5.1.** A Markov automaton (MA) is a tuple  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$ , where

- $S$  is a set of states, with  $s_0 \in S$  as the initial state.
- $Act_\tau$  is a set of actions, containing the distinguished element  $\tau$ .
- $\rightarrow \subseteq S \times Act_\tau \times Distr(S)$  is the countable probabilistic transition relation.
- $\rightsquigarrow \subseteq S \times \mathbb{R}^+ \times S$  is the countable Markovian transition relation.

Definition 5.1 only differs in the fourth bullet point from probabilistic automata. That is, the distinctive feature of Markov automata are their exponentially distributed timed transition comprised in the set  $\rightsquigarrow$ . Given a transition  $(s, \lambda, s') \in \rightsquigarrow$  means there is an exponentially distributed delay of going from  $s$  to  $s'$ . The probability to go from  $s$  to  $s'$  within  $T$  time units is thus given as  $1 - e^{-\lambda T}$ . The sum of all outgoing Markovian transitions of a state  $s$  is called its *exit rate*. In order for the model to be meaningful, we require the exit rate of all states to be finite.

We point out two interesting observations applying for Markov automata:

1. Multiple outgoing Markovian transitions give rise to the so called *race condition*. There is a race between multiple exponential distributions inducing a discrete probability distribution over time, e.g. given Markovian transitions  $(s, \lambda, s')$  and  $(s, 2\lambda, s'')$ , the latter is twice as likely to be taken in the same time interval based on its parameter.
2. Markov automata contain the internal action label  $\tau$ , marking internal progress. We apply the *maximal progress* assumption [93], meaning that time is not allowed to progress in states with an outgoing transition labelled  $\tau$ . To indicate the absence of  $\tau$ , states are called *stable* if they do not have an outgoing  $\tau$ -transition. With no passage of time in unstable states, the probability to take a Markovian transition is effectively 0. This renders Markovian transition in unstable states unnecessary [128].

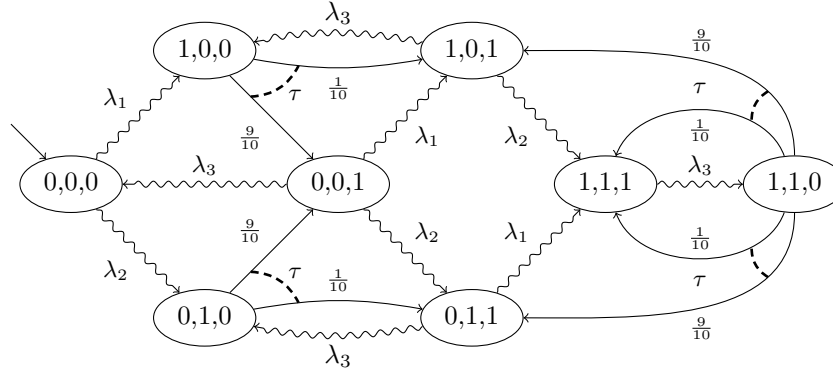


Figure 5.3: Example Markov automaton model used in [84]. The wavy arrows represent Markovian transitions alongside their positive real-valued parameters  $\lambda_i$ , with  $i = 1, 2, 3$ . The steady arrows together with dashed lines between them illustrate discrete probability distributions with corresponding values and action labels along the edges. We refer to Example 5.2 for more details.

**Example 5.2.** Figure 5.3 depicts the state space of a job queueing system comprising two stations and one server used in [84]. System states are represented by a triple  $(s_1, s_2, j)$ , where  $s_i$  are the number of jobs in station  $i$ , and  $j$  is the number of jobs in the server. The initial state is given by  $(0, 0, 0)$ , i.e. all components are empty. For simplicity and illustration purposes, we assume that each component can hold at most one job.

The incoming rate of jobs to the stations are parametrised with Markovian parameters  $\lambda_1$  and  $\lambda_2$ , respectively. These describe the average time duration used to model the delays between two states. The requests are stored by the server until they are fetched. Jobs are then processed with rate  $\lambda_3$ . On the modelling level, they are represented by wavy edges between two states, aiding the reader in the distinction of Markovian and probabilistic transitions.

Upon polling a station, there is a  $\frac{1}{10}$  probability of erroneously keeping a job in the station after being fetched. Like before, discrete probability distributions are represented by the straight edges between two states, alongside dashed lines between all edges belonging to the same distribution.

If both stations contain a job, the server chooses non-deterministically. While certain discrete probability distributions can possibly be represented by so called race conditions between outgoing Markovian actions, this is not possible in state  $(1, 1, 0)$ . The state has two outgoing, non-deterministic  $\tau$  distributions. As such, the given system can neither be modelled by an interactive Markov chain [93], nor by a probabilistic automaton [158] simply based on Markovian delays.

Similar to the steps from LTSs to IOTSs, or from PAs to pIOTS seen before in Chapters 3 and 4, we separate the label alphabet of Markov automata in distinct input and output sets. This captures possible communication of a system with its environment, e.g. a tester, or other components, and gives rise to input

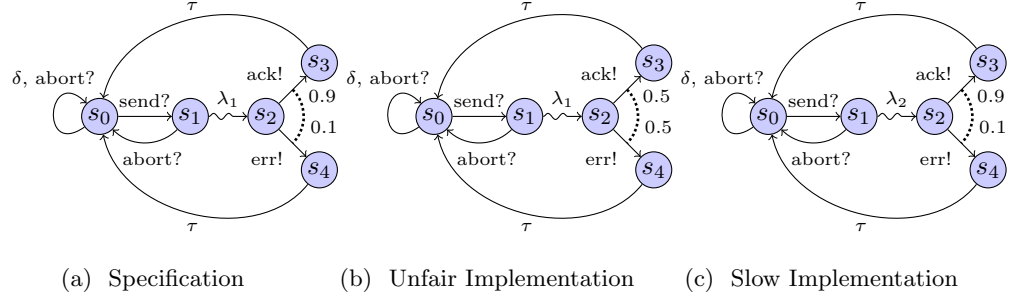


Figure 5.4: Protocol specification IOMA and two erroneous implementations. After the input  $send?$  there is an exponentially delayed transition, followed by an acknowledgement or error output.

output Markov automata (IOMAs). The existence of a distinct quiescence label  $\delta$  is required again in the set of output labels, explicitly characterising the absence of all other outputs. Like before, the set of internal/hidden actions describe internal progress of a system, that is not visible to an external observer.

We define IOMAs according to [185] to be *input-reactive* and *output-generative* to capture a wider range of models. To recall: upon receiving an input action, the automaton decides probabilistically which next state to move to. However, upon producing an output, the automaton decides both the output and the state probabilistically. Formally, this means that each transition either involves one input action, or possibly several outputs, quiescence or internal actions.

**Definition 5.3.** An input output Markov automaton (IOMA) is a seven tuple  $\mathcal{M} = \langle S, s_0, Act_I, Act_O, Act_H, \rightarrow, \rightsquigarrow \rangle$ , where

- $S$  is a set of states with the unique starting state  $s_0$ .
- $Act_I, Act_O$  and  $Act_H$  are disjoint sets of input, output and internal/hidden labels respectively, containing the distinct quiescence label  $\delta \in Act_O$ . We write  $Act = Act_I \sqcup Act_O \sqcup Act_H$  for the set of all labels.
- $\rightarrow \subseteq S \times Distr(Act \times S)$  is the finite probabilistic transition relation, such that for all input actions  $a \in Act_I$  and distributions  $\mu \in Distr(Act \times S)$ :  $\mu(a, s') > 0$  implies  $\mu(b, s'') = 0$  for all  $b \neq a$  and states  $s', s'' \in S$ .
- $\rightsquigarrow \subseteq S \times \mathbb{R}^+ \times S$  is the Markovian transition relation.

**Example 5.4.** Figure 5.4 shows three input-reactive output-generative IOMA. The model describes a protocol that associates a delay with every send action, followed by an acknowledgement or error. Input is suffixed with “?” and output with “!”. Discrete probability distributions are denoted with a dotted arc, together with the action label and corresponding probabilities. Markovian actions are presented as wavy arrows.

After the `send?` input is received, there is an expected delay indicated by the Markovian action  $\lambda_1$ . The delay is exponentially distributed, thus, the probability to go from  $s_1$  to  $s_2$  in  $T$  time units is  $1 - e^{-\lambda_1 T}$ . In state  $s_2$  there is one outgoing discrete probability distribution. The specification in Figure 5.4a implies that only 10% of all messages should end in an error report and the remaining 90% get delivered correctly. After a message is delivered, the automaton goes back to its initial state where it stays quiescent until input is provided. This is denoted with the  $\delta$  self-loop, marking the desired absence of outputs.

The unfair implementation model given in Figure 5.4b essentially has the same structure, except for altered discrete probabilities in the distribution sending an acknowledgement or error, respectively. While the delay conforms to the one prescribed in the specification model, sufficiently many executions of the implementation should reveal the erroneously assigned discrete probabilities, i.e. an error is sent more frequently than is prescribed.

While the slow implementation presented in Figure 5.4c assigns the same probabilities to the output actions, it assigns the parameter  $\lambda_2$  to the exponential delay between input and output. This is regarded as conforming iff  $\lambda_1 = \lambda_2$ .

The test theory presented in this chapter aims at establishing a framework capable of differentiating the two erroneous implementations from the requirements specification model.

We point out that our model comprises exponentially delayed transitions, as well as the quiescence action  $\delta$ . The latter formally describes the absence of outputs for an *indefinite* period of time. However, this imposes an interesting challenge to a testing framework based on MA, as quiescence in practice is frequently judged by waiting a finite amount of time [29]. The interplay of quiescence and exponential delays is therefore further investigated in Section 5.3.

**Notation.** By convention, we use the following notations and concepts:

- Elements of the set of input actions are suffixed by “?”, and elements of the set of output actions are suffixed by “!”. By convention, we let  $\tau$  represent an element of the set of internal actions, and  $\delta$  be the distinct output label to denote *quiescence*. Throughout this chapter we let  $\mu$  and  $\nu$  be discrete probability distributions, and  $\lambda_i$  be positive real-valued numbers denoting parameters of Markovian transitions.
- $\{\} \dots \}$  denotes a *multi-set*.
- We write  $s \xrightarrow{\mu, a} s'$ , if  $(s, \mu) \in \rightarrow$  and  $\mu(a, s') > 0$  for some  $s' \in S$ . Further, we write  $s \rightarrow a$ , if there are  $\mu \in \text{Distr}(\text{Act} \times S)$  and  $s' \in S$  such that  $s \xrightarrow{\mu, a} s'$ , and  $s \not\rightarrow a$ , if not.
- We write  $s \overset{\lambda}{\rightsquigarrow} s'$ , if  $(s, \lambda, s') \in \rightsquigarrow$ . Then  $\lambda$  is called *Markovian action*,
- We write  $s \xrightarrow{\mu, a}_{\mathcal{M}} s'$ , etc. to clarify that a transition belongs to an IOMA  $\mathcal{M}$  if ambiguities arise.

- A state  $s \in S$  is called *probabilistic*, if there is at least one  $a \in Act$  such that  $s \rightarrow a$ . In that case we also say action  $a$  is *enabled* in  $s$ . The set  $enabled(s)$  comprises all enabled actions in  $s$ .
- A state is called *stable*, if it enables no internal action.
- A state  $s \in S$  is called *Markovian*, if there is at least one  $\lambda \in \mathbb{R}^+$  such that  $s \xrightarrow{\lambda} s'$ .
- We call an IOMA  $\mathcal{M}$  *input enabled*, if all input actions are enabled in all states, i.e. for all  $a \in Act_I$  we have  $s \rightarrow a$  for all  $s \in S$ .
- The *rate* to go from a state  $s$  to  $s'$  is the sum of all  $\lambda \in \mathbb{R}^+$ , such that  $(s, \lambda, s') \in \rightsquigarrow$  and is denoted  $\mathbf{R}(s, s')$ .
- The *exit rate* of a state  $s$  is the sum of all rates and is denoted  $\mathbf{E}(s)$ . We require  $\mathbf{E}(s) < \infty$  for all  $s \in S$ .
- The *discrete branching probability distribution* of a state  $s$  quantifying race conditions is given by  $\mathbb{P}_s(s') = \mathbf{R}(s, s') / \mathbf{E}(s)$ .

**Parallel Composition.** In order to allow for synchronisation and communication between multiple system components, we define parallel composition. Two systems synchronize on shared actions, and evolve independently on others. Like for pIOTS, the transitions of multiple components are stochastically independent, thus we need to multiply the probabilities of discrete distributions when taking shared actions. For two distributions  $\mu$  and  $\nu$  this is indicated by the operator  $\mu \otimes \nu$ . To avoid name clashes, we only compose compatible IOMAs.

**Definition 5.5.** *Two IOMAs*

$$\begin{aligned} \mathcal{M} &= \langle S, s_0, Act_I, Act_O, Act_H, \rightarrow, \rightsquigarrow \rangle, \text{ and} \\ \mathcal{M}' &= \langle S', s'_0, Act'_I, Act'_O, Act'_H, \rightarrow', \rightsquigarrow' \rangle, \end{aligned}$$

are compatible if  $Act_O \cap Act'_O = \{\delta\}$ , and  $Act_H \cap Act'_H = Act \cap Act'_H = \emptyset$ .

**Definition 5.6.** Let  $\mathcal{M} = \langle S, s_0, Act_I, Act_O, Act_H, \rightarrow, \rightsquigarrow \rangle$  and  $\mathcal{M}' = \langle S', s'_0, Act'_I, Act'_O, Act'_H, \rightarrow', \rightsquigarrow' \rangle$  be two compatible IOMAs. Their parallel composition is the tuple

$$\mathcal{M} \parallel \mathcal{M}' = \langle S'', (s_0, s'_0), Act''_I, Act''_O, Act''_H, \rightarrow'', \rightsquigarrow'' \rangle, \text{ where}$$

- $S'' = S \times S'$ ,
- $Act''_I = (Act_I \cup Act'_I) \setminus (Act_O \cup Act'_O)$ ,
- $Act''_O = Act_O \cup Act'_O$ ,
- $Act''_H = Act_H \cup Act'_H$ , and finally the transition relation
- $\rightarrow'' = \{((s, t), \mu) \in S'' \times Distr(L'' \times S'') \mid$ 

$$\mu \equiv \begin{cases} \nu_1 \otimes \nu_2, & \text{if } \exists a \in Act \cap Act' \text{ such that } s \xrightarrow{\nu_1, a} \wedge t \xrightarrow{\nu_2, a} \\ \nu_1 \otimes \mathbb{1}, & \text{if } \forall a \in Act \text{ with } s \xrightarrow{\nu_1, a} \text{ we have } t \not\rightarrow a \\ \mathbb{1} \otimes \nu_2, & \text{if } \forall a \in Act' \text{ with } t \xrightarrow{\nu_2, a} \text{ we have } s \not\rightarrow a \end{cases} \},$$

where  $(s, \nu_1) \in \Delta$ , and  $(t, \nu_2) \in \Delta'$ . Further we let  $\nu_1 \otimes \mathbb{1}((s', t'), a) = \nu_1(s', a) \cdot \nu_2(t', a)$ ,  $\nu_1 \otimes \mathbb{1}((s', t'), a) = \nu_1(s', a) \cdot 1$  and  $\mathbb{1} \otimes \nu_2((s', t'), a) = 1 \cdot \nu_2(t', a)$ .

$$\begin{aligned} \bullet \rightsquigarrow'' &= \{((s, t), \lambda, (s', t')) \in S'' \times \mathbb{R}^+ \times S'' \mid ((s, t), \lambda, (s', t')) \\ &= \begin{cases} (s, t), \lambda, (s, t') & \text{if } \mathbf{E}(s) = 0 \wedge (t, \lambda, t') \in \rightsquigarrow' \wedge t \neq t' \\ (s, t), \lambda, (s', t) & \text{if } (s, \lambda, s') \in \rightsquigarrow \wedge s \neq s' \wedge \mathbf{E}(t) = 0 \\ (s, t), \mathbf{R}(s, s) + \mathbf{R}(t, t), (s, t) & \text{if } \mathbf{R}(s, s) > 0 \wedge \mathbf{R}(t, t) > 0 \end{cases} \}. \end{aligned}$$

The synchronisation on non-Markovian actions coincides with the one for pIOTS in Definition 4.5. Markovian transitions evolve independently, with the exception of Markovian self-loops in both states. Formally, having two self-loops with the same parameter  $\lambda$  in both  $s$  and  $t$ , requires the use of  $\mathbf{R}(s, s) + \mathbf{R}(t, t)$  in the new Markovian transition relation, as  $\rightsquigarrow''$  is a relation. That is, there cannot be two elements of the form  $((s, t), \lambda, (s, t))$ , as  $\rightsquigarrow''$  is not a multi-set.

### 5.1.2 Abstract Paths and Abstract Traces

Let  $\mathcal{M} = \langle S, s_0, Act_I, Act_O, Act_H, \rightarrow, \rightsquigarrow \rangle$  be an IOMA. We define the usual language theoretic concepts. A *path*  $\pi$  of  $\mathcal{M}$  is a (possibly) infinite sequence of the form

$$\pi = s_0 t_1 \mu_1 \alpha_1 s_1 t_2 \mu_2 \alpha_2 \dots,$$

where  $s_i \in S$ ,  $t_i \in \mathbb{R}_0^+$ , and either  $(s_{i-1}, \mu_i) \in \rightarrow$  and  $\alpha_i \in Act$ , or  $\mu = \mathbb{P}_{s_{i-1}}$  and  $\alpha_i \in \mathbb{R}_0^+$  for  $i = 1, 2, \dots$ . We require that each finite path ends in a state, and either  $s_{i-1} \xrightarrow{\mu_i, \alpha_i} s_i$ , or  $s_{i-1} \rightsquigarrow^{\alpha_i} s_i$  for each non-final  $i$ . The sequence  $s_{i-1} t_i \mu_i \alpha_i s_i$  means that  $\mathcal{M}$  resided  $t_i$  time units in state  $s_{i-1}$  before moving to  $s_i$  via  $\alpha_i$  using the distribution  $\mu_i$ . We use  $last(\pi)$  to denote the last state of a finite path. We write  $\pi' \sqsubseteq \pi$  to denote  $\pi'$  as a *prefix* of  $\pi$ , i.e.  $\pi'$  is finite, ends in a state, and coincides with  $\pi$  on the first finitely many symbols of the sequence. The set of all finite paths of  $\mathcal{M}$  is set as  $paths^{fin}(\mathcal{M})$ , and all paths by  $paths(\mathcal{M})$ . The set of *complete paths*, denoted  $paths^{com}(\mathcal{M})$ , contains every path ending in a deadlock state, i.e. it neither allows probabilistic nor Markovian progress.

Note that a single time point has probability zero to occur in any given continuous time span. Hence, it is necessary to talk about time intervals instead of individual time values. This gives rise to *abstract paths*. An abstract path is a path, where each occurrence of single time values  $t_i$  is replaced by intervals  $I_i \subseteq \mathbb{R}_0^+$ . However, we limit our interested to intervals of the form  $[0, t]$  with  $t \in \mathbb{R}_0^+$ . Consequently, any path can be replaced with its abstract path by changing  $t_i$  to  $[0, t_i]$ , or vice versa. This convention aids in defining a bijection between paths and their abstract counterparts. Throughout this chapter, we use  $\pi$  to denote a path and  $\Pi$  to denote the corresponding abstract path, and vice versa. We summarise all abstract finite paths in the set  $AbsPaths^{fin}(\mathcal{M})$ , and all abstract paths in  $AbsPaths(\mathcal{M})$ . For two abstract paths  $\Pi$  and  $\Pi'$  with

$$\Pi = s_0 I_1 \mu_1 a_1 s_1 \dots s_n \text{ and } \Pi' = s_0 I'_1 \mu'_1 a'_1 s'_1 \dots,$$



we say  $\Pi$  is a prefix of  $\Pi'$ , denoted  $\Pi \sqsubseteq \Pi'$ , if  $a_i = a'_i$ ,  $s_i = s'_i$ ,  $\mu_i = \mu'_i$  and  $I_i = I'_i$  for  $i = 1, 2, \dots, n$ . That is,  $\Pi$  and  $\Pi'$  coincide on the first  $n$  steps.

The *trace* of a path  $tr(\pi)$  records its visible behaviour, i.e. time and input/output actions. It is a mapping  $tr : paths(\mathcal{M}) \rightarrow (\mathbb{R}_0^+ \times Act_I \cup Act_O)^\omega$ . We formally overload  $tr$  for finite paths, i.e.  $tr : paths^{fin}(\mathcal{M}) \rightarrow (\mathbb{R}_0^+ \times Act_I \cup Act_O)^*$ . Hence, a trace is the (possibly) infinite sequence of the form

$$\sigma = tr(\pi) = t_{i_1} a_{i_1} t_{i_2} a_{i_2} t_{i_3} a_{i_3} \dots,$$

where  $t_{i_j} \in \mathbb{R}_0^+$  and  $a_{i_j} \in Act_I \cup Act_O$  for  $j = 1, 2, \dots$ . Note that a path fragment  $s_1 t_1 \mu_1 \lambda s_2 t_2 \mu_2 a s_3$  collapses to  $(t_1 + t_2) a$  if  $\lambda$  is a Markovian action. This rule is applied recursively for multiple Markovian actions. Markovian actions are parameters for an exponential distribution and deemed invisible.

The set  $tr^{-1}(\sigma)$  is the set of all paths, which have trace  $\sigma$ . The *length* of a path  $\pi$ , denoted  $|\pi|$  is the number of actions on its trace. All traces of  $\mathcal{M}$  are summarized  $traces(\mathcal{M})$ , and all finite traces in  $traces^{fin}(\mathcal{M})$ . The set of *complete traces*, denoted  $traces^{com}(\mathcal{M})$ , contains every trace based on at least one complete path.

Similar to abstract paths, an *abstract trace* is given, if all  $t_i \in \mathbb{R}_0^+$  of a trace are replaced by intervals  $I_i \subseteq \mathbb{R}_0^+$ . Again, we limit our intent to abstract traces only using intervals of the form  $[0, t]$  with  $t \in \mathbb{R}_0^+$ . This enables us to use traces and abstract traces interchangeably. Hence, for a given trace  $\sigma$  we denote  $\Sigma$  as its corresponding abstract trace, and vice versa. We summarise all abstract traces in the set  $AbsTraces(\mathcal{M})$ , and all finite ones in  $AbsTraces^{fin}(\mathcal{M})$ . The prefix relation of abstract traces is defined similarly to prefixes of abstract paths.

Lastly, let  $act(\sigma)$  return visible actions of trace  $\sigma$  only.

**Example 5.7.** Consider the IOMA  $\mathcal{M}$  given in Figure 5.5 and assume the distributions of the three Dirac distributions are denoted  $\mu_a, \mu_b$  and  $\mu_c$ . For the

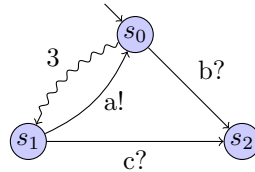


Figure 5.5: Yardstick example for paths and abstract paths.

*path*  $\pi = s_0 \ 2.9 \ \mathbb{P}_{s_0} \ 3 \ s_1 \ 0 \ \mu_a \ a! \ s_0 \ 0 \ \mu_b \ b? \ s_2$  we have

- $\Pi = s_0 \ [0, 2.9] \ \mathbb{P}_{s_0} \ 3 \ s_1 \ [0, 0] \ \mu_a \ a! \ s_0 \ [0, 0] \ \mu_b \ b? \ s_2$ .
- $tr(\pi) = 2.9 \ a! \ 0 \ b?$ ,
- $tr(\Pi) = [0, 2.9] \ a! \ [0, 0] \ b?$ ,
- $act(tr(\pi)) = a! \ b?$ ,
- $|\pi| = 2$ , and lastly  $\pi \in paths^{com}(\mathcal{M})$ .

### 5.1.3 Schedulers and Trace Distributions

With the modelling formalism and language theoretic concepts in place, we turn to schedulers and their trace distribution semantics of Markov automata. Like before, our goal is to quantify the probability of (abstract) traces. Since MA are a conservative extension of PA, it is natural that the need for schedulers to resolve non-determinism is directly carried over. By establishing schedulers and their resulting trace distributions, we eliminate all non-determinism, and receive a purely probabilistic system. However, the introduction of continuous time into the model prevents us from directly applying defined concepts of Chapter 4 to MA. Instead, we define schedulers and study the resulting path probability measure and probability space, before we tend to trace distributions.

Even though the mathematical framework for abstract paths and traces is technically more involved, it is completely standard [196].

**Definition 5.8.** A scheduler  $\mathcal{A}$  of an IOMA  $\mathcal{M} = \langle S, s_0, Act_I, Act_O, Act_H, \rightarrow, \rightsquigarrow \rangle$  is a function

$$\mathcal{A} : paths^{fin}(\mathcal{M}) \longrightarrow SubDistr(Distr(Act \times S) \cup \{\perp\}),$$

such that 1. for each finite path  $\pi$  only available distributions or halting are scheduled, 2.  $\mathcal{A}(\pi)$  is a full distribution if  $last(\pi)$  is not a Markovian state, and 3. internal actions cannot be postponed, i.e.

- $\forall \pi \in paths^{fin}(\mathcal{M}) : \mathcal{A}(\pi)(\mu) > 0$ , then  $(last(\pi), \mu) \in \rightarrow$ ,
- $|\mathcal{A}(\pi)| = 1$  if  $last(\pi)$  is not a Markovian state, and
- if  $Act_H \cap enabled(last(\pi)) \neq \emptyset$ , then  $|\mathcal{A}(\pi)| = 1$ .

The value  $\mathcal{A}(\pi)(\perp)$  is the probability to interrupt/halt the process. A scheduler  $\mathcal{A}$  halts on path  $\pi$ , if  $\mathcal{A}(\pi)(\perp) = 1$ . We say a scheduler is of length  $k \in \mathbb{N}$ , if it halts for all paths  $\pi$  with length greater or equal to  $k$ , and for every complete path smaller than  $k$ . We denote this set by  $Sched(\mathcal{M}, k)$  and the set of all finite schedulers by  $Sched(\mathcal{M})$  respectively.

Definition 5.8 is highly reminiscent to the one for pIOTS (Definition 4.6). We require the scheduler to be *randomised* and *history dependent*. Further, we defined IOMAs as input-reactive and output-generative. This necessitates to schedule *sub-distributions* as opposed to mere actions. Obviously, the definition makes sure that only available distributions are chosen. We require sub-distributions, as opposed to full distributions, such that the remaining probability mass a scheduler did not assign to actions in  $Act$  or halting  $\perp$  is left for Markovian actions. That is, a scheduler chooses an action or halts *immediately*, or leaves a chance for Markovian actions to take place. Again, we refer the reader to Chapter 8 for a study on the hierarchy of scheduler classes for stochastic automata, and point out that Markov automata form a subclass of them.

**Remark 5.9.** We use schedulers in the context of MBT in an open environment. By design the scheduler of one IOMA schedules both inputs and outputs, as

opposed to similar approaches in the literature. For instance, [39] use two schedulers for communicating systems, and an arbiter scheduler that tells precisely how progress of the composed system is determined. We point out, that our approach has the caveat of being non-compositional, see e.g. [164]. We utilize schedulers only to resolve non-deterministic choices to determine probabilities of paths and traces, and are not particularly concerned with their compositionality.

**Remark 5.10.** *With the presence of exponentially delayed Markovian transitions, there is a compelling argument to define schedulers with the capability to wait before making a decision. While this design choice potentially opens up interesting possibilities in a testing scenario, there are drawbacks that lead us to not incorporate this property into the framework:*

*The increasing complexity we experience upon incorporating continuous time is held at a minimum, when only exponential distributions are considered. A key property of exponential distributions is their memorylessness. It would seem natural, to give schedulers the same power: wait before scheduling with an exponentially distributed delay. We found that the benefits this yields do not outweigh the increased complexity that comes with it with respect to MBT: like in the pIOTS case, we are required to find the best fitting scheduler after sampling. Recall that this was **NP-hard** for pIOTS, cf. Section 4.3. Adding continuous time to the search space in the optimisation/constraint solving step would require us to find the best fitting scheduler to maximise the likelihood of delays.*

*Nonetheless, the implications of waiting schedulers in an MA setting are studied in Chapter 6. There, we rid ourselves of the overarching goal to use the framework in an MBT scenario. Rather we study it as a subject of its own by comparing its distinguishing power to other frameworks known in the literature like strong- or weak bisimulation.*

**Probability Spaces associated to Schedulers.** A scheduler naturally induces a probability space over a Markov automaton  $\mathcal{M}$ . In order to define it, we need the notion of *maximal paths*, i.e. infinite paths, or paths that halt with a non-zero probability.

**Definition 5.11.** *A path  $\pi$  of a scheduler  $\mathcal{A}$  is a finite or infinite path*

$$\pi = s_0 t_1 \mu_1 a_1 s_1 t_2 \mu_2 a_2 s_2 t_3 \mu_3 a_3 s_3 \dots,$$

*where  $\mathcal{A}(s_0 t_1 \mu_1 a_1 s_1 \dots s_i)(\mu_{i+1}, a_{i+1}) > 0$ , or  $a_{i+1} \in \mathbb{R}_0^+$  for each  $0 < i \leq |\pi|$ . The maximal paths of a scheduler  $\mathcal{A}$  are the infinite paths of  $\mathcal{A}$  and the finite paths  $\pi$  such that  $\mathcal{A}(\pi)(\perp) > 0$ . The maximal abstract paths of a scheduler  $\mathcal{A}$  are the abstract counterparts to the maximal paths of  $\mathcal{A}$ . We denote paths<sup>max</sup>( $\mathcal{A}$ ) as the set of maximal paths, and AbsPaths<sup>max</sup>( $\mathcal{A}$ ) as the set of maximal abstract paths of  $\mathcal{A}$ .*

A scheduler resolves all non-deterministic choices of an IOMA, thus making it possible to calculate the probability for each (abstract) path via the path probability. It assigns the unique starting state probability 1, and each following

transition either multiplies the probability that the scheduler assigned to an action, or, if no action was scheduled, the probability of a Markovian action taking place in a certain time interval.

We point out, that upon making a decision, a scheduler has to do so without delay. Hence, there are no additional races between Markovian actions and scheduler decisions. Like before, a scheduler induces a probability to all finite (abstract) paths  $\pi$ , i.e. we can compute the probability, that a path generated by  $\mathcal{A}$  starts with  $\pi$ .

**Definition 5.12.** *Let  $\mathcal{A}$  be a scheduler of an IOMA  $\mathcal{M}$ . Then we define the path probability function  $Q^{\mathcal{A}} : \text{AbsPaths}^{\text{fin}}(\mathcal{M}) \rightarrow [0, 1]$  inductively by  $Q^{\mathcal{A}}(s_0) = 1$ , and*

$$Q^{\mathcal{A}}(\Pi \cdot I \alpha \mu s) = Q^{\mathcal{A}}(\Pi) \cdot \begin{cases} \mathcal{A}(\pi)(\mu) \cdot \mu(\alpha, s) & \text{if } \alpha \in \text{Act} \\ (1 - |\mathcal{A}(\pi)|) \cdot \int_0^T \alpha \cdot e^{-\mathbf{E}(\text{last}(\Pi))t} dt & \text{if } \alpha \in \mathbb{R}^+, \end{cases}$$

where  $I = [0, T] \subseteq \mathbb{R}_0^+$ , and  $\pi$  is the corresponding path to the abstract path  $\Pi$ . The probability of exactly  $\Pi$  is  $Q^{\mathcal{A}}(\Pi) \cdot \mathcal{A}(\pi)(\perp)$ .

A scheduler then defines a unique probability measure  $P_{\mathcal{A}}$  on the set of abstract maximal paths. Therefore, the probability of  $\Pi$  is  $P_{\mathcal{A}}[C_{\Pi}] \stackrel{\text{def}}{=} Q^{\mathcal{A}}(\Pi)$ .

**Definition 5.13.** *The probability space associated to a scheduler  $\mathcal{A}$  of an IOMA  $\mathcal{M}$ , is the probability space given by  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$ , where*

- $\Omega_{\mathcal{A}} = \text{AbsPaths}^{\text{max}}(\mathcal{A})$ ,
- $\mathcal{F}_{\mathcal{A}}$  is the smallest  $\sigma$ -field generated by the set  $\{C_{\Pi} \mid \Pi \in \text{AbsPaths}^{\text{fin}}(\mathcal{M})\}$ , where  $C_{\Pi} = \{\Pi' \in \Omega_{\mathcal{A}} \mid \Pi \sqsubseteq \Pi'\}$ , and
- $P_{\mathcal{A}}$  is the unique probability measure on  $\mathcal{F}_{\mathcal{A}}$  such that  $P_{\mathcal{A}}[C_{\Pi}] = Q^{\mathcal{A}}(\Pi)$  for all  $\Pi \in \text{AbsPaths}^{\text{fin}}(\mathcal{M})$ .

**Trace Distributions.** The construction of the probability associated to a trace follows the same pattern as we have seen in Chapter 4; A scheduler removes all non-determinism, and we remove all information not visible to an external observer, i.e. internal- and Markovian actions, discrete distributions, and state information. Given a scheduler, we can quantify the probability to observe a certain (abstract) trace. Thus, the probability assigned to a set of abstract traces  $X$  is the probability of all abstract paths whose trace is an element of  $X$ .

**Definition 5.14.** *The trace distribution  $\mathcal{D}$  of a scheduler  $\mathcal{A} \in \text{Sched}(\mathcal{M})$ , denoted  $\mathcal{D} = \text{trd}(\mathcal{A})$  is the probability space  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$ , where*

- $\Omega_{\mathcal{D}} = \text{AbsTraces}(\mathcal{M})$ ,
- $\mathcal{F}_{\mathcal{D}}$  is the smallest  $\sigma$ -field generated by the sets  $\{C_{\Sigma} \mid \Sigma \in \text{AbsTraces}^{\text{fin}}(\mathcal{M})\}$ , where  $C_{\Sigma} = \{\Sigma' \in \Omega_{\mathcal{D}} \mid \Sigma \sqsubseteq \Sigma'\}$ , and
- $P_{\mathcal{D}}$  is the unique probability measure on  $\mathcal{F}_{\mathcal{D}}$ , such that

$$P_{\mathcal{D}}(X) = P_{\mathcal{A}}(\text{tr}^{-1}(X)) \text{ for } X \in \mathcal{F}_{\mathcal{D}}.$$

The fact that  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$ , and  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$  define a probability spaces is again left undiscussed here. We refer to [46] for the discussion of the standard measure theoretical arguments.

**Trace Distribution Equivalence.** Trace distributions are the probabilistic counterpart to traces. As such, they quantify the probability to observe abstract traces. It is therefore self-evident to regard trace distributions as constructions with the capability to relate two Markov automata. That is, two automata  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are related, if they induce the same trace distributions. In particular, this means that a trace distribution  $\mathcal{D}$  of  $\mathcal{M}_1$  is contained in the set of trace distributions of  $\mathcal{M}_2$ , if there is a scheduler  $\mathcal{A}$  in  $\mathcal{M}_2$ , such that  $\mathcal{D} = \text{trd}(\mathcal{A})$ .

Again, we write  $\text{trd}(\mathcal{M}, k)$  for the set of trace distributions based on a scheduler of length  $k$ . We write  $\text{trd}(\mathcal{M})$  for the set of all finite trace distributions. Lastly, we write  $\mathcal{M}_1 \sqsubseteq_{TD}^k \mathcal{M}_2$ , if  $\text{trd}(\mathcal{M}_1, k) \subseteq \text{trd}(\mathcal{M}_2, k)$  for  $k \in \mathbb{N}$ , and  $\mathcal{M}_1 \sqsubseteq_{TD}^{fin} \mathcal{M}_2$  if  $\mathcal{M}_1 \sqsubseteq_{TD}^k \mathcal{M}_2$  for some  $k \in \mathbb{N}$ .

## 5.2 Markovian Test Theory

Recall that model-based testing entails automatic test generation, execution and evaluation. To judge correctness of an implementation with respect to its requirements, we utilize a conformance relation. This section aims at establishing all theoretical ingredients necessary for the test framework.

A conformance relation for IOMA is again defined as a relation akin to **ioco**. The next step defines what a test case for Markov automata is, and when an observed trace should be judged as correct via test annotations. Working in a stochastic environment additionally necessitates a statistical verdict. Hence, the sampling process of implementations under test is described, before verdict functions are defined. Lastly, the correctness of the framework is proven.

Markov automata can be seen as conservative extension of probabilistic automata visited in Chapter 4. Hence, many of the definitions in this section are very similar in their purpose and shape. In fact, the conformance relation, as well as test cases and annotations are identical, but need to be redefined for Markov automata. The main difference of the Markovian test theory in comparison to the probabilistic test theory comes in the sampling process and its resulting observations. Incorporating continuous real time demands a more involved treatment of the trace frequency counting function. After all, it is virtually impossible to record the *exact same* time stamps of non-trivial abstract traces more than once.

### 5.2.1 The Conformance relation $\sqsubseteq_{Mar-ioco}$

The purpose of the conformance relation is, once again, to judge whether an implementation model conforms to the requirements specification model. We recalled **ioco** theory in Chapter 3 and extended it to probabilistic automata in Chapter 4. It seems natural to continue this design choice for Markov automata.

In fact, the **pioco** test relation of Chapter 4 was defined in such a way, that it only relies on trace distributions. The careful definition of trace distributions enables us to essentially re-use the **pioco** framework for Markov automata. Like before, we introduce trace distribution prefixes and output continuations.

1. The prefix relation for trace distributions  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$  is the analogue of trace prefixes, i.e.  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$  iff  $\forall \sigma \in [\mathbb{R}_0^+ Act]^{\leq k} : P_{\mathcal{D}}(\sigma) = P_{\mathcal{D}'}(\sigma)$ .
2. The output continuation trace distributions are the probabilistic counterpart of the set  $out_{\mathcal{M}}(\sigma)$  used in the **ioco** relation. For an IOMA  $\mathcal{M}$  and a trace distribution  $\mathcal{D}$  of length  $k$ , the output continuation of  $\mathcal{D}$  in  $\mathcal{M}$  contains all trace distribution  $\mathcal{D}'$  of length  $k + 1$ , such that  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$ , that assigns every abstract trace of length  $k + 1$  ending in input probability 0. We set

$$outcont_{\mathcal{M}}(\mathcal{D}) \stackrel{\text{def}}{=} \{ \mathcal{D}' \in trd(\mathcal{S}, k + 1) \mid \mathcal{D} \sqsubseteq_k \mathcal{D}' \wedge \forall \sigma \in [\mathbb{R}_0^+ Act]^k \mathbb{R}_0^+ Act_I : P_{\mathcal{D}'}(\sigma) = 0 \}.$$

We are now able to define the conformance relation aptly called **Mar-ioco**. Intuitively, an implementation is conforming, if the probability of every output trace can be matched by the specification. This includes the three factors: 1. functional behaviour, 2. probabilistic behaviour and 3. stochastic timing. Note that all three are accounted for by the set of *output continuations*.

**Definition 5.15.** Let  $\mathcal{I}$  and  $\mathcal{S}$  be IOMA over the same action signature with  $\mathcal{I}$  input-enabled. We write  $\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ , if for all  $k \in \mathbb{N}$

$$\forall \mathcal{D} \in trd(\mathcal{S}, k) : outcont_{\mathcal{I}}(\mathcal{D}) \subseteq outcont_{\mathcal{S}}(\mathcal{D}).$$

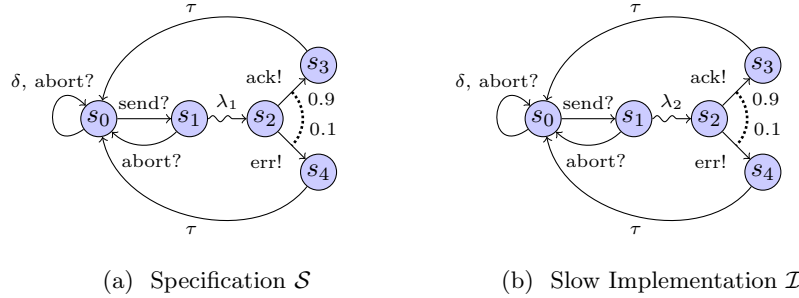


Figure 5.6: Specification IOMA and an erroneous implementation iff  $\lambda_1 \neq \lambda_2$ .

**Example 5.16.** Recall the file transfer protocol from Figure 5.4 studied earlier, which is newly presented in Figure 5.6 After the send input, there is a delay before the file transmission is either acknowledged, or an error is reported. Consider now the scheduler of  $\mathcal{S}$  that schedules send? with probability 1. Its set of output continuations in  $\mathcal{S}$  contains all trace distributions that schedule the outgoing

distribution containing `ack!` and `err!` with probability  $p$  and halts with  $1 - p$ , for  $p \in [0, 1]$ . This holds for the set of output continuations in  $\mathcal{I}$ , but the probability to reach  $s_2$  in a certain amount of time differs from  $\mathcal{S}$  whenever  $\lambda_1 \neq \lambda_2$ . Hence, there are trace distributions in  $\mathcal{I}$ , such that the probability of e.g. `[0,0] send?` `[0,t] ack!` cannot be matched. The implementation is therefore not conforming with respect to **Mar-ioco** in this case.

Note that an input output Markov automaton without Markovian transitions is a regular pIOTS. It is thus natural by our design that **Mar-ioco** and **pioco** coincide on pIOTSs.

**Theorem 5.17.** *For two pIOTSs  $\mathcal{I}, \mathcal{S}$  with  $\mathcal{I}$  input enabled, we have*

$$\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{\text{pioco}} \mathcal{S}.$$

*Proof sketch.* The proof is immediate, if we consider that the conformance relation is defined via trace distributions. Since an IOMA with  $\rightsquigarrow = \emptyset$  is a pIOTS, the notions of schedulers for IOMA and pIOTS coincide.  $\square$

As a result of the preceding theorem and Theorem 4.16, we conclude that **Mar-ioco** conservatively extends **ioco**. That is, both relations coincide on IOTSs.

**Corollary 5.18.** *For two IOTSs  $\mathcal{I}, \mathcal{S}$  with  $\mathcal{I}$  input enabled, we have*

$$\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{\text{ioco}} \mathcal{S}.$$

Not surprisingly, the remaining theorem proven in Chapter 4 carry over to IOMA as well.

**Theorem 5.19.** *Let  $\mathcal{A}, \mathcal{B}$  and  $\mathcal{C}$  be IOMAs and let  $\mathcal{A}$  and  $\mathcal{B}$  be input-enabled, then*

- (i)  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$  if and only if  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ .
- (ii)  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{\text{Mar-ioco}} \mathcal{C}$  imply  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{C}$ .

*Proof sketch.* (i) The fact that finite trace distribution inclusion implies conformance with respect to  $\sqsubseteq_{\text{Mar-ioco}}$  is immediate, if we consider that the conformance relation is defined via trace distributions. The opposite direction follows from the fact that all abstract traces ending in output of  $\mathcal{A}$  assuredly can get assigned the same probabilities in  $\mathcal{B}$  by  $\sqsubseteq_{\text{Mar-ioco}}$ . All abstract traces ending in input are taken care of, because  $\mathcal{A}$  and  $\mathcal{B}$  are input enabled, and all such distributions are input-reactive. (ii) is a direct consequence of (i).  $\square$

## 5.2.2 Test Cases and Annotations

We define test cases over an action signature  $(Act_I, Act_O)$  for input output Markov automata. In fact, the definition coincides with test cases for pIOTSs, but is reiterated here for the sake of completeness. However, this time, test cases

are formally defined as IOMA, as opposed to pIOTS. In a purely theoretical setting, we are interested in the result of a parallel composition of a test case and an implementation model. Recall that we strictly only allow parallel composition of two IOMA, as opposed to an IOMA and a pIOTS (Definition 5.6). Thus, the repeated definition is necessary in that regard.

Like before, a test is a collection of traces that represent behaviour of a tester, and that is summarized as an IOMA in tree structure. The action signature describes the potential interaction of the test case with the implementation. In each state, the test may either stop, wait for a response of the system, or provide some stimulus. When a test is waiting for a system response, it has to take into account all potential outputs including the situation that the system provides no response at all, modelled by  $\delta$ .

**Definition 5.20.** *A test or test case over an alphabet  $(Act_I, Act_O)$  is an IOMA*

$$t = \langle S, s_0, Act_I^t, Act_O^t, \emptyset, \Delta, \emptyset \rangle,$$

*that has the alphabet's outputs as inputs and vice-versa, i.e.  $Act_I^t = Act_O \cup \{\delta\}$  and  $Act_O^t = Act_I \setminus \{\delta\}$ , and that is a finite, internally deterministic, and connected tree. In addition, for all its discrete distributions  $\mu$  we have  $\mu \equiv \text{Dirac}$ , and for every state  $s \in S$  we require:*

- *enabled*( $s$ ) =  $\emptyset$ , or
- *enabled*( $s$ ) =  $Act_I^t$ , or
- *enabled*( $s$ )  $\in Act_O^t$ .

*A test suite  $T$  is a set of test cases. A test case (suite resp.) for an IOMA  $S$ , is a test case (suite resp.) over its action signature  $(Act_I, Act_O)$ , if we additionally require in item 3, that*

- $\mu(a, s') > 0$  with  $a \in Act_O^t$  implies the existence of  $\sigma \in \text{traces}^{\text{fin}}(S)$ , such that  $\sigma \cdot \mathfrak{t}a \in \text{traces}^{\text{fin}}(S)$  for some  $\mathfrak{t} \in \mathbb{R}_0^+$ .

The last item in the definition of test cases ensures that only specified inputs are provided. After all, a test may only judge about the correctness of specified behaviour. This is referred as *input-minimal* in the literature [169].

**Remark 5.21.** *Although test cases are defined as IOMAs, they do not make use of Markovian transitions. While it might seem appealing to equip test cases with the potential to wait before proceeding with the next action, the capabilities of Markov automata are too limited to be of practical relevance. That is, if a test case were supposed to wait, it may only do so with an exponentially distributed delay. The potential of waiting time is thus limited by only allowing to wait  $T$  time units on average, as opposed to exactly  $T$  time units, or other. Moreover, allowing Markovian actions in test cases results in  $\mathcal{I} \parallel t$  not being internally deterministic for any implementation IOMA  $\mathcal{I}$ . This causes technical difficulties in the proofs of correctness for our framework with respect to compositionality of trace distributions. Therefore, we opt for the IOTS design of test cases instead.*



**Test Annotation.** In order to pin down the behaviour, which we deem as acceptable/correct, each trace of the test is annotated with *pass* or *fail* verdicts, determined by the requirements specification.

We have shown that the **Mar-ioco** relation conservatively extends the **ioco** relation. Since annotations are solely responsible for functional correctness, it suffices to directly transfer **ioco** test annotations to Markovian test theory. Note that time stamps in traces do not play a role for the annotation label.

**Definition 5.22.** For a given test  $t$  a test annotation is a function

$$ann : traces^{com}(t) \longrightarrow \{pass, fail\}.$$

A pair  $\hat{t} = (t, ann)$  consisting of a test and a test annotation is called an annotated test. The set of all such  $\hat{t}$ , denoted by  $\hat{\mathcal{T}} = \{(t_i, ann_i)_{i \in \mathcal{I}}\}$  for some index set  $\mathcal{I}$ , is called an annotated test suite. If  $t$  is a test case for an IOMA  $\mathcal{S}$  with signature  $(Act_I, Act_O)$ , we define  $ann_{Mar-ioco}^{\mathcal{S}} : traces^{com}(t) \longrightarrow \{pass, fail\}$  by

$$ann_{Mar-ioco}^{\mathcal{S}}(\sigma) = \begin{cases} fail & \text{if } \exists \varrho \in traces^{fn}(\mathcal{S}), a \in Act_O : \\ & \varrho \mathbf{t} a \sqsubseteq \sigma \wedge \varrho \mathbf{t} a \notin traces^{fn}(\mathcal{S}) \\ pass & \text{otherwise.} \end{cases}$$

**Example 5.23.** Since test cases for IOMA and their annotations coincide with those for pIOTS, we refer to Example 4.21 for an illustration.

### 5.2.3 Test Evaluation and Verdicts

Since discrete probabilistic choices and stochastic time delay are integral parts of Markov automata, there is a twofold evaluation process of functional and probabilistic behaviour, respectively. While functional behaviour is assessed via test annotation as in classical **ioco** theory (Chapter 3), the probabilistic and stochastically timed behaviour is assessed by gathering a sample of traces.

Although we covered the latter for pIOTSs in Chapter 4, we did not account for stochastic time delay. Hence, we shall revisit the counting of (abstract) trace frequencies in the presence of continuous real time in this subsection. Recall that we assessed trace frequencies in a sample via the function *freq* for pIOTSs. In the presence of continuous real time, this process is slightly more involved, but similar in shape and purpose.

In the following we focus on the novelties that continuous real time brings with it, and reiterate fundamental definitions. For a more involved explanation on the test process for discrete probabilities only, we refer to Chapter 4.

**Statistical Testing.** The experiment consists of a push-button experiment in the sense of [134]. We assume a black-box timed trace machine is given, together with inputs, a time and an action window, and a reset button, as illustrated in Figure 5.7. An external observer records each individual execution before the reset button is pressed, and a new execution starts. A clock that increases

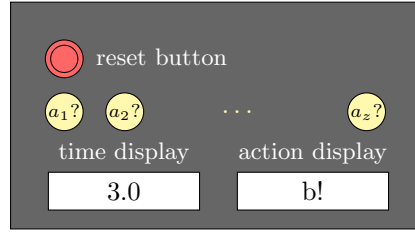


Figure 5.7: Black-box timed trace machine assumed to operate based on an underlying IOMA. The black-box possesses a reset button, and various input buttons, alongside an action window showing the most recently executed action, and a time window showing the relative time since the last observable action.

is started, and is stopped once the next visible action is recorded. We assume that recording an action resets the clock. Thus, the recordings of the external observer matches the notion of (abstract) traces.

After a sample of sufficient size was collected, we compare the collected frequencies of abstract traces to their expected frequencies according to the specification. If the empiric observations are close to the expectations, we accept the probabilistic behaviour of the implementation. We point out, that the latter includes stochastically delayed time.

**Sampling.** At the beginning of the experiment, we set the parameters for sample length  $k \in \mathbb{N}$ , sample size  $m \in \mathbb{N}$ , and level of significance  $\alpha \in (0, 1)$ . That is, we choose the maximal length of an individual experiment, the sample size and the probability of erroneously rejecting a correct implementation.

We establish probabilistic correctness by checking the abstract trace frequencies contained in a sample versus their expectancy with respect to the specification. Like before, the latter is not immediately given in the face of non-determinism in the specification, and requires a scheduler.

Thus, we assume each iteration of the sampling process is governed by a scheduler, resulting in a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{I})$ . In order for any statistical reasoning to work, we assume that  $\mathcal{D}$  is the same in every iteration. Consequently, the implementation chooses a trace distribution, and the trace distribution chooses a trace to execute.

**Frequencies and Expectations.** To quantify how *close* a sample is to its expectations, we require a notion of *distance*. Our goal is to evaluate the deviation of a collected sample to the expected distribution. Thus, we require:

1. a metric space, enabling the quantification of distances between measures,
2. the frequency measure of abstract traces in a sample, and
3. the expected measure of abstract traces of the specification under trace distribution  $\mathcal{D}$ .

For an IOMA  $\mathcal{M}$ , we define the metric space as  $(\text{Meas}(\mathcal{M}), \text{dist})$ , where

$$\text{dist}(u, v) \stackrel{\text{def}}{=} \sup_{\sigma \in [\mathbb{R}_0^+ \text{Act}]^{\leq k}} |u(\Sigma) - v(\Sigma)|$$

is the maximal variation distance of two measures  $u$  and  $v$ . It is easy to check that  $dist$  is a metric, and consequently  $(Meas(\mathcal{M}), dist)$  is a metric space.

We proceed by defining the two measures that need to be compared. The function assessing the frequencies of traces within a sample  $O = (\sigma_1, \dots, \sigma_m)$  cannot be directly transferred from the pIOTS scenario. Depending on the accuracy of time measurement, it is unlikely to record the *exact* same timed trace more than once. To illustrate, consider the two abstract traces

$$\begin{aligned}\sigma_1 &= 0.5 a? 0.6 b!, & \text{and} \\ \sigma_2 &= 0.6 a? 0.5 b!\end{aligned}$$

Evidently,  $\sigma_1$  and  $\sigma_2$  wield the same actions in  $a?$  followed by  $b!$ . However, their timed behaviour does not match precisely. Therefore, we group traces in classes based on the same visible action behaviour. For a given trace  $\sigma$ , its class  $\Sigma_\sigma$  is the set of all traces  $\varrho \in O$ , such that  $act(\varrho) = act(\sigma)$ . A sample of length  $k$  and width  $m$  then induces a frequency measure as the function  $freq : [(\mathbb{R}_0^+ \times \mathbb{R}_0^+)Act]^{\leq k \times m} \rightarrow Meas([\mathbb{R}_0^+ Act]^{\leq k})$ , such that

$$freq(O)(\Sigma) = \frac{|\Sigma_\sigma|}{m} \prod_{i=1}^k \frac{|\{\varrho \in \Sigma_\sigma \mid t_i^\varrho \leq t_i^\sigma\}|}{|\Sigma_\sigma|},$$

where  $t_i^\varrho$  denotes the  $i$ -th time stamp of trace  $\varrho$ . Note that this function assumes the independence of all time intervals from each other. In particular, working with Markov automata warrants this approach, as there are no *clocks* attached to delay transitions, and the delay is memoryless. Thus, the  $i$ -th time intervals of all  $\varrho$  are ordered increasingly and compared to  $\sigma$ . We point out, that the distributions for each time stamp in a trace thus converge to the true underlying distribution by the Glivenko-Cantelli Theorem [77].

The last missing ingredient is the expected measure according to a specification. In order to resolve all non-deterministic choices, assume a trace distribution  $\mathcal{D}$  is given. We treat each iteration of the sampling process of the implementation as Bernoulli trial. Recall that a Bernoulli trial has two outcomes: *success* with probability  $p$  and *failure* with probability  $1 - p$ . For any trace  $\sigma$ , we say that success occurred at position  $i$  of the sample, if  $\sigma = \sigma_i$ . Therefore, let  $X_i \sim Ber(P_{\mathcal{D}}(\Sigma))$  be Bernoulli distributed random variables for  $i = 1, \dots, m$ . Let  $Z = \frac{1}{m} \sum_{i=1}^m X_i$  be the empiric mean with which we observe  $\sigma$  in a sample. Note that the expected probability under  $\mathcal{D}$  is then calculated as

$$\mathbb{E}^{\mathcal{D}}(Z) = \mathbb{E}^{\mathcal{D}}\left(\frac{1}{m} \sum_{i=1}^m X_i\right) = \frac{1}{m} \sum_{i=1}^m \mathbb{E}^{\mathcal{D}}(X_i) = P_{\mathcal{D}}(\Sigma).$$

Hence, the expected probability for each abstract trace  $\Sigma$  is the probability of  $\Sigma$  under trace distribution  $\mathcal{D}$ , as should be expected. Since this holds for all abstract traces, the expected measure is given by  $\mathbb{E}^{\mathcal{D}}$ .

**Example 5.24.** *Returning to the example of  $\sigma_1 = 0.5 a? 0.6 b!$  and  $\sigma_2 =$*

0.6 a? 0.5 b!. Assume  $O = \{\sigma_1, \sigma_2\}$ . Then

$$\begin{aligned} \text{freq}(O) ([0, 0.5] a? [0, 0.5] b!) &= \frac{2}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}, \\ \text{freq}(O) ([0, 0.5] a? [0, 0.6] b!) &= \frac{2}{2} \cdot \frac{1}{2} \cdot \frac{2}{2} = \frac{1}{2}, \\ \text{freq}(O) ([0, 0.6] a? [0, 0.6] b!) &= \frac{2}{2} \cdot \frac{2}{2} \cdot \frac{2}{2} = 1. \end{aligned}$$

**Acceptable Outcomes.** A sample  $O$  is accepted, if  $\text{freq}(O)$  lies within some distance  $r_\alpha$  of the expected measure  $\mathbb{E}^{\mathcal{D}}$ . All measures deviating at most  $r_\alpha$  from the expected measures are contained within the ball  $B_{r_\alpha}(\mathbb{E}^{\mathcal{D}})$ . The actual  $r_\alpha$  is chosen, such that the error of accepting an erroneous sample is limited, while keeping the error of rejecting a correct sample smaller than  $\alpha$ , i.e.

$$r_\alpha \stackrel{\text{def}}{=} \inf\{r \in \mathbb{R}_0^+ \mid P_{\mathcal{D}}(\text{freq}^{-1}(B_r(\mathbb{E}^{\mathcal{D}}))) \geq 1 - \alpha\}.$$

**Definition 5.25.** For  $k, m \in \mathbb{N}$  and an IOMA  $\mathcal{M}$  the acceptable outcomes of  $\mathcal{D} \in \text{trd}(\mathcal{M}, k)$  of significance level  $\alpha \in (0, 1)$  are given by the set

$$\text{Obs}(\mathcal{D}, \alpha, k, m) = \{O \in [\mathbb{R}_0^+ \text{Act}]^{\leq k \times m} \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha\}$$

We obtain the set of acceptable outcomes of  $\mathcal{M}$  by

$$\text{Obs}(\mathcal{M}, \alpha, k, m) = \bigcup_{\mathcal{D} \in \text{trd}(\mathcal{M}, k)} \text{Obs}(\mathcal{D}, \alpha, k, m).$$

Like in the pIOTS case, the set of acceptable outcomes consists of all possible samples, we are willing to accept as *close enough* to the expectations. Note that this takes *all possible* trace distributions of the IOMA  $\mathcal{M}$  into consideration. As such, the construction of the set of acceptable outcomes has a striking similarity to the one defined for pIOTSs (Definition 4.23). This is due to the fact, that both sets are built on trace distributions of the respective underlying modelling formalism.

The set of acceptable outcomes of an IOMA  $\mathcal{M}$  has two properties, reflecting the error of false rejection, and the error of false acceptance respectively.

1. If a sample was generated by a truthful trace distribution of the specification, we correctly accept it with probability higher than  $1 - \alpha$ , i.e.

$$P_{\mathcal{D}}(\text{Obs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$$

2. If a sample was generated by a trace distribution not admitted by the specification, the chance of falsely accepting it is smaller than some  $\beta_m$ .

Again,  $\alpha$  is the *a priori* defined level of significance, and  $\beta_m$  is unknown, but minimal by construction. Additionally,  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ . Thus, the error of falsely accepting an observation decreases with increasing sample size.

**Remark 5.26.** The set of acceptable outcomes comprises samples of the form  $O \in [\mathbb{R}_0^+ \text{Act}]^{\leq k \times m}$ . In order to align observations with the **Mar-ioco** relation, we define the set of acceptable output outcomes like follows

$$\begin{aligned} \text{OutObs}(\mathcal{D}, \alpha, k, m) &= \{O \in ([\mathbb{R}_0^+ \text{Act}]^{\leq k-1} \cdot \mathbb{R}_0^+ \text{Act}_O)^m \mid \\ &\quad \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha\}. \end{aligned}$$

**Verdict Functions.** With all necessary components in place, the following decision process summarizes if an implementation fails a test, or test suite respectively, based on a functional or a statistical verdict. The overall pass verdict is given iff both sub-verdicts yield a *pass*.

**Definition 5.27.** *Given a specification IOMA  $\mathcal{S}$ , an annotated test  $\hat{t}$  for  $\mathcal{S}$ ,  $k, m \in \mathbb{N}$ , where  $k$  is given by the length of the longest trace of  $\hat{t}$ , and  $\alpha \in (0, 1)$ , we define the functional verdict as  $v_{func} : IOMA \times IOMA \rightarrow \{pass, fail\}$ , with*

$$v_{func}(\mathcal{I}, \hat{t}) = \begin{cases} pass, & \text{if } \forall \sigma \in \text{traces}^{com}(\mathcal{I} \parallel \hat{t}) : \text{ann}_{Mar-ioco}^{\mathcal{S}}(\sigma) = pass \\ fail, & \text{otherwise,} \end{cases}$$

the statistical verdict as the function  $v_{prob} : IOMA \times IOMA \rightarrow \{pass, fail\}$ , with

$$v_{prob}(\mathcal{I}, \hat{t}) = \begin{cases} pass, & \text{if } \forall \mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}) \exists \mathcal{D}' \in \text{trd}(\mathcal{S}, k) : \\ & P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha \\ fail, & \text{otherwise,} \end{cases}$$

and the overall verdict as  $V : IOMA \times IOMA \rightarrow \{pass, fail\}$ , with

$$V(\mathcal{I}, \hat{t}) = \begin{cases} pass, & \text{if } v_{func}(\mathcal{I}, \hat{t}) = v_{prob}(\mathcal{I}, \hat{t}) = pass \\ fail, & \text{otherwise.} \end{cases}$$

An implementation passes a test suite  $\hat{T}$ , if it passes the overall verdict for all tests  $\hat{t} \in \hat{T}$ .

Even though IOMA include three properties in 1. functional behaviour, 2. discrete probabilistic behaviour, and 3. continuous real time we only have two verdicts. We point out, that continuous real time is only present in the form of stochastic delay. Thus, on the purely mathematical level the decision whether or not a delay in the implementation adheres to the one specified, is covered by the probabilistic verdict  $v_{prob}$ . It is only on the practical side of things, that a new decision procedure is needed. We study this in Section 5.3.

#### 5.2.4 Correctness of the Framework

In Chapter 4 we saw that soundness and completeness are no absolute entities, when referring to probabilistic systems - A system of inherently probabilistic nature may only be sound and complete *with some probability*. This is due to the fact, that the statistical verdict is of probabilistic nature. Thus, the capability to judge probabilistic systems as correct, is largely dependent on the errors of *first and second kind*. Since IOMA are a conservative extension of pIOTSSs, it is only natural that this trade-off carries over.

We reiterate: Soundness incorporates the probability that the *fail* verdict is erroneously assigned to a conforming implementation, while completeness is concerned with the probability to assign the *pass* verdict to a non-conforming implementation. A test suite may only fulfil these properties with a guaranteed (high) probability.

**Definition 5.28.** Let  $\mathcal{S}$  be a specification IOMA over an action signature  $(Act_I, Act_O)$ ,  $\alpha \in (0, 1)$  be the level of significance, and  $\hat{T}$  an annotated test suite for  $\mathcal{S}$ . Then

- $\hat{T}$  is sound for  $\mathcal{S}$  with respect to  $\sqsubseteq_{Mar-ioco}$ , if for all input-enabled IOMAs  $\mathcal{I}$  and sufficiently large  $m \in \mathbb{N}$  it holds for all  $\hat{t} \in \hat{T}$

$$\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = pass.$$

- $\hat{T}$  is complete for  $\mathcal{S}$  with respect to  $\sqsubseteq_{Mar-ioco}$ , if for all input-enabled IOMAs  $\mathcal{I}$  and sufficiently large  $m \in \mathbb{N}$ , there is at least one  $\hat{t} \in \hat{T}$ , such that

$$\mathcal{I} \not\sqsubseteq_{Mar-ioco} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = fail.$$

Soundness expresses for a given  $\alpha \in (0, 1)$ , that there is a  $1 - \alpha$  chance that a correct system passes the annotated test suite for sufficiently large sample size  $m$ . This relates to false rejection of a correct hypothesis, or rejection of a correct implementation, respectively.

**Theorem 5.29.** Each annotated test for an IOMA  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{Mar-ioco}$ .

*Proof sketch.* The proof sketch is similar to the one for pIOTS (Theorem 4.27), since the verdicts were defined in a similar manner, and both proofs rely on trace distributions. We assume  $\mathcal{I}$  to be an input enabled IOMA and  $\hat{t}$  to be an annotated test case for the IOMA  $\mathcal{S}$ . The proof requires to show that  $\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{t}) = pass$ . The latter is split into functional verdict  $v_{func}(\mathcal{I}, \hat{t})$  and probabilistic verdict  $v_{prob}(\mathcal{I}, \hat{t})$ , which are treated separately.

- The functional verdict requires to show that the annotation of every trace  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$  is *pass*. Note that the annotations do not depend on time delay, and the proof is therefore equivalent to the pIOTS case. We start by taking a prefix  $\sigma'$  of  $\sigma$ . We can assume  $\sigma'$  to be a trace of  $\mathcal{S}$  – such a trace surely exists, as the empty trace  $\varepsilon$  is always a prefix of  $\sigma$ . The proof consists of showing that  $\sigma' \mathfrak{t} a \sqsubseteq \sigma$  is a trace in  $\mathcal{S}$  for all outputs  $a$  and some  $\mathfrak{t} \in \mathbb{R}_0^+$ . If no such output exists, then  $\sigma$  solely consists of inputs and gets the *pass* label by the definition of annotations (Definition 5.22). Without loss of generality, we choose a trace distribution  $\mathcal{D} \in trd(\mathcal{S})$  which assigns positive probability to  $\sigma'$ . Since  $\sigma' \mathfrak{t} a$  is a trace in  $\mathcal{I} \parallel \hat{t}$  it must also be a trace in  $\mathcal{I}$ . Hence, there is a trace distribution  $\mathcal{D}' \in outcont_{\mathcal{I}}(\mathcal{D})$  that assigns positive probability to it. By  $\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}$  we know that  $\mathcal{D}'$  is also a trace distribution of  $\mathcal{S}$ , which implies  $\sigma' \mathfrak{t} a$  to be a trace in  $\mathcal{S}$ . This results in the *pass* annotation, and thus the functional *pass* verdict.
- The probabilistic verdict requires that all observations of  $\mathcal{I} \parallel \hat{t}$  get assigned a measure greater or equal to  $1 - \alpha$  for a trace distribution of  $\mathcal{S}$ . The proof consists of letting  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t})$ , and showing that  $\mathcal{D} \in trd(\mathcal{S})$ . This is

sufficient by merit of the definition of observations (Definition 5.25), i.e. we always have  $P_{\mathcal{D}}(Obs(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$  for any  $\mathcal{D}$ .

The proof consists of three steps: 1.  $\mathcal{D}$  might still schedule positive probability to input actions in the  $k$ -th step; we choose a new scheduler that assigns all this probability to halting instead. The measure of  $OutObs$  is unaffected by this, as it only consists of traces ending in output. 2. We show that  $\mathcal{D}$  is a trace distribution of  $\mathcal{I}$ . In particular, this lets us use the assumption  $\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}$  in the next step. Intuitively,  $\mathcal{I} \parallel \hat{t}$  is internally deterministic by the construction of test cases. Hence, there is an injective mapping from paths of  $\mathcal{I} \parallel \hat{t}$  to the paths of  $\mathcal{I}$ . We can then construct a scheduler in  $Sched(\mathcal{I})$  that can copy the behaviour of the scheduler of  $\mathcal{I} \parallel \hat{t}$  step-by-step. Lastly, 3. we apply the assumption  $\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S}$  to show  $\mathcal{D} \in trd(\mathcal{S})$ . This ultimately yields  $v_{prob}(\mathcal{S}, \hat{t}) = pass$ .

□

Completeness of a test suite is inherently a theoretical result. Since loops are allowed in the system models, we naturally require test suites of infinite size. Moreover, there is the chance of falsely accepting an erroneous hypothesis or implementation, respectively. However, the latter is bound from above by construction, and decreases for larger sample sizes (Definition 5.25).

**Theorem 5.30.** *The set of all annotated test cases for an IOMA  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{Mar-ioco}$  for sufficiently large sample size.*

*Proof sketch.* The proof is again similar to the one for pIOTs (Theorem 4.28). We assume  $\mathcal{I}$  to be an input enabled IOMA and  $\hat{T}$  to be the test suite containing *all* annotated test cases for  $\mathcal{S}$ . We prove the statement by showing that  $\mathcal{I} \not\sqsubseteq_{Mar-ioco} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = fail$ . By the definition of verdicts (Definition 5.27) this requires the existence of a test case  $\hat{t}$  such that either the functional verdict, or the probabilistic verdict fails. Assuming  $\mathcal{I} \not\sqsubseteq_{Mar-ioco} \mathcal{S}$  implies the existence of a trace ending in output in  $\mathcal{I}$ , whose probability cannot be matched under any trace distribution in  $\mathcal{S}$ . This can have two causes: 1. the mismatch arises due to the trace not being present in  $\mathcal{S}$ , or 2. all such traces are present in  $\mathcal{S}$ , and the mismatch arises due to different probability distributions in  $\mathcal{I}$  and  $\mathcal{S}$ . We relate the first to  $v_{func}(\mathcal{I}, \hat{t}) = fail$  and the second to  $v_{prob}(\mathcal{I}, \hat{t}) = fail$  for some test  $\hat{t}$ .

- We show that the first case implies  $v_{func}(\mathcal{I}, \hat{t}) = fail$ . The proof is straightforward, and requires to show that the trace in  $\mathcal{S}$  has the *fail* annotation for some annotated test case. This follows immediately from the definitions of test cases and their annotations (Definitions 5.20 and 5.22), and by  $\hat{T}$  containing *all* test cases for  $\mathcal{S}$ .
- We show that the second case implies  $v_{prob}(\mathcal{I}, \hat{t}) = fail$ . According to the definition of verdicts (Definition 5.27), we need to show that there is a test case  $\hat{t}$  and a trace distribution of  $\mathcal{I} \parallel \hat{t}$  under which all observations

get assigned a measure smaller than  $1 - \alpha$  for all trace distributions of  $\mathcal{S}$  for sufficiently large  $m$ . By the definition of acceptable outcomes (Definition 5.25) we know that  $P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) < \beta_m$  for some  $\beta_m \xrightarrow{m \rightarrow \infty} 0$  whenever  $\mathcal{D} \neq \mathcal{D}'$ . By initial assumption, we know this holds for all  $\mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$  with  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$ . This estimate is unaffected by increasing the search space to all  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k + 1)$  instead, since the measure of the set  $\text{OutObs}$  is maximised for trace distributions of  $\text{outcont}$ .

It remains to be shown that there is a test case  $\hat{t}$  for all trace distributions  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$ , such that  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t})$ . By assumption, all traces getting assigned a positive probability under  $\mathcal{D}$  are traces in  $\mathcal{S}$ . We select a test case  $\hat{t}$ , that contains all these traces. In particular, all such traces end in output by assumption. The IOTS structure and absence of internal- or Markovian actions ensures that  $\mathcal{I} \parallel \hat{t}$  is internally deterministic, and there is no interleaving. This means that a scheduler of  $\mathcal{I} \parallel \hat{t}$  can copy the behaviour of the scheduler inducing  $\mathcal{D}$  step-by-step. A careful construction shows that  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t})$ , for which all  $\mathcal{D}' \in \text{trd}(\mathcal{S})$  assign all of  $\mathcal{D}'$ 's observations a measure smaller than  $1 - \alpha$  for sufficiently large  $m$ . This yields  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$ .

The mismatch of probability of a trace has to belong to one of the two mentioned cases, and thus  $V(\mathcal{I}, \hat{T}) = \text{fail}$ .  $\square$

### 5.3 Implementing Markovian Testing

The previous section laid the theoretical foundations of the Markov automata-based testing framework. In order to cope with the rather abstractly defined concepts in practice, we present applicable procedures. We point out that the majority of them carries over from the pIOTS case. However, having stochastically delayed continuous real-time in the underlying modelling formalism demands slight adjustments from the ones seen in Section 4.3.

First, we discuss how the goodness of fit-method we saw in form of Pearson's  $\chi^2$  test is adjusted to cope with real time. While we keep Pearson's  $\chi^2$  test as a necessary condition for further reasoning, the overall procedure is enriched with confidence interval analysis on the time stamps. In particular this means that the corresponding waiting times recorded in traces are grouped together and compared to their prescribed Markovian parameter  $\lambda$  in the model. Note that additional assumptions are necessary to enable a clean and efficient framework.

Next, we present the intricate interplay that stochastically delayed time and the notion of quiescence induces in a practical testing scenario. The presence of inherent delay requires us to wait before possibly judging the implementation quiescent. On a purely formal level quiescence means the *indefinite* absence of outputs. In practice, this is done via global waiting times [15, 29] indicating when an implementation is judged as quiescent *indefinitely*. However, in the interest of the *global time* of the overall test procedure we would like to keep



this waiting time at a minimum. We discuss sensible solutions to deal with this dilemma.

Lastly, we point out that test cases for IOMA are *essentially* IOTSs. Hence, the test generation algorithms seen in Chapters 3 and 4 can directly be transferred, and are not further discussed here.

The section ends with a summary of all practical steps, that enable the model-based test approach when working with the Markovian formalism.

### 5.3.1 Goodness of Fit

We need practically applicable methods to decide about the verdicts given by Definition 5.27. The functional verdict requires that all traces of the implementation under test are labelled with the *pass* label. However, in practice we do not have access to *all* traces, and rely on those encountered during testing instead. While the functional verdict is determined via test annotations, we are currently lacking a procedure to decide about the probabilistic verdict, i.e. the probabilistic (incl. Markovian) correctness of an implementation. This procedure consists of two subcomponents: Pearson's  $\chi^2$  hypothesis test for discrete probabilities, and interval estimation for time stamps.

**Pearson's  $\chi^2$  test.** A method we used in the case of discrete probabilities only, was the  $\chi^2$  hypothesis test, cf. Section 4.3. Our approach was based on a theorem known from the literature [40]: Between two pIOTSs, there is an embedding of trace distributions iff there is an embedding on the set of observations. Neither the set of trace distributions, nor the entire set of observations of the implementation are directly accessible to us. Instead, we pose a null-hypothesis for a statistical hypothesis test. Its outcome is based on a sample  $O$  taken from the implementation under test. Should  $O$  prove to be a sample of the set  $OutObs(\mathcal{S}, \alpha, k, m)$  for some  $\alpha \in (0, 1)$ , we are willing to accept the hypothesis of the embeddings of observations. This, in turn, lets us conclude the embedding of trace distributions, and consequently, the statistical *pass* verdict.

In the Markovian case, we argue along the same lines. However, only applying the  $\chi^2$  hypothesis test is insufficient, as it does not take into account the delay times observed in abstract traces. Nonetheless, passing the  $\chi^2$  test is a necessary condition for an implementation to be accepted.

We reiterate how the  $\chi^2$  score of a sample is calculated. Therefore, for a finite trace  $\sigma = t_1 a_1 t_2 a_2 \dots t_n a_n$ , let  $\bar{\sigma} = \mathbb{R}_0^+ a_1 \mathbb{R}_0^+ a_2 \dots \mathbb{R}_0^+ a_n$  be its *time closure*. Then the empiric  $\chi^2$  score is given as

$$\chi^2 \stackrel{\text{def}}{=} \sum_{\bar{\sigma} \in \{\bar{\sigma} \mid \sigma \in O\}} \frac{(|\{\bar{\varrho} \mid \varrho \in O \wedge \bar{\varrho} = \bar{\sigma}\}| - m\mathbb{E}^{\mathcal{D}}(\bar{\sigma}))^2}{m\mathbb{E}^{\mathcal{D}}(\bar{\sigma})}. \quad (5.1)$$

Equation (5.1) essentially compares observed traces to their respective expected counterparts. We use the time closure of traces to ignore time stamps for the  $\chi^2$

analysis. The empirical  $\chi^2$  value is compared to critical values of given degrees of freedom and levels of significance. These values can be calculated, or looked up in a  $\chi^2$  table (Appendix A.2). In case the empiric  $\chi^2$  score is below the given threshold  $\chi_{crit}^2$ , the hypothesis is accepted, or consequently rejected if it is not.

However, the expected value  $\mathbb{E}^{\mathcal{D}}$  depends on the resulting trace distribution of a scheduler. Thus, finding a scheduler such that  $\chi^2 \leq \chi_{crit}^2$  turns (5.1) into a minimisation problem (or satisfaction problem, respectively).

$$\min_{\mathcal{D} \in \text{trd}(\mathcal{S}, k)} \sum_{\bar{\sigma} \in \{\bar{\sigma} \mid \sigma \in O\}} \frac{(|\{\bar{\varrho} \mid \varrho \in O \wedge \bar{\varrho} = \bar{\sigma}\}| - m\mathbb{E}^{\mathcal{D}}(\bar{\sigma}))^2}{m\mathbb{E}^{\mathcal{D}}(\bar{\sigma})}. \quad (5.2)$$

The probability of a trace is given by a scheduler and the corresponding path probability function, (Definition 5.12). Hence, by construction, we need to find probabilities  $p$  used by a scheduler to resolve non-determinism. This turns (5.2) into a minimisation or constraint solving problem of a rational function  $f(p)/g(p)$  with inequality constraints on the vector  $p$ . As [137] show: minimizing general rational functions is **NP**-hard.

**Remark 5.31.** *Recall the discussion of Remark 5.10 on schedulers with the capability to wait before scheduling the next action. The minimisation step presented in Equation (5.2) highlights our choice of schedulers in Definition 5.8. We chose not to equip schedulers with the potential to wait before choosing the next action, as this would make the minimisation or constraint solving step needlessly complex to warrant any application outside of trivial scenarios.*

*An illustrative example is given by the trace  $\sigma = 0.1 a!$ . Even for a simple trace like  $\sigma$ , it would not be clear whether the time stamp 0.1 originated from the implementation, or a scheduler that decided to wait before scheduling the  $a!$  action. Moreover, it is possible that both the implementation and the scheduler contributed to the delay. Backwards analysis on the best fitting scheduler of a non-trivial sample, and deciding the null-hypothesis becomes too complex to utilize this approach in practice, and warrants the choice of non-waiting schedulers.*

**Interval estimation.** While passing the  $\chi^2$  test is a necessary condition in order to accept the null-hypothesis  $O \in \text{OutObs}(\mathcal{S}, \alpha, k, m)$ , it is not yet sufficient. Note that the used hypothesis test did neglect time information completely, and we are currently lacking a metric to decide whether observed time delays correspond to a prescribed exponential distribution. We overcome this with the help of interval estimation on the parameter of the exponential distributions.

Assume values  $x_1, \dots, x_n$  are given, and suppose we ought to test whether the values follow an exponential distribution with parameter  $\lambda$ . Our goal is to construct the confidence interval for given  $\alpha \in (0, 1)$  of these values, i.e. upon further sampling and interval estimations, there is a  $1 - \alpha$  chance that the true parameter  $\lambda_{real}$  is contained in the interval. The  $1 - \alpha$  confidence interval for

the given values is given by

$$\left[ \frac{\chi_{1-\alpha/2,2n}^2}{2\sum_{i=1}^n x_i}, \frac{\chi_{\alpha/2,2n}^2}{2\sum_{i=1}^n x_i} \right], \quad (5.3)$$

where  $\chi_{\alpha,2n}^2$  is the  $1 - \alpha$  quantile of the  $\chi^2$  distribution of  $2n$  degrees of freedom. We refer to Appendix A.1 Proposition A.20 for the derivation of the presented interval.

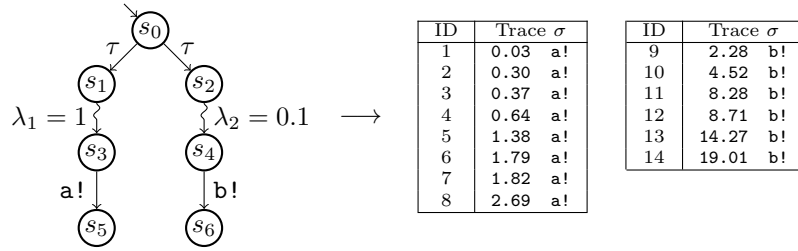


Figure 5.8: Example specification model IOMA and sample observation.

**Example 5.32.** Figure 5.8 shows an example specification model alongside a potential sample observation from a real implementation. In state  $s_0$  are two outgoing internal  $\tau$  actions, followed by two Markovian transitions in  $s_1$  and  $s_2$  respectively. In states  $s_3$  and  $s_4$  we either observe action  $a!$  or  $b!$ , respectively.

The sample shows 14 recorded traces of length one, thus  $m = 14$  and  $k = 1$ . We ordered the traces in an increasing manner according to their time stamps. Note that this is not possible in general, as there is no such simple order on higher dimensional vectors.

There are two steps to assess whether the observed data is a truthful sample of the specification model with a confidence of  $\alpha = 0.1$ : 1. find a trace distribution that minimises the  $\chi^2$  statistic and 2. evaluate two confidence intervals to assess whether the observed time data is a sample of  $\lambda_1 = 1$  and  $\lambda_2 = 0.1$ , respectively.

There are two classes of traces solely based on the action signature: ID 1-8 with  $a!$  and ID 9-14 with  $b!$ . Let  $p$  be the probability that a scheduler assigns to taking the left branch in  $s_0$ , and  $1 - p$  the probability for the right branch. Upon drawing a sample with  $m = 14$  we expect  $m \cdot p$  as frequency for  $a!$  and  $m \cdot (1 - p)$  as frequency for  $b!$ . The empirical  $\chi^2$  score therefore calculates as

$$\chi^2 = \frac{8 - 14 \cdot p}{14 \cdot p} + \frac{6 - 14 \cdot (1 - p)}{14 \cdot (1 - p)}.$$

An easy calculation yields  $\chi^2 = 0$  for  $p = 8/14$ . This is obviously smaller than the value  $\chi_{crit}^2 = \chi_{0.1,1}^2 = 2.706$ . We proceed to the second step; confidence interval estimation.

Let  $t_1 = 0.03, \dots, t_8 = 2.69$  be the data associated to  $\lambda_1$  and  $t'_1 = 2.28, \dots, t'_6 = 19.01$  be the data associated to  $\lambda_2$ . Calculating the confidence intervals according

to equation (5.3) yields

$$C_1 = [0.441, 1.458] \text{ and } C_2 = [0.092, 0.368].$$

We see that  $\lambda_1 \in C_1$  and  $\lambda_2 \in C_2$  and are therefore willing to accept that the recorded sample was drawn from the prescribed parameters.

Note that these two subsequent steps do not yet make a sound statement about the acceptance of the hypothesis  $O \in \text{OutObs}(\mathcal{S}, 0.05, 1, 14)$ , since we test multiple hypothesis at once. We need to adjust the individual level of significance for the statistical tests, to conclude the overall acceptance with  $\alpha = 0.1$ . This is known as the inflation of the error of first kind, and discussed subsequently.

Example 5.32 highlights the necessity of two additional assumptions we need to make, if we are to apply confidence intervals as the method of choice:

- The model illustrates the importance to uniquely identify every recorded trace. Assume for the sake of the illustration that  $a! = b!$ . It is not directly possible to associate values  $t_i$  with  $\lambda_1$  and  $t'_i$  with  $\lambda_2$ . Without an additional assumption, we were left to check all possible permutations of  $t_i$  and  $t'_i$  for the induced confidence intervals. The increase of computational complexity is immediately visible, and therefore we assume all specification models to be internally deterministic for this method to be applicable. That is, we assume there to be a bijection between paths and traces.
- The sum of two exponential distribution is not an exponential distribution. Hence, confidence interval estimation may be flawed for two sequential Markovian actions. It is known that the sum of exponential distributions is a *phase type distribution*; a distribution known to be dense in the set of all positively valued distributions. It is easy to imagine a non-exponential distribution with the same mean. To avoid calculating confidence intervals for variance, skewness etc. we assume models to not enable two consecutive Markovian actions (possibly intercepted by an internal  $\tau$  action).

**Multiple comparisons problem.** Since the  $\chi^2$  test and all subsequent confidence interval estimations are statistical hypothesis tests on their own, we face the issue of error accumulation. To illustrate: if a statistical hypothesis test is performed at  $\alpha = 0.05$  there is a 5% chance of performing an error of first kind. That is the erroneous rejection of a true hypothesis. If we are to apply 100 individual tests with  $\alpha = 0.05$  we expect to perform this error 5 times. If we assume the tests to be independent of each other, the probability of committing at least one error of first kind calculates as  $1 - (1 - 0.05)^{100} = 99.4\%$ .

There are several techniques to cope with the inflation of the error of first kind. We refer to Appendix A.2 for a more detailed discussion on possibly applicable methods. For the remainder of this section, we use the straightforward Bonferroni correction, i.e.

$$\alpha_{local} = \frac{\alpha_{global}}{l}$$

where  $l$  is the total number of statistical hypothesis tests to be performed.

**Example 5.33.** We return to Example 5.32. We established that performing three hypothesis tests with  $\alpha = 0.1$  does not have an overall type I error of  $\alpha = 0.1$ . Applying Bonferroni correction for a total of three hypothesis tests yields  $\alpha_{local} \approx 0.033$ . These entail the  $\chi^2$  test and two interval estimations. The  $\chi^2$  test still passes, and the new confidence intervals are

$$C'_1 = [0.353, 1.677] \text{ and } C'_2 = [0.070, 0.432].$$

Naturally  $\lambda_1 \in C'_1$  and  $\lambda_2 \in C'_2$  still hold, and we give the implementation the probabilistic pass verdict.

### 5.3.2 Stochastic Delay and Quiescence

Testing needs to assess if an implementation is allowed to be unresponsive when output was expected [166]. In our formalism, quiescence  $\delta$  models the absence of outputs for an *indefinite* time. It should be regarded with caution in practice. A common way to deal with quiescence is a *global* fixed time-out value set by a user [29, 15]. The time progress in an IOMA is governed by exponential distributions, hence a global time-out has two disadvantages: First, a time-out might occur before a specified Markovian transition is taken. The average waiting time of this event might be substantially higher than the global time-out. Second, a global time-out might unnecessarily prolong the overall test process.

A time-out can be seen as a delay that follows a Dirac distribution, e.g. a *deterministic* time-out after  $t$  time units. This is incompatible with IOMA: Dirac delays cannot be represented in IOMA, and consequently they were not considered in the statistical evaluation that we developed in the previous section.

We now detail a possible solution for IOMA that avoids the problem of Dirac distributions, and aims at minimising the probability of erroneously declaring quiescence, while keeping the overall testing time as low as possible. In order to avoid Dirac distributions, an MBT tool for IOMA needs to implement quiescence by racing an exponentially distributed delay with rate  $\lambda_\delta$  against the implementation; this *quiescence timer* winning the race is then treated as the quiescence output  $\delta$ . Let  $\lambda > 0$  be the minimum exit rate over all Markovian states. With level of significance  $\alpha \in (0, 1)$ , we would like the probability that the quiescence timer expires before a Markovian transition is executed, i.e. that we incorrectly report quiescence when the implementation could make progress, to be at most  $\alpha$ . Choosing  $\mu_\delta = \lambda \cdot \frac{\alpha}{1-\alpha}$  as the quiescence timer's rate achieves this probability with the shortest waiting time in case of actual quiescence. We can further reduce the waiting time by using a different rate in every state: if the exit rate of state  $s$  is  $\lambda_s$ , we use rate  $\mu_\delta^s = \lambda_s \cdot \frac{\alpha}{1-\alpha}$  to judge quiescence in  $s$ .

The statistical evaluation only has to be adjusted to consider the new exit rate  $\lambda + \mu_\delta$  and the newly added “Markovian transition” for quiescence. In fact, we can directly represent this approach by rewriting the specification model as shown in Example 5.34 below. For non-Markovian states a *default* maximal waiting time is still applicable.

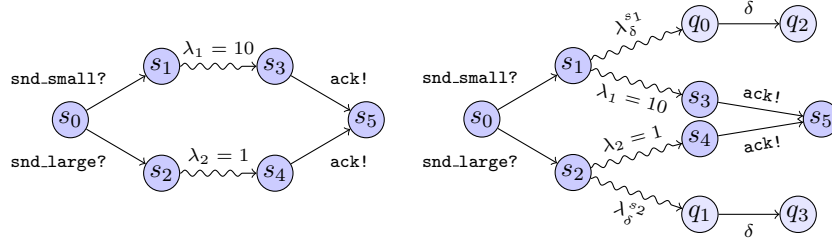


Figure 5.9: Two example specifications for quiescence timeouts

**Example 5.34.** The left hand side of Figure 5.9 shows a simple specification of a file transmission protocol. Exponential distributions model the delay between sending a file and acknowledging its reception. Different delays are associated with sending small or large files, respectively. After a file was sent, there is a chance that it gets lost, and we do not receive an acknowledgement. In this case the system is judged as quiescent, and therefore erroneous. However, since  $\lambda_2 \ll \lambda_1$ , a test should use a quiescence timer rate of  $\lambda_\delta^{s_1} = 10 \cdot \frac{\alpha}{1-\alpha}$  in  $s_1$ , and  $\lambda_\delta^{s_2} = \frac{\alpha}{1-\alpha}$  in  $s_2$  to minimise the probability to erroneously judge quiescence, while also keeping the global testing time as low as possible. Regardless, for sufficiently large sample size, an MBT tool eventually erroneously observes quiescence. Rather than assigning the functional fail verdict right away, the right hand side of Figure 5.9 therefore allows some amount of quiescence observations depending on  $\alpha$ , i.e. how many erroneous quiescent judgements we are willing to accept.

We compare a global quiescence timer rate to individual ones by assuming  $\alpha = 0.05$ , and by assuming we are to test the protocol as illustrated on the left hand side in Figure 5.9 100 times:

**Long global:** A sensible long global quiescence timer rate is  $\lambda_\delta^{s_2} \approx 0.053$ . Executing 100 test cases yields a worst case expected waiting time for the case where the implementation is always quiescent of  $100/\lambda_\delta^{s_2} = 1900$  time units. However, we are (more than) guaranteed to incorrectly judge the implementation quiescence in at most 5% of all cases.

**Short global:** A sensible short global quiescence timer rate is  $\lambda_\delta^{s_1} \approx 0.526$ . The worst-case expected time is only 190 time units. However, the probability of the transition with rate  $\lambda_2$  not firing before the quiescence timer becomes  $\approx 34\%$ . We would then incorrectly judge the implementation quiescent even though the Markovian transition might still take place.

**Individual:** Using the long rate in state  $s_2$  and the short one in state  $s_1$  guarantees that we erroneously judge quiescence overall in 5% of the cases. Note that this is accounted for in the specification on the right hand side in Figure 5.9. The worst case waiting time now depends on the probability  $p$  of sending a small file instead of a large one; it is  $p \cdot 190 + (1-p) \cdot 1900$ . In general, time is saved in the overall test process whenever a small file is sent.

### 5.3.3 Markovian Test Algorithm Outline

We summarize all necessary steps to perform model-based testing with Markov automata using our framework:

1. Generate a test case (suite resp.) for the specification IOMA
2. Execute the test case (all test cases of the test suite resp.)  $m$  times. If the functional fail verdict is encountered in any of the  $m$  executions, then fail the implementation for *functional* reasons.
3. Calculate the number of necessary statistical hypothesis tests for each test case. Apply  $\alpha$  correction accordingly.
4. Perform statistical analysis on the gathered sample of size  $m$  for the test case with the new parameter  $\bar{\alpha}$  (all test cases of the test suite resp.).
  - (a) Use optimisation or constraint solving to find a scheduler such that  $\chi^2 \leq \chi_{crit}^2$ . If no such scheduler is found, reject the implementation for *probabilistic* reasons.
  - (b) Perform confidence interval estimation, and check if all Markovian parameters are contained in their respective intervals. If there is at least one parameter, which is not contained in its confidence interval, reject the implementation for *probabilistic* reasons.
5. If no *fail* was encountered during the process, accept the implementation.

We point out that item 4.b is generally only applicable if we assume that the specification model 1. is internally deterministic, and 2. only contains paths, where a Markovian action is eventually preceded by an externally visible action before traversing another Markovian transition. These are necessary assumptions for the interval estimation to work, as pointed out in this section.

## 5.4 Experiments on the Bluetooth Device Discovery Protocol

Bluetooth is a wireless communication technology standard [161] specifically aimed at low-powered devices that communicate over short distances. To cope with inference, the protocol uses a frequency hopping scheme in its initialisation period. Before any communication can take place, Bluetooth devices organise themselves into small networks called *piconets* consisting of one *master* and up to seven *slave* devices.

To illustrate our framework, we study the discovery phase for one master and one slave device. The protocol is inherently stochastic due to the initially random and unsynchronised state of the devices. We give a high level overview of the protocol in this case. The reader is referred to a case study performed with PRISM [64] for a detailed description and for formal analysis on the protocol in a more general setting.

We point out, that the protocol specification prescribes a time delay, which is not exponentially distributed. Therefore, we have to approximate the true distribution via parameter estimation as our theory intends to.

**Device Discovery Protocol.** To resolve possible interference, the master and slave device communicate via a prescribed sequence of 32 frequencies. Both devices have a 28-bit clock that ticks every  $312.5\mu s$ . The master device broadcasts on two frequencies for two consecutive ticks, followed by a two-tick listening period on the same frequencies. It picks the broadcasting frequency according to the formula:

$$freq = [CLK_{16-12} + off + (CLK_{4-2,0} - CLK_{16-12}) \bmod 16] \bmod 32,$$

where  $CLK_{i-j}$  marks the bits  $i$  to  $j$  of the clock and  $off \in \mathbb{N}$  is an offset. The master device chooses one of two *tracks* and switches to the respective other every  $2.56s$ . Moreover, every  $1.28s$ , i.e. every time the 12th bit of the clock changes, a frequency is *swapped* between the two tracks. For simplicity, we chose  $off = 1$  for track one and  $off = 17$  for track two, such that the two tracks initially comprise frequencies  $1, \dots, 16$  and  $17, \dots, 32$ .

Conversely, the slave device periodically scans on the 32 frequencies and is either in a sleeping or listening state. To ensure the eventual connection, the hopping rate of the slave device is much slower. Every  $0.64s$  it listens to one frequency in a window of  $11.25ms$  and is in a sleeping state during the remaining time. It cycles to the next frequency after  $1.28s$ . This is enough for the master device to broadcast on 16 different frequencies.

**Parameter Estimation.** Note that the time to connect two devices is deterministic for any initial state. That is, assuming we know the initial state of both devices, we can calculate the time needed until a connection is established. We assume that the clocks of both devices are desynchronized to avoid this trivial scenario, i.e. the master sends out packages, while the slave starts listening after a uniformly chosen random waiting time. Naturally, this obfuscates the initial states, and prevents an easy calculation of the connection time.

However, we are left with four scenarios that enable synchronisation:

- Synchronisation happens during the first 16 broadcasted frequencies. This happens between  $0s$  and  $1.28s$  and comprises 16 frequencies.
- Synchronisation happens after the first frequency swap of the master device. This happens between  $1.28s$  and  $2.56s$  and comprises one frequency.
- Synchronisation happens after the first switch of tracks and two frequency swaps of the master device. This happens between  $2.56s$  and  $3.84s$  and comprises 14 frequencies.
- Synchronisation happens after the first switch of tracks and three frequency swaps of the master device. This happens between  $3.84s$  and  $5.12s$  and comprises one frequency.



Note that these four scenarios are exhaustive, i.e. the master device broadcasts frequencies, such that the slave device necessarily must listen to at least one coinciding frequency within 5.12s according to the specification. This also causes the probability jumps notable in Figure 5.11b.

The different scenarios yield 32 possible exact waiting times to connect, i.e. after 2 or 3 ticks, 6 or 7 ticks, etc. Calculating the mean of all waiting times, gives us the average waiting time as approximately 1.325s. Taking the reciprocal finally yields the parameter  $\lambda = 0.755$  as the estimated parameter for the exponential delay, i.e.  $\frac{1}{\lambda} = 1.325$  is the average time we expect to wait before connection is established. We use this exponential distribution to approximate the true distribution.

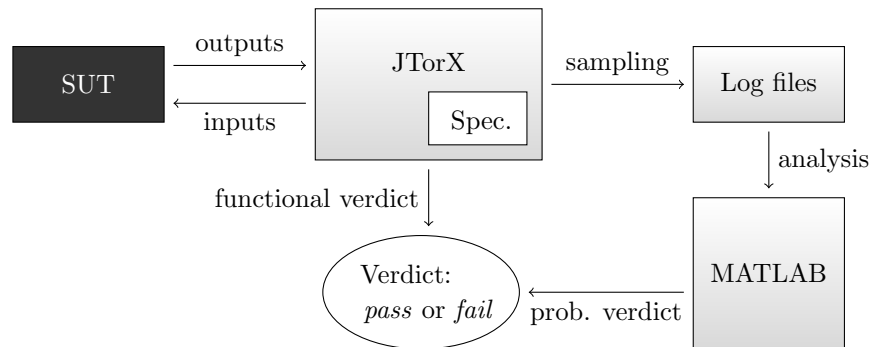


Figure 5.10: Experimental set up entailing the system under test, the MBT tool JTorX [15] and MATLAB [86]. Logs are gathered during the conformance test, and analysed later for a probabilistic verdict.

**Experimental Setup.** Our tool-chain is depicted in Figure 5.10. Although it is the same as for the case studies performed in Chapter 4, we point out, that MATLAB additionally calculates the acceptance regions for the timed parameters. The implementation is tested on-the-fly via the MBT tool JTorX [15], which generates tests with respect to a transition system abstraction of the specification (Figure 5.11a). JTorX returns the functional *fail* verdict, if unforeseen output is observed at any time throughout the test process. Additionally, we chose a *global* time-out of approximately 5.2s in accordance with the specification, i.e. the time that the master device needs to broadcast *all* available frequencies at least once. The recorded log files of JTorX comprise the sample.

We implemented the protocol and three mutants in Java 7;

$\mathcal{M}_1$  The *master mutant*  $\mathcal{M}_1$  never switches between tracks one and two, therefore covering fewer different frequencies than the correct protocol in the same time. An easy calculation yields, that this mutant needs a total of  $16 \times 1.28s = 20.48s$  to cover all 32 frequencies. Hence, we expect a much *lower* probability to connect in the same time when compared to the correct counterpart.

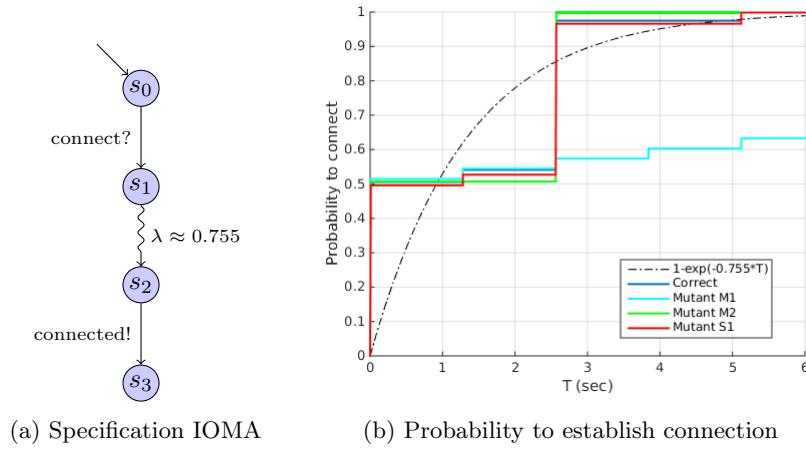


Figure 5.11: High level specification of the Bluetooth device discovery protocol for one master and one slave device. The time to establish a connection for a correct implementation and two mutants is compared to the assumed underlying exponential distribution with parameter  $\lambda \approx 0.755$ .

$\mathcal{M}_2$  The *master mutant*  $\mathcal{M}_2$  never swaps frequencies, and only switches between the two tracks. Therefore, it covers all available frequencies in at most  $3.84s$ . The expected probability to connect is *higher* in the same time when compared to the correct counterpart.

$\mathcal{S}_1$  The *slave mutant* has its listening period halved, and thus only listens for  $5.65ms$  every  $1.28s$ . Therefore it has a longer sleeping period, and we expect that the probability to connect in the same time as the correct implementation is slightly reduced.

Figure 5.11a shows the high level specification of the protocol. The request for both devices to synchronise is either followed by an acknowledgement or a time-out. A collected sample therefore consisted of the traces

$$\begin{aligned}\sigma_1 &= 0 \text{ connect? } t \text{ connected!}, \\ \sigma_2 &= 0 \text{ connect? } t \delta.\end{aligned}$$

**Results.** We collected samples of varying sizes to calculate the confidence intervals of the delay parameter  $\lambda = 0.755$  for  $\alpha = 0.05$ . Figure 5.11b shows the cumulative probability distribution to connect within  $T$  seconds of the assumed underlying exponential distribution  $1 - e^{-0.755 \cdot T}$ , alongside empiric distribution functions of sample data collected for 1000 executions of the correct implementation and the three mutants, respectively.

Table 5.1 shows the resulting calculated confidence intervals for  $\alpha = 0.05$  for sample sizes  $m = 100$ ,  $m = 1000$ , and  $m = 10000$ , respectively. Note that

	correct	mutants		
	$\mathcal{M} \parallel \mathcal{S}$	$\mathcal{M}_1 \parallel \mathcal{S}$	$\mathcal{M}_2 \parallel \mathcal{S}$	$\mathcal{M} \parallel \mathcal{S}_1$
$k = 2$	Accept	Reject	Accept	Accept
$m = 100$	[0.586, 0.868]	—	[0.597, 0.885]	[0.673, 0.997]
Timeouts	0	33	0	0
$k = 2$	Accept	Reject	Reject	Reject
$m = 1000$	[0.729, 0.826]	—	[0.767, 0.868]	[0.756, 0.855]
Timeouts	0	376	0	0
$k = 2$	Accept	Reject	Reject	Reject
$m = 10000$	[0.735, 0.764]	—	[0.772, 0.803]	[0.757, 0.787]
Timeouts	0	3753	0	0

Table 5.1: Verdicts and interval estimations for the Bluetooth initialisation. An implementation is accepted if  $\lambda \approx 0.755 \in C$ , where  $C$  is the confidence interval for  $\alpha = 0.05$  of the true mean of the assumed underlying exponential distribution.

no  $\alpha$  correction was necessary, as the confidence intervals were the only tested statistical hypotheses on the sample data. An implementation was accepted, if the parameter  $\lambda = 0.755$  was contained in the calculated confidence intervals. The confidence intervals were calculated according to Equation (5.3).

The correct implementation  $\mathcal{M} \parallel \mathcal{S}$  was accepted for all three sample sizes. Master mutant one  $\mathcal{M}_1 \parallel \mathcal{S}$  was rejected for all cases due to the functional fail verdict, as multiple time-outs were observed. Conversely, master mutant two  $\mathcal{M}_2 \parallel \mathcal{S}$  and the slave mutant  $\mathcal{M} \parallel \mathcal{S}_1$  were accepted for  $m = 100$ , because the parameter  $\lambda$  was contained in the calculated confidence interval. However, we see that both were rejected for larger sample sizes, as the confidence intervals grew smaller to the point of not containing  $\lambda$  any longer.

**Discussion.** The performed case study was not tailored towards MBT with Markov automata. The waiting time of interest is clearly not exponentially distributed, and only means of the delay until connection happens are compared. Evidently, the underlying distribution has an entirely different shape, when compared to an exponential distribution, while approximately preserving the mean value. Nonetheless, the framework is applicable and rightfully judged the correct implementation as conforming, while eliminating the three mutants.

The confidence interval for the waiting time of the slave mutant, marginally not contained the parameter  $\lambda$ . Sufficiently many repetitions of the experiment should eventually result in a confidence interval that contains  $\lambda$ , and a type II error is performed. However, upon comparing one specific implementation over varying sample sizes, we conjecture that the confidence intervals grow even smaller with a larger sample. This is in line with the decreasing error of second kind for increasing sample size pointed out in Section 5.2. On the contrary, master mutant two was eliminated with a larger margin. This illustrates the potential of Markovian test theory given the right circumstances. The framework is best applied, if the specification prescribes average time delays, as opposed to fully-fledged non-exponential distributions. The latter is treated in Chapter 7, where testing is based on the more powerful stochastic automata. There is a

trade-off in flexibility and complexity when comparing test theory for Markov- and stochastic automata: While the latter allows more involved distributions, the statistical analysis is reduced to a mere estimation of confidence intervals for the former.

## 5.5 Conclusions

We presented a sound and complete MBT framework to test probabilistic systems with stochastic-time delays. The underlying modelling formalism are Markov automata with a separation of its alphabet in inputs and outputs. They limit the use of time delay to exponential distributions, but mark a relevant intermediate step between probabilistic and stochastic automata. The relevance comes with two feats over the more involved stochastic automata: 1. they are intuitive to construct and apply, and 2. they are straightforward to evaluate in the subsequent statistical analysis necessary to our framework.

To that end, we recalled their definition and defined trace distribution semantics based on schedulers. Further, we defined a conformance relation in the **ioco** tradition called **Mar-ioco** pinning down precisely what *conformance* between two Markov automata means. Much like for pIOTSs, we derived the notion of test cases and their annotations based on this conformance relation. The

Physical Ingredients:	Formal Ingredients:
<ul style="list-style-type: none"> <li>• Informal requirements</li> <li>• Black-box implementation</li> <li>• Observations: Definition 5.25, <math>Obs(\mathcal{I} \parallel \hat{t}, \alpha, k, m)</math></li> </ul>	<ul style="list-style-type: none"> <li>• Model: Definition 5.3, IOMA</li> <li>• Conformance: Definition 5.15, <math>\sqsubseteq_{Mar-ioco}</math></li> <li>• Test verdicts: Definition 5.27</li> </ul>
Tooling:	Objectives:
<ul style="list-style-type: none"> <li>• MBT tool: JTorX [15]</li> <li>• Test adapter: Implementable</li> <li>• Test generation method: Random testing &amp; quiescence estimates</li> </ul>	<ul style="list-style-type: none"> <li>• Soundness: Theorem 5.29</li> <li>• Completeness: Theorem 5.30</li> </ul>
Assumptions:	
<ul style="list-style-type: none"> <li>• Every physical implementation has a corresponding IOMA model</li> <li>• The specification is internally deterministic</li> <li>• The specification does not allow consecutive Markovian actions</li> </ul>	

Table 5.2: The MBT ingredients instantiated by the **Mar-ioco** framework.

practical test generation algorithms carried over from pIOTSs, too. Probabilistic correctness is assessed after a sampling process that counts frequencies of traces and compares them to statistical requirements. As an addition, we check if the observed time stamps correspond to the prescribed Markovian parameters, which are mean values of an exponential distribution. The use of multiple statistical hypothesis tests in tandem necessitates the use of methods to correct the level of significance  $\alpha$ . We had to make additional assumptions in order to end up with a practically functioning algorithmic outline. That is, in order for confidence intervals to work, we assumed that the specification model is internally deterministic. This ensures that timestamps can be uniquely identified, and attributed to their respective Markovian parameters. We instantiated all necessary items, and summarized them in Table 5.2.

To put the framework into practice, we studied the Bluetooth device discovery protocol for two communicating parties: a master device and a slave device. While the case study enables vastly more complex scenarios, we focused on the time it takes for the two devices to connect, checking if it coincides with the prescribed average time. To test the generosity of the framework, we compared the correct implementation to three mutants: All mutants were consequently eliminated, and the correct implementation was accepted.

## 5.6 Proofs

We present the proofs of the theorems within this chapter. Reoccurring theorems are numbered according to their occurrence in the chapter.

**Theorem 5.17.** *Let  $\mathcal{I}$  and  $\mathcal{S}$  be two pIOTSs and  $\mathcal{I}$  be input enabled, then*

$$\mathcal{I} \sqsubseteq_{Mar-ioco} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{pioco} \mathcal{S}.$$

*Proof.*  $\mathcal{I}$  and  $\mathcal{S}$  have  $\rightsquigarrow_{\mathcal{I}} = \rightsquigarrow_{\mathcal{S}} = \emptyset$ , and are pIOTSs in the sense of Definition 4.1. This means that every trace distribution in  $trd(\mathcal{I})$  and  $trd(\mathcal{S})$  in the sense of Definition 5.14 is a trace distribution in the sense of Definition 4.11. Note that the reverse always holds. The proof is immediate, if we consider that schedulers according to Definition 5.8 require to schedule probability mass 1, i.e. no mass remains for Markovian transitions.  $\square$

**Theorem 5.19.** *Let  $\mathcal{A}, \mathcal{B}$  and  $\mathcal{C}$  be IOMAs and let  $\mathcal{A}$  and  $\mathcal{B}$  be input-enabled, then*

- (i)  $\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{B}$  if and only if  $\mathcal{A} \sqsubseteq_{TD}^{fin} \mathcal{B}$ .
- (ii)  $\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{Mar-ioco} \mathcal{C}$  imply  $\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{C}$ .

*Proof.*  $\boxed{(i)}$   $\implies$  Assume  $\mathcal{A} \sqsubseteq_{Mar-ioco} \mathcal{B}$ . We need to show  $\mathcal{A} \sqsubseteq_{TD}^{fin} \mathcal{B}$ , i.e. that  $\mathcal{D} \in trd(\mathcal{A}, n)$  implies  $\mathcal{D} \in trd(\mathcal{B}, n)$  for all  $n \in \mathbb{N}$ .

Let  $n \in \mathbb{N}$  and  $\mathcal{D}^* \in \text{trd}(\mathcal{A}, n)$ . We prove the statement for every trace distribution prefix smaller or equal to  $n$  via induction: Assume  $\mathcal{D} \in \text{trd}(\mathcal{A}, 0)$ . Obviously  $\mathcal{D} \sqsubseteq_0 \mathcal{D}^*$ , and consequently  $\mathcal{D} \in \text{trd}(\mathcal{B}, 0)$ .

Now assume that the above statement has been shown for  $m$  with  $m = k-1 < n$ . We proceed by showing it holds for  $m = k$ . Let  $\mathcal{D} \in \text{trd}(\mathcal{A}, k)$  with  $\mathcal{D} \sqsubseteq_k \mathcal{D}^*$ . Then take  $\mathcal{D}' \in \text{trd}(\mathcal{A}, k-1)$  with  $\mathcal{D}' \sqsubseteq_{k-1} \mathcal{D}$ . By induction assumption we know  $\mathcal{D}' \in \text{trd}(\mathcal{B}, k-1)$ . With the initial assumption, i.e.  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$ , we know in particular that

$$\text{outcont}_{\mathcal{A}}(\mathcal{D}') \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}').$$

Therefore, we choose  $\mathcal{D}'' \in \text{outcont}_{\mathcal{A}}(\mathcal{D}')$ , such that

$$\forall \sigma \in [\mathbb{R}_0^+ \text{Act}]^{k-1} \mathbb{R}_0^+ \text{Act}_O : P_{\mathcal{D}}(\Sigma) = P_{\mathcal{D}''}(\Sigma), \quad (5.4)$$

i.e.  $\mathcal{D}$  and  $\mathcal{D}''$  assign the same probability to abstract traces of length  $k$  ending in output. Note that  $\mathcal{D}'' \in \text{outcont}_{\mathcal{B}}(\mathcal{D}')$  and thus  $\mathcal{D}'' \in \text{trd}(\mathcal{B}, k)$ .

We are left to show that there is a trace distribution, say  $\mathcal{D}'''$ , that assigns

$$\forall \sigma \in \text{Act}^{k-1} \text{Act}_I : P_{\mathcal{D}}(\Sigma) = P_{\mathcal{D}'''}(\Sigma),$$

in addition to (5.4), i.e.  $\mathcal{D}'''$  assigns the same probability to *all* abstract traces of length  $k$ . However, the existence of  $\mathcal{D}'''$  is straightforward, since  $\mathcal{A}$  and  $\mathcal{B}$  are input-enabled. That is, all inputs are enabled in every state of both  $\mathcal{A}$  and  $\mathcal{B}$ . We conclude  $\text{trd}(\mathcal{A}, m) \subseteq \text{trd}(\mathcal{B}, m)$  for all  $m \leq n$ , and with it consequently  $\text{trd}(\mathcal{A}, n) \subseteq \text{trd}(\mathcal{B}, n)$ . Hence  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ .

$\boxed{\Leftarrow}$  Let  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ . We need to show  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$ , i.e. for all  $n \in \mathbb{N}$  and all  $\mathcal{D}^* \in \text{trd}(\mathcal{B}, n)$ , we have  $\text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . Let  $n \in \mathbb{N}$  and  $\mathcal{D}^* \in \text{trd}(\mathcal{B}, n)$ . Choose  $\mathcal{D} \in \text{outcont}_{\mathcal{A}}(\mathcal{D}^*)$ , then we need to show  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ .

By definition of the set *outcont*, we know that  $\mathcal{D} \in \text{trd}(\mathcal{A}, n+1)$ . Together with the initial assumption, i.e.  $\mathcal{A} \sqsubseteq_{TD}^{\text{fin}} \mathcal{B}$ , we conclude  $\mathcal{D} \in \text{trd}(\mathcal{B}, n+1)$ . We are left to show that  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . However, this is straightforward, as for all traces  $\sigma \in [\mathbb{R}_0^+ \text{Act}]^n \mathbb{R}_0^+ \text{Act}_I$  it holds that  $P_{\mathcal{D}}(\Sigma) = 0$  by construction of *outcont*. Consequently  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ , and therefore  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$ .

$\boxed{(ii)}$  We need to show that  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{B}$  and  $\mathcal{B} \sqsubseteq_{\text{Mar-ioco}} \mathcal{C}$  imply  $\mathcal{A} \sqsubseteq_{\text{Mar-ioco}} \mathcal{C}$ . By initial assumptions we know

1.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{B}, n) : \text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ , and
2.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{C}, n) : \text{outcont}_{\mathcal{B}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .

We need to show

3.  $\forall n \in \mathbb{N} \forall \mathcal{D}^* \in \text{trd}(\mathcal{C}, n) : \text{outcont}_{\mathcal{A}}(\mathcal{D}^*) \subseteq \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .

Let  $n \in \mathbb{N}$ , choose  $\mathcal{D}^* \in \text{trd}(\mathcal{C}, n)$ , and assume  $\mathcal{D} \in \text{outcont}_{\mathcal{A}}(\mathcal{D}^*)$ . Obviously,  $\mathcal{D} \in \text{trd}(\mathcal{A}, n+1)$ . Together with (i), and since  $\mathcal{A}$  and  $\mathcal{B}$  are input enabled by assumption, we know  $\mathcal{D} \in \text{trd}(\mathcal{B}, n+1)$ . Note that for all traces  $\sigma \in [\mathbb{R}_0^+ \text{Act}]^n \mathbb{R}_0^+ \text{Act}_I$  we have  $P_{\mathcal{D}}(\Sigma) = 0$ . With item 1., we know  $\mathcal{D} \in \text{outcont}_{\mathcal{B}}(\mathcal{D}^*)$ . Together with the assumption  $\mathcal{B} \sqsubseteq_{\text{Mar-ioco}} \mathcal{C}$ , we conclude  $\mathcal{D} \in \text{outcont}_{\mathcal{C}}(\mathcal{D}^*)$ .  $\square$

**Theorem 5.29** *Each annotated test for an IOMA  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{\text{Mar-ioco}}$ .*

*Proof.* Let  $\mathcal{I}$  be an input enabled IOMA and  $\hat{t}$  be a test for  $\mathcal{S}$ . Further assume that  $\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ . Then we want to show  $V(\mathcal{I}, \hat{t}) = \text{pass}$ , i.e. we show that a **Mar-ioco** correct implementation passes an annotated test case. By the definition of verdicts (Definition 5.27) we have  $V(\mathcal{I}, \hat{t}) = \text{pass}$  if and only if

$$v_{\text{func}}(\mathcal{I}, \hat{t}) = v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}.$$

We proceed by showing that  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ , and  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  in two separate steps:

1. In order for  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ , we need to show that

$$\text{ann}_{\text{Mar-ioco}}^{\mathcal{S}}(\sigma) = \text{pass} \text{ for all } \sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t}),$$

according to the definition of verdicts (Definition 5.27). Therefore, let  $\sigma \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t})$ . We need to show  $\text{ann}_{\text{Mar-ioco}}^{\mathcal{S}}(\sigma) = \text{pass}$  by definition of annotations (Definition 5.22). Assume  $\sigma' \in \text{traces}^{\text{fin}}(\mathcal{S})$  and  $a! \in \text{Act}_O$  such that  $\sigma' \uparrow a! \sqsubseteq \sigma$  for some  $\uparrow \in \mathbb{R}_0^+$ .

We observe two things:

- Since  $\varepsilon \in \text{traces}^{\text{fin}}(\mathcal{S})$ , i.e. the empty trace is a trace and is in  $\text{traces}^{\text{fin}}(\mathcal{S})$ ,  $\sigma'$  always exists.
- If no such  $a! \in \text{Act}_O$  exists, then  $\sigma$  is a trace solely consisting of inputs. By definition of annotations (Definition 5.22) consequently  $\text{ann}_{\text{Mar-ioco}}^{\mathcal{S}}(\sigma) = \text{pass}$ .

By construction of  $\sigma$  we have  $\sigma' \uparrow a! \in \text{traces}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$  and therefore also  $\sigma' \uparrow a! \in \text{traces}^{\text{fin}}(\mathcal{I})$ . We conclude,  $\sigma' \in \text{traces}^{\text{fin}}(\mathcal{I}) \cap \text{traces}^{\text{fin}}(\mathcal{S})$ . Our goal is to show  $\sigma' \uparrow a! \in \text{traces}^{\text{fin}}(\mathcal{S})$ .

Let  $l = |\sigma'|$  be the length of  $\sigma'$ . Without loss of generality, we can now choose  $\mathcal{D} \in \text{trd}(\mathcal{S}, l)$ , such that  $P_{\mathcal{D}}(\Sigma') > 0$ , where  $\Sigma'$  is the corresponding abstract trace to  $\sigma'$ . Note that this, together with the previous observation, yields that  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \neq \emptyset$ . Again, without loss of generality, we choose  $\mathcal{D}' \in \text{outcont}_{\mathcal{I}}(\mathcal{D})$ , such that  $P_{\mathcal{D}'}(\Sigma' [0, \uparrow] a!) > 0$ .

Lastly, we assumed  $\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ , hence  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D})$ . We conclude  $\mathcal{D}' \in \text{trd}(\mathcal{S}, l + 1)$ , and  $P_{\mathcal{D}'}(\Sigma' [0, \uparrow] a!) > 0$ . By construction of trace distributions (Definition 5.14), this implies that  $\sigma' \uparrow a! \in \text{traces}^{\text{fin}}(\mathcal{S})$ . If additionally  $\sigma' \uparrow a \in \text{traces}^{\text{com}}(\mathcal{I} \parallel \hat{t})$ , then  $\sigma' \uparrow a = \sigma$ . Hence  $\text{ann}_{\text{Mar-ioco}}^{\mathcal{S}}(\sigma) = \text{pass}$ , which ultimately yields  $v_{\text{func}}(\mathcal{I}, \hat{t}) = \text{pass}$ .

2. In order for  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  we need to show that

$$\forall \mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k) \exists \mathcal{D}' \in \text{trd}(\mathcal{S}, k) : P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha,$$

according to the definition of verdicts (Definition 5.27). Therefore, let  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$ . By definition of output-observations (Remark 5.26), we have

$$\text{OutObs}(\mathcal{D}, \alpha, k, m) = \{O \in ([\mathbb{R}_0^+ \text{Act}]^{\leq k-1} \mathbb{R}_0^+ \text{Act}_O)^m \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha\}.$$

There exists  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$  with

$$P_{\mathcal{D}'}(\Sigma) = \begin{cases} 0 & \text{if } \sigma \in [\mathbb{R}_0^+ \text{Act}]^{k-1} \mathbb{R}_0^+ \text{Act}_I \\ P_{\mathcal{D}}(\Sigma) & \text{if } \sigma \in [\mathbb{R}_0^+ \text{Act}]^{\leq k-1} \mathbb{R}_0^+ \text{Act}_O. \end{cases} \quad (5.5)$$

To see why, consider the scheduler that assigns all probability to halting instead of inputs for traces of length  $k$ , while assigning the same probability to outputs as the scheduler of  $\mathcal{D}$ . This scheduler exists by definition (Definition 5.8). By construction of the set  $\text{OutObs}$  (Remark 5.26), observe that

$$\begin{aligned} P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

since only traces ending in output are measured.

It is now sufficient to show that  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k)$ . However, as an intermediate step, we first show that  $\mathcal{D}' \in \text{trd}(\mathcal{I}, k)$ , as this will let us make use of the assumption  $\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ .

Consider the mapping  $f$  from the finite paths of  $\mathcal{I} \parallel \hat{t}$  to the finite paths of  $\mathcal{I}$ , i.e.  $f : \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t}) \rightarrow \text{paths}^{\text{fin}}(\mathcal{I})$ , where for every fragment of the path we have

$$f(\dots (s, q) \mu \alpha (s', q') \dots) = \dots s \mu \alpha s' \dots$$

Since tests are internally deterministic IOTSSs, it holds that  $\mu((s, q), \alpha) = \nu(s, \alpha)$ . Note that this holds for  $\alpha \in \text{Act}$ , as well as  $\alpha \in \mathbb{R}_0^+$ , where  $\nu = \mathbb{P}_s$ . In the latter case, and whenever  $a = \tau$ , it is  $q = q'$ , since test cases do not contain Markovian actions (Definition 5.20). That is, the parallel composition of an implementation with a test case does not change the discrete (or Markovian) probability distributions, and probabilities directly transfer. It is then easy to see that  $f$  is an injective mapping, i.e.  $f(\pi_1) = f(\pi_2) \Rightarrow \pi_1 = \pi_2$ .

By definition of trace distributions (Definition 5.14), there is a scheduler, say  $\mathcal{A}' \in \text{Sched}(\mathcal{I} \parallel \hat{t}, k)$ , such that  $\text{trd}(\mathcal{A}') = \mathcal{D}'$ . With the help of  $f$ , we construct a scheduler  $\mathcal{A}'' \in \text{Sched}(\mathcal{I})$ , such that for all traces we have  $P_{\text{trd}(\mathcal{A}')}(\Sigma) = P_{\text{trd}(\mathcal{A}'')}(\Sigma)$ , i.e.  $\text{trd}(\mathcal{A}'') = \mathcal{D}'$ .



For every path  $\pi \in \text{paths}^{\text{fin}}(\mathcal{I})$  with  $f^{-1}(\pi) \in \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$  we define  $\mathcal{A}''$  as

$$\mathcal{A}''(\pi)(\nu) \stackrel{\text{def}}{=} \mathcal{A}'(f^{-1}(\pi))(\mu),$$

where  $\mu$  and  $\nu$  are discrete probability distributions. Observe  $|\mathcal{A}''(\pi)| = |\mathcal{A}'(\pi)|$ , hence, Markovian actions also get assigned the same probability under  $\mathcal{A}''$ . Note in particular that  $P_{\mathcal{A}''}(\Pi) = 0$  if  $\Pi \notin \text{AbsPaths}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$ .

The construction of  $\mathcal{A}''$  is straightforward: By definition of test cases (Definition 5.20) we know that  $\mathcal{I} \parallel \hat{t}$  is internally deterministic.  $\mathcal{I}$  and  $\hat{t}$  are defined over the same alphabet, and  $\hat{t}$  does not contain internal- or Markovian actions. Hence, in particular  $\mathcal{I} \parallel \hat{t}$  does not contain interleaving. This means  $\mathcal{A}''$  can copy the behaviour of  $\mathcal{A}'$  in a step-by-step manner. We set  $\mathcal{D}'' = \text{trd}(\mathcal{A}'')$  and conclude  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, k)$ .

Further, we have

$$\begin{aligned} P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}'', \alpha, k, m)) &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) \\ &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

We proceed to show that  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$ .

The proof is by induction over trace distribution length of prefixes of  $\mathcal{D}''$  up to  $k$ . Trivially, if  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, 0)$ , then also  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, 0)$ . Assume this has been shown for length  $n$ . We proceed by showing that the statement holds for  $n + 1 \leq k$ .

Let  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, n + 1)$  and take  $\mathcal{D}''' \sqsubseteq_n \mathcal{D}''$ . By induction assumption  $\mathcal{D}''' \in \text{trd}(\mathcal{S}, n)$ . Together with the assumption  $\mathcal{I} \sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ , we have

$$\text{outcont}_{\mathcal{I}}(\mathcal{D}''') \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}''').$$

Since  $\mathcal{D}'' \in \text{outcont}_{\mathcal{I}}(\mathcal{D}''')$  (Equation (5.5)) we have  $\mathcal{D}'' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}''')$ , and consequently  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, n + 1)$ . We have shown  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$  and conclude  $P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$ . Ultimately, this yields  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{pass}$  by the definition of verdicts (Definition 5.27).

Both parts together give  $V(\mathcal{I}, \hat{t}) = \text{pass}$ . This means that an annotated test for  $\mathcal{S}$  is sound with respect to  $\sqsubseteq_{\text{Mar-ioco}}$  for every  $\alpha \in (0, 1)$ .  $\square$

**Theorem 5.30** *The set of all annotated test cases for an IOMA  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{\text{Mar-ioco}}$  for sufficiently large sample size.*

*Proof.* In order to show the completeness of test suite  $\hat{T}$  consisting of all annotated tests for  $\mathcal{S}$ , assume that  $\mathcal{I} \not\sqsubseteq_{\text{Mar-ioco}} \mathcal{S}$ . Our goal is to show that

$V(\mathcal{I}, \hat{T}) = fail$ . By the definition of verdicts (Definition 5.27) this is the case iff  $v_{func}(\mathcal{I}, \hat{t}) = fail$  or  $v_{prob}(\mathcal{I}, \hat{t}) = fail$  for some  $\hat{t} \in \hat{T}$ .

Since  $\mathcal{I} \not\sqsubseteq_{Mar-ioco} \mathcal{S}$ , there is  $k \in \mathbb{N}$ , such that there is  $\mathcal{D}^* \in trd(\mathcal{S}, k)$ , for which

$$outcont_{\mathcal{I}}(\mathcal{D}^*) \not\sqsubseteq outcont_{\mathcal{S}}(\mathcal{D}^*).$$

More specifically

$$\exists \mathcal{D} \in outcont_{\mathcal{I}}(\mathcal{D}^*) \forall \mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*) \exists \sigma \in \mathcal{C} : P_{\mathcal{D}}(\Sigma) \neq P_{\mathcal{D}'}(\Sigma), \quad (5.6)$$

where  $\mathcal{C} \stackrel{\text{def}}{=} traces^{fin}(\mathcal{I}) \cap [\mathbb{R}_0^+ Act]^k \mathbb{R}_0^+ Act_O$ , and  $\Sigma$  is the corresponding abstract trace to  $\sigma$ . Without loss of generality, we can assume  $k$  to be minimal. There are two cases to consider:

1.  $\exists \sigma \in \mathcal{C} : \sigma \notin traces^{fin}(\mathcal{S})$ , or
2.  $\forall \sigma \in \mathcal{C} : \sigma \in traces^{fin}(\mathcal{S})$ ,

We will relate the two cases to the functional and the statistical verdict (Definition 5.27), respectively. We prove that item 1. implies  $v_{func}(\mathcal{I}, \hat{T}) = fail$ , and item 2. implies  $v_{prob}(\mathcal{I}, \hat{T}) = fail$ . Therefore, let  $\mathcal{D} \in outcont_{\mathcal{I}}(\mathcal{D}^*)$  such that Equation (5.6) holds for all  $\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)$ .

1. In order for  $v_{func}(\mathcal{I}, \hat{t}) = fail$ , we need to show

$$\exists \sigma \in traces^{com}(\mathcal{I} \parallel \hat{t}) : ann_{Mar-ioco}^{\mathcal{S}}(\sigma) = fail$$

for some  $\hat{t} \in \hat{T}$ , according to the definition of verdicts (Definition 5.27). Assume there is  $\sigma \in \mathcal{C}$ , such that  $\sigma \notin traces^{fin}(\mathcal{S})$ . We show that there is  $\hat{t} \in \hat{T}$  for which  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ , and  $ann_{Mar-ioco}^{\mathcal{S}}(\sigma) = fail$ .

Without loss of generality we can assume  $P_{\mathcal{D}}(\Sigma) > 0$ . To see why, assume  $P_{\mathcal{D}}(\Sigma) = 0$ . Then we can find a trace distribution in  $outcont_{\mathcal{S}}(\mathcal{D}^*)$  with an underlying scheduler  $Sched(\mathcal{S}, k+1)$  that does not assign positive probability to the last action in  $\Sigma$  to obtain probability 0. This violates the assumption that  $P_{\mathcal{D}}(\Sigma) \neq P_{\mathcal{D}'}(\Sigma)$ . We conclude  $\sigma = \sigma' \mathfrak{t} a$ , for some  $\sigma' \in [\mathbb{R}_0^+ Act]^k$ ,  $\mathfrak{t} \in \mathbb{R}_0^+$ , and  $a \in Act_O$ .

The prefix  $\sigma'$  is in  $traces^{fin}(\mathcal{S})$ , because it is of length  $k$ , and since  $\mathcal{D}^* \in trd(\mathcal{S}, k)$ . Since  $\mathcal{D}$  and all  $\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)$  are continuations of  $\mathcal{D}^*$ , we conclude  $P_{\mathcal{D}^*}(\Sigma') = P_{\mathcal{D}}(\Sigma') = P_{\mathcal{D}'}(\Sigma')$ , i.e. all trace distributions of the respective sets assign every prefix of  $\sigma$  the same probability by merit of *outcont*. We conclude  $\sigma' \in traces^{fin}(\mathcal{S})$ , but  $\sigma' \mathfrak{t} a \notin traces^{fin}(\mathcal{S})$ .

By initial assumption  $\hat{T}$  contains *all* annotated test cases for  $\mathcal{S}$ . Hence, let  $\hat{t} \in \hat{T}$  such that  $\sigma \in traces^{com}(\hat{t})$ . By the definition of annotations (Definition 5.22) we have  $ann_{Mar-ioco}^{\mathcal{S}}(\sigma) = fail$ . Since  $\sigma \in traces^{fin}(\mathcal{I})$  and  $\sigma \in traces^{com}(\hat{t})$ , we obviously also have  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ . Ultimately, this yields  $v_{func}(\mathcal{I}, \hat{t}) = fail$ .

2. In order for  $v_{prob}(\mathcal{I}, \hat{t}) = fail$ , we need to show

$$\exists \mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, l) \forall \mathcal{D}' \in trd(\mathcal{S}, l) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, l, m)) < 1 - \alpha,$$

for some  $\hat{t} \in \hat{T}$  and  $l \in \mathbb{N}$ , according to the definition of verdicts (Definition 5.27).

Together with Equation (5.6) and the definition of acceptable outcomes (Definition 5.25), we conclude

$$\forall \mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) < \beta_m \quad (5.7)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ . Observe that

$$\begin{aligned} & \sup_{\mathcal{D}' \in trd(\mathcal{S}, k+1)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) \\ &= \sup_{\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)), \end{aligned} \quad (5.8)$$

by definition of  $OutObs$  (Remark 5.26).  $OutObs$  only comprises traces ending in output, thus its measure under any trace distribution of  $trd(\mathcal{S}, k+1)$  cannot be larger than the ones already contained in  $outcont_{\mathcal{S}}(\mathcal{D}^*)$ . Together with Equation (5.7) this yields

$$\forall \mathcal{D}' \in trd(\mathcal{S}, k+1) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) < \beta_m \quad (5.9)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ .

We are left to show that  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, k+1)$  for some  $\hat{t} \in \hat{T}$ . Let  $\mathfrak{K} = \{\sigma \in traces^{fin}(\mathcal{I}) \mid P_{\mathcal{D}}(\Sigma) > 0\}$ , i.e. all traces getting assigned a positive probability under  $\mathcal{D}$ . Obviously  $\mathfrak{C} \subseteq \mathfrak{K}$ . By initial assumption, we know that  $\mathfrak{C} \subseteq traces^{fin}(\mathcal{S})$ , but implicitly also  $\mathfrak{K} \subseteq traces^{fin}(\mathcal{S})$ . That means there is a test case  $\hat{t}$  for  $\mathcal{S}$ , such that all  $\sigma \in \mathfrak{K}$  are in  $traces^{com}(\hat{t})$ . In particular, observe that all  $\sigma$  end in output by assumption. Hence, the last stage of the test case is the second bullet of the definition of test cases (Definition 5.20). We now construct a scheduler  $\mathcal{A}' \in Sched(\mathcal{I} \parallel \hat{t}, k+1)$  such that  $trd(\mathcal{A}') = \mathcal{D}$ .

Consider the mapping  $f : \mathfrak{K} \rightarrow paths^{fin}(\mathcal{I} \parallel \hat{t})$ , where for every path fragment

$$f(\dots s \mu a s' \dots) = \dots (s, q) \nu a (s', q') \dots$$

The state  $q$  is uniquely determined, because test cases are internally deterministic. In particular  $q = q'$  iff  $a = \tau$  or  $a \in \mathbb{R}_0^+$ . In the latter case  $\mu = \nu = \mathbb{P}_s$ . Since every discrete distribution of test cases is the Dirac distribution, it is  $\mu(s, a) = \nu((s, q), a)$ . It is then easy to see that  $f$  is an injection, i.e.  $f(\pi_1) = f(\pi_2) \Rightarrow \pi_1 = \pi_2$ .

We now construct the scheduler  $\mathcal{A}' \in Sched(\mathcal{I} \parallel \hat{t}, k+1)$ , such that  $\mathcal{D} = trd(\mathcal{A}')$ . Let  $\mathcal{A} \in Sched(\mathcal{I}, k+1)$  be the scheduler inducing  $\mathcal{D}$  by definition of trace distributions (Definition 5.14). For every  $\pi \in tr^{-1}(\mathfrak{K})$ , we define

$$\mathcal{A}'(\pi)(\nu) \stackrel{\text{def}}{=} \mathcal{A}(f^{-1}(\pi))(\mu),$$

where  $\mu$  and  $\nu$  are discrete probability distributions as above. Observe  $|\mathcal{A}'(\pi)| = |\mathcal{A}(\pi)|$ , i.e. all Markovian actions get assigned the same probability. The construction of  $\mathcal{A}'$  is straightforward: Since  $\hat{t}$  is internally deterministic, and does neither contain internal- nor Markovian actions, there is no interleaving in  $\mathcal{I} \parallel \hat{t}$ . Since there are no non-Dirac distributions in  $\hat{t}$ , the scheduler of  $\mathcal{I} \parallel \hat{t}$  can simply copy the behaviour of  $\mathcal{A}$  step-by-step. Hence  $P_{trd(\mathcal{A}')}(\Sigma) = P_{\mathcal{D}}(\Sigma)$  for all  $\sigma \in \mathfrak{K}$ .

Together with Equation (5.8), we have found a scheduler  $\mathcal{A}'$  such that  $trd(\mathcal{A}') \in trd(\mathcal{I} \parallel \hat{t}, k+1)$ , and for all  $\mathcal{D}' \in trd(\mathcal{S}, k+1)$  we have

$$P_{\mathcal{D}'}(OutObs(trd(\mathcal{A}'), \alpha, k+1, m)) < \beta_m. \quad (5.10)$$

Now iff  $\alpha \leq 1 - \beta_m$ , we estimate (5.10) further to

$$P_{\mathcal{D}'}(OutObs(trd(\mathcal{A}'), \alpha, k+1, m)) < \beta_m \leq 1 - \alpha.$$

However, the inequality  $\alpha \leq 1 - \beta_m$  always holds for sufficiently large  $m$ , since  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$  by the definition of acceptable outcomes (Definition 5.25). Ultimately, this yields  $v_{prob}(\mathcal{I}, \hat{t}) = fail$ .

Together, the two cases yield  $\mathcal{I} \not\sqsubseteq_{Mar-ioco} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = fail$ .  $\square$

## CHAPTER 6

---

### Stoic Trace Semantics for Markov Automata

---

Equivalence relations are a corner stone in process theory, providing fundamental insight when two systems are *essentially* the same. Also, they are paramount for both the design of complex systems and their analysis by abstraction. Typical and important concepts include [184]: 1. bisimulation 2. testing, and 3. trace-based equivalences or pre-orders. These concepts and their interrelations have been studied in many different settings, including (interactive) Markov chains [93, 10], and probabilistic automata [158]. A coarse and abstract framework for the prominent formalism of Markov automata, relating processes whenever an external observer cannot distinguish them is, however, missing. Instead, the scientific focus has so far been on bisimulation relations.

This chapter develops a trace-based semantics for *open* Markov automata. What sets this framework apart from other, more restricted, approaches is that the underlying schedulers may decide to wait before scheduling the next action. Hence their name - *stoic* schedulers. This property is natural in an open setting, where processes may have to wait before being able to synchronize with their environment, and allows us to relate processes for which other propositions struggle. We characterize the resulting notion of trace distribution equivalence in terms of an intuitive button pushing experiment, thereby relating it to observations an external observer could make.

To that end, we follow a different notion of schedulers compared to the previous chapter. In particular Remark 5.10 discusses this fork in the definition of schedulers. However, we opted to not incorporate the potential to wait previously, to guarantee more accessible statistical analysis in MBT. Since the focal point of this chapter is not model-based testing, we investigate this potential branch of stoic schedulers and their implications nonetheless.

At each step of the computation, a scheduler resolves non-deterministic choices and decides for the system how to proceed. We consider *open* Markov automata that entail the capability to interact with the environment. Working with open MA grants us to supersede the urgency assumption [93], stating that any action has precedence over Markovian transitions. In particular, our schedulers may decide to wait a certain amount of time prior to scheduling actions

— hence their name *stoic*. This imposes an additional stoic race between the scheduler decisions and Markovian actions. The ability to wait before scheduling an action is crucial in open semantics, since in an open context, one MA may need to wait before it can synchronize with its environment. As a result, our stoic relation enjoys several natural properties that existing approaches fail to have. In particular, our trace semantics are coarser than weak bisimulation, a classical property for linear semantics.

Further, two systems are deemed equivalent if an external observer cannot distinguish them. We argue that this setting allows for better analysis, due to its most natural abstraction from superfluous details. We propose an intuitive button pushing experiment for MAs. Each experiment is run a number of times and frequency information about the observed traces is collected. Then, non-equivalent MAs can be distinguished via appropriate trace classes and statistical techniques, and our trace semantics can be characterized in terms of this experiment; This observational characterization provides additional evidence that our trace semantics arise as a natural equivalence for MAs. As such, we relate prominent equivalences cross cutting the earlier mentioned approaches based on bisimulation, testing, and traces.

We summarize the main contributions of this chapter:

- solid definition of stoic schedulers and resulting trace based semantics,
- a testing scenario, relating trace distributions to observations, and
- a comparison to prevalent equivalences known from the literature.

**Related Work.** For processes with non-determinism, trace equivalence is usually based on the notion of schedulers, both in the discrete-time setting of probabilistic automata (PA) [158, 164], as well as for the continuous-time formalism of interactive Markov chains (IMC) [196]. In continuous-time, the scheduler was not assumed [196] to be able to delay, i.e. scheduled actions have to be taken immediately. We add the ability to delay, inspired by previous work on verification of *open* interactive Markov chains [32]. Other works define trace semantics for continuous stochastic systems from a co-algebraic perspective [113].

Research in the context of Markov automata focussed primarily on bisimulation relations. Eisentraut et al. [67] were first to define weak bisimulation for Markov automata. Their definition is a conservative extension of weak bisimulation for both IMCs and PAs. The intricacy of the formalism required to employ bisimulation on distributions over states. This technical tool inspired further work of Deng and Hennessy [59] and Song et al. [66] that refined the original definitions. Distribution bisimulations are actively studied in various related contexts [38, 70, 95]. The algorithmic analysis of MA was studied in [83, 85, 170]. Lastly, similar testing scenarios for probabilistic systems are given in [40, 195].

**Origins of the chapter.** The research underlying this chapter was performed from September 2015 to March 2017 in collaboration with Dennis Guck, Holger Hermanns, Jan Krčál, and Mariëlle Stoelinga.

**Organisation of the chapter.** Section 6.1 shortly recalls Markov automata, and establishes *stoic* trace semantics. We relate the semantics to test observations in a black-box testing scenario in Section 6.2. A hierarchy of equivalences is presented in Section 6.3, covering similar approaches and bisimulation. Lastly, Section 6.4 ends the chapter with concluding remarks.

## 6.1 Markov Automata

We shortly defined Markov automata in Chapter 5, before going to the purely open modelling formalism in input/output Markov automata (IOMA). In this section, we reiterate the definition of basic Markov automata, and define language theoretic concepts in paths and traces, as well as their respective abstract counterparts. Note that IOMA were defined as *input-reactive* and *output-generative*. This necessitates a fresh definition of (abstract) paths and traces, since we have not cleanly defined them for Markov automata yet.

Even though our definition of Markov automata does not separate the action alphabet into inputs and outputs, we consider them to be open, i.e. capable of interacting with their environments. Working with open Markov automata grants us to supersede the urgency assumption [93], stating that any action has precedence over Markovian transition.

This provides a segue to investigate an alternate definition of schedulers, compared to the one seen in Chapter 5. Recall that Remark 5.10 discusses a notion of schedulers capable to *wait* before scheduling the next action. While we chose not to use this notion in Chapter 5 because it increases the complexity of the subsequent statistical analysis in an MBT setting, we utilize it here. A *stoic* scheduler may decide to wait a certain amount of time prior to scheduling actions. This implies an additional race between scheduler and Markovian actions, whereas before only multiple Markovian actions caused a *race condition*.

Since model-based testing is not the focal point of this chapter, there is no need for posteriori statistical analysis. This lets us focus on the theoretical implications that such a framework brings with it.

### 6.1.1 Definition and Notation

We recall the definition of Markov automata. This is a reiteration of Definition 5.1, and only mentioned here for the sake of textual completeness.

**Definition 6.1.** A Markov Automaton (MA) is a tuple  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$ , where

- $S$  is a set of states, with  $s_0 \in S$  as the initial state.
- $Act_\tau$  is a set of actions, containing the distinguished element  $\tau$ .
- $\rightarrow \subseteq S \times Act_\tau \times Distr(S)$  is the countable probabilistic transition relation.
- $\rightsquigarrow \subseteq S \times \mathbb{R}^+ \times S$  is the countable Markovian transition relation.

Even though there is no obvious separation into inputs and outputs in the alphabet  $Act_\tau$ , we consider Markov automata to be open. This enables interaction with their environment, and motivates stoic schedulers.

Note that we worked with input output Markov automata in Chapter 5, that were input-reactive and output-generative. This detail becomes relevant in the set of probabilistic transitions, as the target of a distribution is now a set of states, compared to a set of actions and states like before.

**Remark 6.2.** *We reiterate notations for the sake of completeness. We point out that some notations changed, due to MA not being input-generative and output-reactive.*

- Throughout this chapter we let  $\tau$  represent an internal action. Further, we let  $\mu$  and  $\nu$  be discrete probability distributions, and  $\lambda_i$  be positive real-valued numbers denoting parameters of Markovian transitions.
- We abbreviate  $(s, a, \mu) \in \rightarrow$  with  $s \xrightarrow{a} \mu$ , and for  $(s, \lambda, s') \in \rightsquigarrow$ , we write  $s \xrightarrow{\lambda} s'$ .
- The element  $\lambda$  in  $(s, \lambda, s') \in \rightsquigarrow$  is referred to as Markovian action.
- We use  $\alpha \in Act_\tau \cup \mathbb{R}^+$  to denote Markovian and non-Markovian actions.
- We write  $s \rightarrow a$ , if there are  $\mu \in Distr(S)$ , such that  $s \xrightarrow{a} \mu$ .
- We write  $s \xrightarrow{a}_{\mathcal{M}} \mu$ , etc. to clarify that a transition belongs to the MA  $\mathcal{M}$  if ambiguities arise.
- A state  $s \in S$  is called probabilistic, if there is at least one  $a \in Act_\tau$  such that  $s \rightarrow a$ . In that case, we also say action  $a$  is enabled in  $s$ . The set  $enabled(s)$  comprises all enabled actions in  $s$ .
- A state is called stable, if it enables no internal action  $\tau$ .
- A state  $s \in S$  is called Markovian, if there is at least one  $\lambda \in \mathbb{R}^+$  such that  $s \xrightarrow{\lambda} s'$ . A state can be probabilistic and Markovian.
- The rate to go from a state  $s$  to  $s'$  is the sum of all  $\lambda \in \mathbb{R}^+$ , such that  $(s, \lambda, s') \in \rightsquigarrow$  and is denoted  $\mathbf{R}(s, s')$ .
- The exit rate of a state  $s$  is the sum of all rates and is denoted  $\mathbf{E}(s)$ . We require  $\mathbf{E}(s) < \infty$  for all  $s \in S$ .
- The discrete branching probability distribution quantifying race conditions is given by  $\mathbb{P}_s(s') = \mathbf{R}(s, s') / \mathbf{E}(s)$ .

## 6.1.2 Language Theoretic Concepts

Let  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$  be an MA. We define the usual language theoretic concepts. A *path*  $\pi$  of  $\mathcal{M}$  is a (possibly) infinite sequence of the form

$$\pi = s_0 t_1 \alpha_1 \mu_1 s_1 t_2 \alpha_2 \mu_2 s_2 \dots,$$

where  $s_i \in S$ ,  $t_i \in \mathbb{R}^+$ , and either  $\mu_i \in Distr(S)$  and  $\alpha_i \in Act$ , or  $\mu_i = \mathbb{P}_{s_i}$  and  $\alpha_i \in \mathbb{R}_0^+$  for  $i = 1, 2, \dots$ . We require that each finite path ends in a state,



and either  $s_{i-1} \xrightarrow{\alpha_i} \mu_i$  with  $\mu(s_i) > 0$ , or  $s_{i-1} \overset{\alpha_i}{\rightsquigarrow} s_i$  for each non-final  $i$ . The sequence  $s_{i-1} t_i \alpha_i \mu_i s_i$  means that the MA resided  $t_i$  time units in state  $s_{i-1}$  before moving to  $s_i$  using action  $\alpha_i$  via distribution  $\mu_i$ . If  $\pi$  is finite, then it ends in the state  $last(\pi)$ . We write  $\pi' \sqsubseteq \pi$  to denote  $\pi'$  as a *prefix* of  $\pi$ , i.e.  $\pi'$  is finite, ends in a state, and coincides with  $\pi$  on the first finitely many symbols of the sequence. The set of all finite paths of  $\mathcal{M}$  is set as  $paths^{fin}(\mathcal{M})$ , and all paths by  $paths(\mathcal{M})$ .

Note that a single time point has probability zero to occur in any given continuous time span. Hence, it is necessary to talk about *time intervals* instead of individual time values. This gives rise to *abstract paths*. An abstract path is a path, where each occurrence of single time values  $t_i$  is replaced by intervals  $I_i \subseteq \mathbb{R}_0^+$ . However, we are only interested in intervals of the form  $[0, t]$  with  $t \in \mathbb{R}_0^+$ . Thus, any path can be replaced with its corresponding abstract path by changing  $t$  to  $[0, t]$ , or vice versa. Hence, there is an obvious one-to-one mapping between paths and abstract paths. Throughout this chapter, we use  $\pi$  to denote a path and  $\Pi$  to denote the corresponding abstract path, or vice versa. We summarise all abstract finite paths in the set  $AbsPaths^{fin}(\mathcal{M})$ , and all abstract paths in  $AbsPaths(\mathcal{M})$ . For two abstract paths  $\Pi$  and  $\Pi'$  with

$$\Pi = s_0 I_1 \alpha_1 \mu_1 s_1 \dots s_n \text{ and } \Pi' = s_0 I'_1 \alpha'_1 \mu'_1 s'_1 \dots,$$

we say  $\Pi$  is a prefix of  $\Pi'$ , denoted  $\Pi \sqsubseteq \Pi'$ , if  $\alpha_i = \alpha'_i$ ,  $s_i = s'_i$ ,  $\mu_i = \mu'_i$  and  $I_i = I'_i$  for  $i = 1, 2, \dots, n$ . That is,  $\Pi$  and  $\Pi'$  coincide on the first  $n$  steps.

The *trace* of a path  $tr(\pi)$  records its visible behaviour, i.e. time and actions. It is a mapping  $tr : paths(\mathcal{M}) \rightarrow (\mathbb{R}_0^+ \times Act)^\omega$ . We overload  $tr$  to account for finite paths, i.e.  $tr : paths^{fin}(\mathcal{M}) \rightarrow (\mathbb{R}_0^+ \times Act)^*$ . Hence, a trace is the (possibly) infinite sequence of the form

$$\sigma = tr(\pi) = t'_{i_1} a_{i_1} t'_{i_2} a_{i_2} t'_{i_3} a_{i_3} \dots,$$

where  $t'_{i_j} \in \mathbb{R}_0^+$  and  $a_{i_j} \in Act$  for  $j = 1, 2, \dots$ . Note that a path fragment  $s_1 t_1 \lambda \mu_1 s_2 t_2 a \mu_2 s_3$  collapses to  $(t_1 + t_2) a$  if  $\lambda$  is a Markovian action. This rule is applied recursively in case of multiple Markovian actions. The set  $tr^{-1}(\sigma)$  is the set of all paths, which have  $\sigma$  as their trace. The *length* of a path  $\pi$ , denoted  $|\pi|$  is the number of actions on its trace. All finite traces of  $\mathcal{M}$  are summarized in  $traces^{fin}(\mathcal{M})$ , and all traces in  $traces(\mathcal{M})$ .

Similar to abstract paths, an *abstract trace* is given, if all  $t_i \in \mathbb{R}_0^+$  of a trace are replaced by intervals  $I_i \subseteq \mathbb{R}_0^+$ . Again, we are only interested in abstract traces using intervals of the form  $[0, t]$  with  $t \in \mathbb{R}_0^+$ . This induces a one-to-one mapping between traces and abstract traces. Like for paths, we denote the corresponding abstract trace of a trace  $\sigma$  via  $\Sigma$ . We summarise all finite abstract traces in the set  $AbsTraces^{fin}(\mathcal{M})$ , and all abstract traces in  $AbsTraces(\mathcal{M})$ . We define the prefix relation for abstract traces similarly to the prefix relation for abstract paths. Let  $act(\pi)$  return the *action path* of  $\pi$  by removing all values  $t_i$  and distributions  $\mu_i$ . For traces  $act(\sigma)$  returns visible actions only.

### 6.1.3 Stoic Trace Semantics

Trace semantics represent a linear view on a system's behaviour and are essential to understand how it evolves over time. Trace semantics for MAs are given by their trace distributions, assigning probabilities to measurable sets of traces, and are established via schedulers. A scheduler resolves all non-deterministic choices in the MA. Its trace distribution is obtained by removing all invisible behaviour, leading to a purely probabilistic execution tree.

Schedulers form the core concept of our framework. Given any finite piece of history leading to the current state, a scheduler returns a probability measure over time and the available transitions. The essential difference to existing schedulers [196] and schedulers of Chapter 5 is the ability to wait before scheduling an action. This feature is crucial for semantics of open systems. One MA may have to wait until its environment is ready to synchronize on a certain action.

**Definition 6.3.** *A stoic scheduler  $\mathcal{A}$  of an MA  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$  is a measurable function*

$$\mathcal{A} : paths^{fin}(\mathcal{M}) \longrightarrow Meas(\mathbb{R}_0^+ \times ([Act_\tau \times Distr(S)] \cup \{\perp\}))$$

such that 1. only available transitions or halting are chosen, and 2. internal actions are immediate and cannot be postponed, i.e.

1.  $\forall \pi \in paths^{fin}(\mathcal{M}) : \mathcal{A}(\pi)(\mathbb{R}_0^+ \times (a, \mu)) > 0 \implies (last(\pi), a, \mu) \in \rightarrow.$
2. *If  $\tau \in enabled(last(\pi))$ , then  $\mathcal{A}(\pi)(0 \times (Act_\tau \times Distr(S)) \cup \{\perp\}) = 1.$*

The value  $\mathcal{A}(\pi)(t, \perp)$  is the probability to interrupt/halt the process at time  $t$ . A scheduler  $\mathcal{A}$  halts on path  $\pi$  at time  $t \in \mathbb{R}_0^+$ , if  $\mathcal{A}(\pi)(t, \perp) = 1$ . We say a scheduler is of length  $k \in \mathbb{N}$ , if it halts for all paths  $\pi$  with length greater or equal than  $k$  at time 0. We denote this set by  $Sched(\mathcal{M}, k)$  and the set of all finite schedulers by  $Sched(\mathcal{M})$ , respectively.

The key difference to the schedulers of Definition 5.8 is that stoic schedulers may assign a measure over time after every finite path, as opposed to only discrete probability distributions. Thus, rather than accumulating all probability mass at time zero, stoic schedulers may spread out the probability to choose an action over time. Naturally, this leads to an additional race condition between Markovian actions and scheduler decisions.

To illustrate, let  $X$  be the random variable with underlying probability measure accounting for scheduler decisions, and let  $Y$  be the random variables accounting for all Markovian actions. That is,  $Y$  is exponentially distributed with the exit rate of a state as parameter. In order to calculate the probability to see a particular action (Markovian, or not) within time interval  $I = [0, t]$ , we are studying events like  $P(X < Y \wedge X < t)$ , or  $P(Y < X \wedge Y < t)$ .

This has vast implications for Markovian states; For instance, a scheduler might choose to halt the execution *after some time*. However, before the halting is executed, a Markovian action might intervene, resulting in a new scheduler

decision in the new state etc. The only way to guarantee halting is by scheduling at time zero, where the probability of any Markovian action to take place is zero.

A scheduler induces a probability measure to all abstract finite paths. Since the probability to choose an action at precisely time  $T$  is zero, we require the Lebesgue integral [178] to assign probability to time intervals  $[0, T]$  in case the scheduler chooses a continuous probability distribution.

**Definition 6.4.** *Let  $\mathcal{A}$  be a scheduler of  $\mathcal{M}$ , then we define the path probability function  $Q^{\mathcal{A}} : \text{AbsPaths}^{\text{fin}}(\mathcal{M}) \rightarrow [0, 1]$  inductively by  $Q^{\mathcal{A}}(s_0) = 1$ , and for given  $\Pi$  let  $s = \text{last}(\Pi)$ , then  $Q^{\mathcal{A}}(\Pi \cdot I\alpha\mu s') = Q_{\mathcal{A}}(\Pi) \cdot$*

$$\begin{cases} \int_0^T \int_x^\infty \mathcal{A}(\pi)(x, \alpha, \mu) \cdot \mu(s') \mathbf{E}(s) e^{-\mathbf{E}(s)t} dt dx & \text{if } \alpha \in \text{Act}_\tau \\ \int_0^T \int_t^\infty \sum_{(s,a,\mu) \in \rightarrow} \mathcal{A}(\pi)(x, a, \mu) \cdot \alpha \cdot e^{-\mathbf{E}(s)t} dx dt & \text{if } \alpha \in \mathbb{R}^+ \end{cases},$$

and the probability to halt after  $\Pi$  in time interval  $I$  is given as

$$Q^{\mathcal{A}}(\Pi) \cdot \int_0^T \int_x^\infty \mathcal{A}(\pi)(x, \perp) \mathbf{E}(s) e^{-\mathbf{E}(s)t} dt dx$$

Here  $I = [0, T] \subseteq \mathbb{R}^+$ , and  $\pi$  is the corresponding path to the abstract path  $\Pi$ .

The path probability function assigns the unique starting state probability 1, and each following transition either multiplies by the probability of taking the Markovian transition or by taking a scheduled probabilistic transition in a time interval  $I$ . In addition, if  $\mathcal{A}(\pi)$  is a measure, there is a race condition between Markovian actions and scheduler decisions taking place. The probability to see action  $\alpha \in \text{Act}_\tau$  in time interval  $I$ , is the product of the probability that the scheduler wins the race *and* whatever probability mass the scheduler to the given interval to  $\alpha$ . Conversely, the probability to perform  $\alpha \in \mathbb{R}^+$ , is the product of the probability that Markovian actions win the race *and* the probability that a particular Markovian action wins the Markovian race condition in interval  $I$ .

We summarize all infinite paths and all paths that *may halt* in the set of maximal paths.

**Definition 6.5.** *A path  $\pi$  of a scheduler  $\mathcal{A}$  is a finite or infinite path*

$$\pi = s_0 t_1 \mu_1 \alpha_1 s_1 t_2 \mu_2 \alpha_2 s_2 t_3 \mu_3 \alpha_3 s_3 \dots,$$

where  $\mathcal{A}(s_0 t_1 \mu_1 \alpha_1 s_1 \dots s_i)(t_i, \mu_i, \alpha_i) > 0$  for each  $i = 1, 2, \dots$ . The maximal paths of a scheduler  $\mathcal{A}$  are the infinite paths of  $\mathcal{A}$  and the finite paths  $\pi$  such that  $\mathcal{A}(\pi)(t, \perp) > 0$  with  $t \in \mathbb{R}_0^+$ . The maximal abstract paths of a scheduler  $\mathcal{A}$  are the corresponding abstract paths of maximal paths of  $\mathcal{A}$ . We denote paths<sup>max</sup>( $\mathcal{A}$ ) as the set of maximal paths, and  $\text{AbsPaths}^{\text{max}}(\mathcal{A})$  as the set of maximal abstract paths of  $\mathcal{A}$ .

Like before, a scheduler and its induced path probability function lead to a probability space via the standard cone construction [158].

**Definition 6.6.** *The probability space associated to a scheduler  $\mathcal{A}$  of an MA  $\mathcal{M}$  is the probability space  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$ , where*

- $\Omega_{\mathcal{A}} = \text{AbsPaths}^{\text{max}}(\mathcal{A})$ ,
- $\mathcal{F}_{\mathcal{A}}$  is the smallest  $\sigma$ -field generated by the set  $\left\{C_{\Pi} \mid \Pi \in \text{AbsPaths}^{\text{fin}}(\mathcal{M})\right\}$ , where  $C_{\Pi} = \{\Pi' \in \Omega_{\mathcal{A}} \mid \Pi \sqsubseteq \Pi'\}$ , and
- $P_{\mathcal{A}}$  is the unique probability measure on  $\mathcal{F}_{\mathcal{A}}$ , such that  $P_{\mathcal{A}}[C_{\Pi}] = Q^{\mathcal{A}}(\Pi)$  for all  $\Pi \in \text{AbsPaths}^{\text{fin}}(\mathcal{M})$

Standard measure theory arguments [46] ensure that  $P_{\mathcal{A}}$  induces a unique probability measure on the measurable space  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}})$ . Hence, a scheduler induces the unique probability space  $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, P_{\mathcal{A}})$ .

**Trace Distributions.** A trace distribution is obtained from the probability space of a scheduler by removing all invisible information. The probability assigned to a set of abstract traces  $X$ , is the probability assigned to all abstract paths whose abstract trace is an element of  $X$ .

**Definition 6.7.** *For a scheduler  $\mathcal{A}$  of a Markov automaton  $\mathcal{M}$ , we define its trace distribution  $\mathcal{D} = \text{trd}(\mathcal{A})$  as the probability space  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$ , where*

- $\Omega_{\mathcal{D}} = \text{AbsTraces}(\mathcal{M})$ ,
- $\mathcal{F}_{\mathcal{D}}$  is the smallest  $\sigma$ -field generated by the set  $\left\{C_{\Sigma} \mid \Sigma \in \text{AbsTraces}^{\text{fin}}(\mathcal{M})\right\}$ , where  $C_{\Sigma} = \{\Sigma' \in \Omega_{\mathcal{D}} \mid \Sigma \sqsubseteq \Sigma'\}$ , and
- $P_{\mathcal{D}}$  is the unique probability measure for  $\mathcal{F}_{\mathcal{D}}$ , such that the probability measure  $P_{\mathcal{D}}(X) = P_{\mathcal{A}}(\text{Tr}^{-1}(X))$ , where  $X \in \mathcal{F}_{\mathcal{D}}$ .

Since  $P_{\mathcal{A}}$  is the unique probability measure induced by scheduler  $\mathcal{A}$ , we conclude that  $P_{\mathcal{D}}$  is the unique probability measure on the measurable space  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}})$ . Thus, a scheduler induces a unique *trace distribution*  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$ .

**Trace distribution equivalence.** Trace distributions are the probabilistic counterpart to traces. They quantify the probability to observe abstract traces. Thus, it is natural to regard trace distributions as relation between two Markov automata, i.e. we relate two Markov automata  $\mathcal{M}_1$  and  $\mathcal{M}_2$  if they have the same set of trace distributions, and write  $\mathcal{M}_1 =_{TD} \mathcal{M}_2$ . The intuition is, that an external observer cannot distinguish  $\mathcal{M}_1$  and  $\mathcal{M}_2$  based on observed traces and regards them as equivalent.

A trace distribution is of length  $k$ , if it is based on a scheduler of length  $k$ . The set of such trace distributions is denoted  $\text{trd}(\mathcal{M}, k)$ , and the set of all finite trace distributions by  $\text{trd}(\mathcal{M})$ , respectively. This captures the notion of trace distribution inclusion, denoted  $\mathcal{M}_1 \sqsubseteq_{TD}^k \mathcal{M}_2$  iff  $\text{trd}(\mathcal{M}_1, k) \subseteq \text{trd}(\mathcal{M}_2)$ .

Trace distribution equivalence thus joins other equivalence relations defined for Markov automata. While [196] follow an equivalent approach in equating IMCs based on trace distributions, the main focal point of the literature has so far been on bisimulation [67, 59, 162]. We explore other relations in greater detail in Section 6.3.

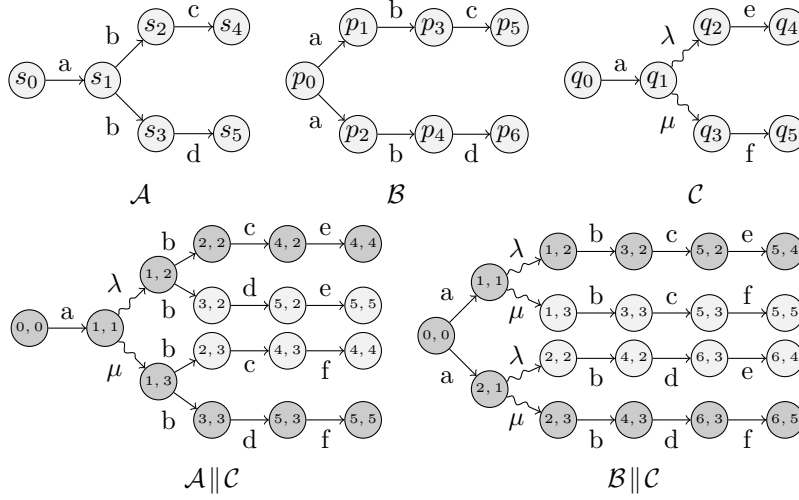


Figure 6.1: Counterexample that shows that compositionality of the trace distribution semantics does not hold for IMCs. Note that states in  $\mathcal{A} \parallel \mathcal{C}$  and  $\mathcal{B} \parallel \mathcal{C}$  were abbreviated, and that states not relevant for the counterexample were withheld for readability.

### 6.1.4 Compositionality

By being a conservative extension of both probabilistic automata and interactive Markov chains, Markov automata naturally inherit the property of being non-compositional under trace distributions. That is, two trace distribution equivalent Markov automata  $\mathcal{M}_1$  and  $\mathcal{M}_2$  in general do not have the same trace distribution under parallel composition with a third Markov automaton  $\mathcal{M}_3$ . Non-compositional of both PAs and IMCs is a well known problem [164], and equipping schedulers with the capability to wait does not circumvent this.

We give a classical counterexample known from the literature [164] for IMCs. Recall that an IMC is a degenerate MA where every distribution in  $\rightarrow$  is the Dirac distribution. Thus, non-compositional of trace distributions carries over, if we consider that IMCs are a subset of MAs, and that parallel composition of the latter is a conservative extension of the former.

**Theorem 6.8.** *Stoic trace distribution semantics are not compositional, i.e. for interactive Markov chains  $\mathcal{A}, \mathcal{B}$  and  $\mathcal{C}$  it generally does **not** hold that*

$$\text{trd}(\mathcal{A}) = \text{trd}(\mathcal{B}) \Rightarrow \text{trd}(\mathcal{A} \parallel \mathcal{C}) = \text{trd}(\mathcal{B} \parallel \mathcal{C}).$$

*Proof.* Consider the IMCs in Figure 6.1. We have  $\text{trd}(\mathcal{A}) = \text{trd}(\mathcal{B})$ . However, there are trace distributions in  $\mathcal{A} \parallel \mathcal{C}$ , which are not in  $\mathcal{B} \parallel \mathcal{C}$  and vice versa.

Consider the deterministic scheduler in  $\mathcal{A} \parallel \mathcal{C}$ , that assigns probability 1 at time 0 to the  $b$  transition from  $(s_1, q_2)$  to  $(s_2, q_2)$  and probability 1 at time 0 to the  $b$  transition from  $(s_1, q_3)$  to  $(s_3, q_3)$ . A scheduler of  $\mathcal{B} \parallel \mathcal{C}$  intending to

imitate this behaviour has to assign probability 1 to both  $a$  transitions in  $(p_0, q_0)$  at time 0. Obviously, this is impossible, hence there is no scheduler in  $\mathcal{B} \parallel \mathcal{C}$  that can imitate this behaviour. The dark-gray states in Figure 6.1 illustrate this. Non-compositionality for IMCs then implies non-compositionality for MA.  $\square$

## 6.2 A Testing Scenario

A fundamental scheme in concurrency theory is that two processes are deemed equivalent, if an external observer cannot distinguish them. A vast number of behavioural equivalences have been characterized via intuitive testing scenarios or button pushing experiments in various settings [56, 134], including systems with discrete probability [40, 122, 158]. Except for [196], such notions have hardly been studied in the context of IMCs and MAs.

Our testing scenario uses a black-box called *timed trace machine* from which actions and their timing are recorded and collected in a sample. It is similar to the one studied in Chapter 5. Provided a large enough sample, we can distinguish two MAs that are not (finite) trace distribution equivalent up to a given confidence level  $\alpha$ .

Thus, the goal of this section is to establish a relation between two Markov automata based on their observations, that implies trace distribution equivalence. First, we describe the sampling process, and how samples relate to a single Markov automaton. This is similar to the sampling process used in previous chapters. After establishing all acceptable observations for a Markov automaton, we proceed to relate two Markov automata via their observations. Lastly, we extend the theorem by Cheung, Stoelinga & Vaandrager [40] to finite trace distributions of Markov automata. The theorem states that two probabilistic automata are trace distribution equivalent, iff they are equivalent with respect to their observations.

Since we are working in a continuous time domain, each individual trace has probability 0. Hence, we identify recorded times  $t$  with the intervals  $[0, t]$  and thus map traces to their corresponding abstract traces. Recall that this induces a one-to-one mapping between traces and abstract traces. Therefore, we use both interchangeably in this section.

### 6.2.1 Sampling and Expectations

Assume that the underlying model of the black-box depicted in Figure 6.2 is a Markov automaton. The black-box is equipped with a reset button, as well as time- and action windows. The eponymous reset button resets the system to its initial state, whereas both the time- and the action window help an external observer record traces. The action window displays the currently executed action, while the time window shows the relative passed time since the last action.

We perform a button pushing experiment in the sense of [134], by recording visible actions and their respective time occurrence. After recording a visible action, the timer is set back to zero and starts anew. Given two such black-box

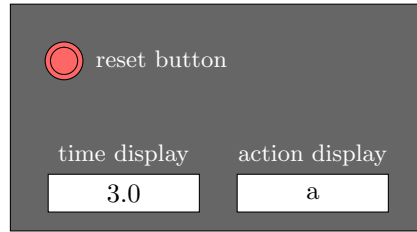


Figure 6.2: Black-box timed trace machine with reset button, time and action windows. The action window displays the current action, while the time window shows the relative time that passed since the last action.

MAs  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , we say they are *observational equivalent*, if the frequencies of all observable traces coincide. Naturally, an external observer does not have a view on the inner workings of the machine. Our goal is to establish a result in the lines of Proposition 4.30, stating that two Markov automata are finitely trace distribution equivalent iff they are observationally equivalent. To that end, we first define when to relate an observation to a trace distribution by means of *acceptable outcomes*.

**Sampling.** An external observer performs various experiments on the black-box, and records a sample. Before conducting an experiment, parameters for trace length  $k \in \mathbb{N}$  and sample size  $m \in \mathbb{N}$  are set. The parameters characterise how many visible actions need to be recorded before restarting the machine, and how many runs of the machine are recorded. This yields a sample consisting of  $m$  traces of length  $k$ .

The intuition of the experiment is to compare the *measure induced by the trace frequencies of a sample* to the *expected measure*. The deviation of both measures is quantified, and compared to a threshold. We call the sample an *acceptable outcome* for a given level of significance  $\alpha \in (0, 1)$ , if the deviation is below the threshold. This is similar to accepting a coin as fair, if the number of *heads* ranges between 42 and 58 for  $\alpha = 0.05$ .

We proceed by defining the probability measures quantifying deviation.

**Expected Measure.** For given parameters  $k, m \in \mathbb{N}$  we assume that the machine is governed by the same trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{M}, k)$  in each of the  $m$  runs. In Section 6.1 we showed that  $\mathcal{D}$  induces a unique probability space, assigning each trace of length  $k$  a probability.

We treat each run of the trace machine as Bernoulli experiment, where success occurs if trace  $\sigma$  is observed, and failure if not. Let  $X_i \sim \text{Ber}(P_{\mathcal{D}}(\Sigma))$  for  $i = 1, \dots, m$  be Bernoulli distributed random variables. We define  $Z = \frac{1}{m} \sum_{i=1}^m X_i$  as the empirical mean of  $\sigma$  in  $m$  runs. Note that

$$\mathbb{E}^{\mathcal{D}}(Z) = \mathbb{E}^{\mathcal{D}}\left(\frac{1}{m} \sum_{i=1}^m X_i\right) = \frac{1}{m} \sum_{i=1}^m \mathbb{E}^{\mathcal{D}}(X_i) = P_{\mathcal{D}}(\Sigma).$$

Since this holds for all traces  $\sigma$ ,  $\mathbb{E}^{\mathcal{D}}$  defines the expected measure.

**Sample Measure.** The expected measure is compared to the sample measure, i.e. the probability measure derived from a gathered sample. We use the information gained from a sample in two steps: 1. collect traces with the same action behaviour in *trace classes*, and 2. within a trace class, we use order statistics to make inference about time intervals [77]. For a sample  $O$ , the class of a trace  $\sigma$  is defined as  $\Sigma_{\sigma} = \{\varrho \in O \mid \text{act}(\varrho) = \text{act}(\sigma)\}$ . The frequency of any abstract trace  $\Sigma$  within a sample is then given as

$$\text{freq}(O)(\Sigma) = \frac{|\Sigma_{\sigma}|}{m} \prod_{i=1}^k \frac{|\{\varrho \in \Sigma_{\sigma} \mid t_i^{\varrho} \leq t_i^{\sigma}\}|}{|\Sigma_{\sigma}|},$$

where  $t_i^{\varrho}$  denotes the  $i$ -th time stamp of trace  $\varrho$ . Like in Chapter 5 the frequency function assumes the independence of all time intervals from each other. This approach is warranted, since neither Markov automata nor scheduled probability measures make use of *clocks*.

This holds for all  $\sigma$ , hence,  $\text{freq}(O)$  defines the sample measure. Note that the empirical measure of each time step converges uniformly to the unknown real measure after the Glivenko-Cantelli Theorem as  $m \rightarrow \infty$ , cf. [77]. That is, the sample measure becomes increasingly more accurate with growing sample size.

ID	Recorded Trace $\sigma$						$\#\sigma$
$\sigma_1$	0	a	0.1	b	2.8	c	1
$\sigma_2$	0	a	0.5	b	2.3	c	1
$\sigma_3$	0	a	3.6	b	0.3	c	1
$\sigma_4$	0	a	3.8	b	0.4	c	1
$\sigma_5$	0	a	2.2	d	1.3	c	1
$\sigma_6$	0	a	2.4	d	1.4	c	1

Table 6.1: Illustrating sample with  $k = 3$  and  $m = 6$  drawn from a black-box timed trace machine.

**Example 6.9.** Table 6.1 shows a possible sample recorded from a timed trace machine. It contains  $m = 6$  traces of length  $k = 3$ , together with their ID and the amount of equivalent traces we observed. Naturally, recording in continuous time makes it highly unlikely to record the exact same trace more than once, barring rounding. The example shows two distinct trace classes  $\Sigma_{\sigma_1} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$  and  $\Sigma_{\sigma_5} = \{\sigma_5, \sigma_6\}$ . Measuring real-time makes grouping them any further impractical, e.g. judging whether time stamps 0.1 and 0.5 belong to the same trace, and would be highly vulnerable to different rounding techniques.

However, by using the frequency function, we can still derive the induced sample measure. To illustrate, the probability to see  $\Sigma = [0, 1] a [0, 3] b [0, 2] c$  equates to  $\text{freq}(O)(\Sigma) = \frac{4}{6} \cdot (1 \cdot \frac{2}{4} \cdot \frac{2}{4}) = \frac{1}{6}$ . Note that we essentially infer three



*independent properties from a sample in tandem: what is the probability of 1. recording a within one time unit, 2. recording b within three time units, and 3. recording c within two time units.*

### 6.2.2 Observational Equivalence

We set a level of significance  $\alpha \in (0, 1)$ . If the sample measure induced by  $O$  lies within some distance  $r_\alpha > 0$  from the expected measure under  $\mathcal{D}$ , we call  $O$  an *acceptable outcome*. Here  $r_\alpha$  is chosen such that the probability to identify a truthful sample of the Markov automaton is at least  $1 - \alpha$ . This reflects the error of false rejection from hypothesis testing. The union of all acceptable outcomes forms the set of observations.

**Definition 6.10.** *For  $k, m \in \mathbb{N}$  and an MA  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$  the acceptable outcomes under  $\mathcal{D} \in \text{trd}(\mathcal{M}, k)$  of significance level  $\alpha \in (0, 1)$  are given by*

$$\text{Obs}(\mathcal{D}, k, m, \alpha) = \{O \in (\mathbb{R}_{\geq 0} \times Act)^{\leq k \times m} \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha\},$$

where  $\text{dist}(\mu, \nu) = \sup_{\Sigma \in ([0, t] \times Act)^{\leq k}} |\mu(\Sigma) - \nu(\Sigma)|$  is the total variation distance of two measures. We obtain the set of acceptable outcomes of  $\mathcal{M}$  by

$$\text{Obs}(\mathcal{M}, k, m, \alpha) = \bigcup_{\mathcal{D} \in \text{trd}(\mathcal{M}, k)} \text{Obs}(\mathcal{D}, \alpha, k, m).$$

The set of acceptable outcomes of a Markov automaton provides a link between externally visible behaviour and trace distributions. Going back to the example of a fair coin, this is similar to relating the interval [42, 58] for the amount of observed *heads* to the probability 0.5 to flip *heads*.

The acceptable outcomes provide yet another way to characterize equivalence of two Markov automata. We say two Markov automata are *observationally equivalent*, if their respective sets of acceptable outcomes coincide. This is the pendant to equating two coins, if their acceptance intervals coincide.

While trace distributions are inaccessible for an external observer, samples and observations are not. Thus, we strive to characterize their trace distribution equivalence via observational equivalence. That is: two Markov automata are observationally equivalent if and only if they have the same set of finite trace distributions. This extends the theorem by Cheung, Stoeling & Vaandrager [40] used as Proposition 4.30 for probabilistic automata to Markov automata.

**Theorem 6.11.** *Let  $\mathcal{M}_1$  and  $\mathcal{M}_2$  be two MAs. Let  $k \in \mathbb{N}$  and  $\alpha \in (0, 1)$ . Then there exists  $M \in \mathbb{N}$  such that for all  $m > M$  we have*

$$\text{trd}(\mathcal{M}_1, k) = \text{trd}(\mathcal{M}_2, k) \iff \text{Obs}(\mathcal{M}_1, k, m, \alpha) = \text{Obs}(\mathcal{M}_2, k, m, \alpha).$$

*Proof sketch.* Note that [40] show the result for probabilistic automata. Our proof follows the existing one closely, but is eased by the fact, that we only consider finite trace distributions. The “ $\implies$ ” is trivial, since the set of observations

is defined via trace distributions. The proof of “ $\Leftarrow$ ” is done via contra-position and uses two auxiliary lemmas. The first establishes the set of finite trace distributions of Markov automata to be closed. Hence,  $trd(\mathcal{M}_1) \not\subseteq trd(\mathcal{M}_2)$  implies that these two sets have a positive distance  $\delta$  via some distance function. The second lemma ensures that there is  $m \in \mathbb{N}$ , such that the observations contained in the  $\frac{\delta}{3}$ -ball around the expected measure has a probability greater or equal to  $1 - \alpha$ . The observations are contained in these  $\frac{\delta}{3}$ -balls and hence do not intersect. This shows that  $Obs(\mathcal{M}_1, \alpha, k, m) \not\subseteq Obs(\mathcal{M}_2, \alpha, k, m)$ .  $\square$

The eponymous black-box prohibits an external observer to study its inner workings. This is similar to the model-based testing hypotheses of the previous chapters. We assume every black-box functions according to an underlying Markov automaton, that is inaccessible to us. Theorem 6.11 lets us infer the relation of two Markov automata, by making *sufficiently* many observations.

### 6.3 Relation to other Equivalences

Studying the relations between process equivalences is a natural way to investigate their distinguishing power. In the style of [184], we compare trace distribution equivalence to notions of bisimulation, thereby relating in one spectrum the MA equivalences based on bisimulation, testing and traces. A natural and desirable property is that trace-based relations are coarser than bisimulation. We show that this holds for stoic trace distribution equivalence, but fails for existing notions from the literature, that is if we generalize to MAs the trace equivalence on IMCs. The rest of this section is dedicated to establish a hierarchy presented at its end in Figure 6.5. The hierarchy includes trace distribution equivalence by Baier et al. [196], strong and weak bisimulation by Eisentraut et al. [67] and Hennessy and Deng [59], as well as late weak bisimulation by Song et al. [162]. Naturally, this requires certain repetitions of their equivalence definitions first.

#### 6.3.1 Trace Distribution Equivalence by Baier et al.

An equivalence relation for IMCs using an approach eminently similar to ours is given in [196]. The framework resolves non-determinism via schedulers and builds a probabilistic computation tree to measure probabilities of abstract traces. The core difference to the approach of this chapter is, that in [196] schedulers cannot decide to wait before scheduling action. Evidently, this is also the framework used in Chapter 5.

**Definition 6.12.** *A scheduler  $\mathcal{A}$  for an MA  $\mathcal{M}$  is called a BMW-scheduler if we require*

$$\mathcal{A} : \text{paths}^{\text{fin}}(\mathcal{M}) \longrightarrow \text{SubDistr}((\text{Act}_\tau \times \text{Distr}(S)) \cup \{\perp\}).$$

**Remark 6.13.** *We lifted the work of Baier, Majster-Cederbaum and Wolf (hence BMW scheduler) [196] to MAs to capture the essential properties of*

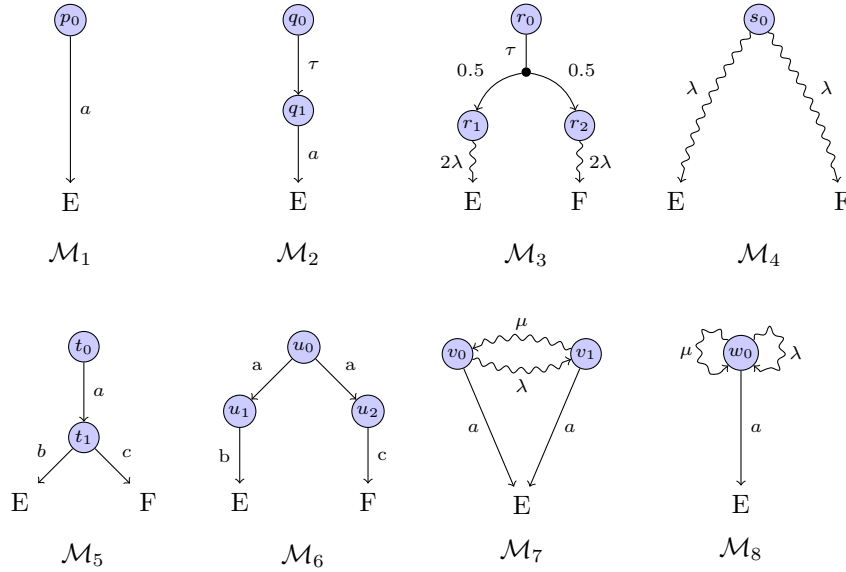


Figure 6.3: Yardstick examples to relate Markov automata. Throughout this section we study equivalence relations that distinguish or relate each pair.

their schedulers. In [196] a scheduler returns a discrete sub-distribution over available actions and states. The remaining probability mass is assigned to wait for Markovian transitions. If no such transition is present, the process halts. We explicitly schedule halting here.

In contrast to schedulers defined in our sense, if a BMW-scheduler decides to schedule an action, it must do so without delay. This makes BMW-schedulers a subset of stoic schedulers and justifies probability spaces based on BMW-schedulers and BMW-trace distributions respectively. Hence, trace distributions of BMW-schedulers are denoted by  $Trd_{BMW}(\mathcal{M})$ . Further, two MAs are BMW-equivalent, denoted by  $\mathcal{M}_1 =_{BMW} \mathcal{M}_2$ , if  $Trd_{BMW}(\mathcal{M}_1) = Trd_{BMW}(\mathcal{M}_2)$ .

Despite the evident subset relation by construction of BMW- and stoic trace distributions, their induced equivalence relations are incomparable. This fact is spoiled by the maximal progress assumption, which forces immediate progress and prohibits waiting.

**Theorem 6.14.** *BMW-trace distribution equivalence, and stoic trace distribution equivalence are incomparable.*

*Proof.* Consider the Markov automata in Figure 6.4.

We see that  $\mathcal{M}_1 =_{BMW} \mathcal{M}_2$ , since either  $a$  or  $b$  have to be scheduled immediately in  $\mathcal{M}_1$  using BMW-schedulers. A BMW-scheduler of  $\mathcal{M}_2$  can then schedule  $a$  or  $\tau$  with the same probability chosen in  $\mathcal{M}_1$  and vice versa. However, we have  $\mathcal{M}_1 \neq_{TD} \mathcal{M}_2$ , because a scheduler of  $\mathcal{M}_1$  might decide to wait before

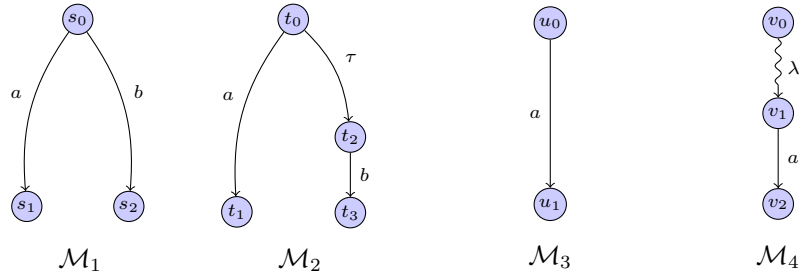


Figure 6.4: Distinguishing example for stoic trace distribution equivalence and BMW-trace distribution equivalence. It is  $\mathcal{M}_1 =_{BMW} \mathcal{M}_2$ , but  $\mathcal{M}_1 \neq_{TD} \mathcal{M}_2$  and  $\mathcal{M}_3 =_{TD} \mathcal{M}_4$ , but  $\mathcal{M}_3 \neq_{BMW} \mathcal{M}_4$ .

scheduling  $a$ . This behaviour cannot be mimicked by schedulers of  $\mathcal{M}_2$ , because  $\tau$  is enabled in  $t_0$  and waiting is prohibited due to maximal progress.

On the contrary, we have  $\mathcal{M}_3 =_{TD} \mathcal{M}_4$ . Intuitively, the Markovian transition of rate  $\lambda$  in  $\mathcal{M}_4$  can be imitated by a stoic scheduler in  $\mathcal{M}_3$  scheduling  $a$  based on the measure of an exponential distribution with rate  $\lambda$  in  $u_0$ . This is not possible, when we restrict schedulers to the BMW-type, and hence  $\mathcal{M}_3 \neq_{BMW} \mathcal{M}_4$ .  $\square$

### 6.3.2 Bisimulation

Bisimulation takes a state-wise view on equating two systems. In strong bisimulation two states are considered equivalent, if they enable the exact same stepwise behaviour. However, internal actions might make it impossible for an external observer to distinguish states, which are not strongly bisimilar. The notion of weak bisimulation abstracts away from internal actions. Eisentraut et al. [67] laid the foundation of defining weak bisimulation for Markov automata, such that it conservatively extends weak bisimulation for both IMCs and PAs. Their work was refined by Deng and Hennessy [59] and later by Song et al. [66, 162].

**Strong bisimulation  $\approx_s$ .** We first recall strong bisimulation for Markov automata as given in [67]. To relate two Markov automata, we relate their respective initial states in their *direct sum*, i.e. the union of states, actions and probabilistic and Markovian transitions.

**Definition 6.15.** Let  $\mathcal{M} = \langle S, s_0, Act_\tau, \rightarrow, \rightsquigarrow \rangle$  be an MA. An equivalence relation  $\mathcal{R}$  on  $S$  is a strong bisimulation iff  $s \mathcal{R} s'$  implies for all  $\alpha \in Act_\tau \cup \mathbb{R}^+$  :  $s \xrightarrow{\alpha} \mu$  implies  $s' \xrightarrow{\alpha} \mu'$  with  $\mu(C) = \mu'(C)$  for all classes of equivalent states  $C \in S/\mathcal{R}$ .

Two MAs  $\mathcal{M}_1, \mathcal{M}_2$  are strongly bisimilar  $\mathcal{M}_1 \approx_s \mathcal{M}_2$ , if their initial states are contained in a strong bisimulation of their direct sum.

**Weak Bisimulation  $\approx_w$  and  $\approx_w^\circ$ .** Instead of relating two states, we relate two (sub)distributions over states. We write  $\mu \xrightarrow{\alpha} \mu'$  with  $\alpha \in Act_\tau \cup \mathbb{R}^+$ , if there

is a transition  $s \xrightarrow{\alpha} \mu_s$  for all  $s \in \text{supp}(\mu)$ , such that  $\mu' = \sum_{s \in \text{supp}(\mu)} \mu(s) \mu_s$ .

To abstract away from internal actions, [61, 59] first define weak transitions as *hyper-derivations*. They are needed in the definition of weak bisimulation. Let  $\mu \xRightarrow{\tau} \mu'$ , if there are  $\mu = \mu_0^{\rightarrow} + \mu_0^{\times}$ ,  $\mu_0^{\rightarrow} \xrightarrow{\tau} \mu_1^{\rightarrow} + \mu_1^{\times}$ ,  $\mu_1^{\rightarrow} \xrightarrow{\tau} \mu_2^{\rightarrow} + \mu_2^{\times}$ ,  $\dots$ , where  $\mu' = \sum_{i \geq 0} \mu_i^{\times}$ . We write  $\mu \xRightarrow{\alpha} \mu'$  if there exists  $\mu \xRightarrow{\tau} \mu'' \xRightarrow{\alpha} \mu'$ .

Lastly, for a transition relation  $\rightarrow: \text{Distr}(S) \times \text{Act}_{\tau} \times \text{Distr}(S)$ , we let  $\mu \xrightarrow{\alpha}_C \mu'$ , if there exist a finite set of real numbers  $w_i > 0$  and transitions  $\mu \xrightarrow{\alpha} \mu_i$ , such that  $\sum_i w_i = 1$  and  $\mu' = \sum_i w_i \cdot \mu_i$ . We call  $\rightarrow_C$  the *combined transition* of  $\rightarrow$ .

Following [59], we give an alternate representation of MAs in the form of *Markov labelled transition systems* (MLTSs) by treating race conditions as *one* probabilistic transition.

**Definition 6.16.** *A Markov labelled transition system (MLTS) is a quadruple  $\mathfrak{M} = \langle S, s_0, \text{Act}_{\tau}, \rightarrow \rangle$ , where*

- $S$  is a set of states with unique starting state  $s_0$ ,
- $\text{Act}_{\tau}$  is a set of action labels containing the distinguished label  $\tau$ , and
- $\rightarrow \subseteq S \times (\text{Act}_{\tau} \cup \mathbb{R}^+) \times \text{Distr}(S)$ .

Thus every MA  $\mathcal{M}$  has a corresponding MLTS  $\mathfrak{M} = \langle S, s_0, \text{Act}_{\tau}, \rightarrow \rangle$ , where  $\rightarrow \subseteq S \times (\text{Act}_{\tau} \cup \mathbb{R}^+) \times \text{Distr}(S)$ . Every probabilistic state in an MLTS has at least one outgoing transition for each of its enabled actions, but at most one outgoing transition for all of its Markovian transitions.

We are now able to recapture weak bisimulation taken from [59]. Albeit there are several notions of it, each reformulation proves that they are *essentially the same*, cf. [67, 162]. Deng and Hennessy also show that their weak bisimulation relation is the coarsest reduction barbed congruence possible [59].

**Definition 6.17.** *Let  $\mathfrak{M} = \langle S, s_0, \text{Act}_{\tau}, \rightarrow \rangle$  be an MLTS. For a given relation  $\mathcal{R}$  over the set  $\text{Distr}(S) \times \text{Distr}(S)$  let  $\mathcal{B}(\mathcal{R})$  be the relation over  $\text{Distr}(S) \times \text{Distr}(S)$  determined by  $\Delta \mathcal{B}(\mathcal{R}) \Theta$ , if for all  $\alpha \in \text{Act}_{\tau} \cup \mathbb{R}^+$  and for all  $p_1, \dots, p_n$  with  $\sum_{i=1}^n p_i = 1$  we have:*

1. *If  $\Delta \xRightarrow{\alpha} \sum_{i=1}^n p_i \cdot \Delta_i$  for any distributions  $\Delta_i$ , there are distributions  $\Theta_i$  with  $\Theta \xRightarrow{\alpha} \sum_{i=1}^n p_i \cdot \Theta_i$ , such that  $\Delta_i \mathcal{R} \Theta_i$  for each  $i = 1, \dots, n$ .*
2. *If  $\Theta \xRightarrow{\alpha} \sum_{i=1}^n p_i \cdot \Delta_i$  for any distributions  $\Delta_i$ , there are distributions  $\Delta_i$  with  $\Delta \xRightarrow{\alpha} \sum_{i=1}^n p_i \cdot \Delta_i$ , such that  $\Delta_i \mathcal{R} \Theta_i$  for each  $i = 1, \dots, n$ .*

*A relation  $\mathcal{R}$  is called weak bisimulation if  $\mathcal{R} \subseteq \mathcal{B}(\mathcal{R})$ . The largest of these weak bisimulations is denoted by  $\approx_w$ . Two MAs  $\mathcal{M}_1, \mathcal{M}_2$  are weakly bisimilar  $\mathcal{M}_1 \approx_w \mathcal{M}_2$  if the initial distributions of their MLTSs are contained in a weak bisimulation in the direct sum.*

Note that Deng and Hennessy [59] point out, that the restriction to full distributions in Definition 6.17 is insignificant and can be replaced by sub-distributions.

**Remark 6.18.** *Weak bisimulation definitions for IMCs frequently add an additional clause to relate states that have different exit rates, e.g.  $v_0$  and  $v_1$  of Figure 6.3. Checking clauses 1. and 2. in Definition 6.17 except when  $\alpha \in \mathbb{R}^+$  and  $\mu \mathcal{R} \nu$  extends this property to MA. This gives rise to weak bisimulation ignoring Markovian self-loops, denoted as  $\approx_w^\circ$ . It is then easy to see that  $\bar{v}_0 \mathcal{R} \bar{v}_1$  also for  $\mu \neq \nu$  in  $\mathcal{M}_7$  and  $\approx_w \subset \approx_w^\circ$ . Here  $\bar{s}$  is the Dirac distribution, i.e.  $\text{supp}(\bar{s}) = \{s\}$ .*

**Late Weak Bisimulation  $\approx_{lw}$  and  $\approx_{lw}^\circ$ .** Song et al. [162] observe that weak bisimulation does not equate  $\mathcal{M}_3$  and  $\mathcal{M}_4$  in Figure 6.3, and introduce *late weak bisimulation*. The idea is to interpret Markovian transitions as a sequence of probabilistic choices followed by sojourn time distributions. A technical report on late weak bisimulation is provided in [162] and a version for PAs in [66].

We recall some notation used in [162]: A distribution  $\mu$  is called *transition consistent*, denoted  $\vec{\mu}$ , if for all states  $s \in \text{supp}(\mu)$  and  $\alpha \neq \tau$ , we have  $s \xrightarrow{\alpha} \gamma$  implies  $\mu \xrightarrow{\alpha} \gamma'$  for some distributions  $\gamma, \gamma'$ . Intuitively, all states in the support of a transition consistent distribution have the same enabled action set.

Lastly, [162] introduce another way to lift transitions of states to transitions of distributions. The main difference is that internal actions cannot be blocked in a bisimulation sense. For two distributions  $\mu, \mu'$ , we write  $\mu \xrightarrow{\alpha} \mu'$  with  $\alpha \in \text{Act}_\tau \cup \mathbb{R}^+$  if either

1. for all  $s \in \text{supp}(\mu)$ , there is  $s \xrightarrow{\alpha} \mu_s$  such that  $\mu' = \sum_{s \in \text{supp}(\mu)} \mu(s) \cdot \mu_s$  or
2.  $\alpha = \tau$  and there is  $s \in \text{supp}(\mu)$  and  $s \xrightarrow{\alpha} \mu_s$  such that  $\mu' = (\mu - s) + \mu(s) \cdot \mu_s$ .

**Definition 6.19.** *Let  $\mathfrak{M} = \langle S, s_0, \text{Act}_\tau, \rightarrow \rangle$  be an MLTS. We call a given relation  $\mathcal{R} \subseteq \text{Distr}(S) \times \text{Distr}(S)$  late weak bisimulation, iff  $\mu \mathcal{R} \nu$  implies*

1. For  $\mu \xrightarrow{\theta}_C \mu'$ , there exists  $\nu \xrightarrow{\theta}_C \nu'$  such that  $\mu' \mathcal{R} \nu'$ .
2. If not  $\vec{\mu}$ , then there exists  $\mu = \sum_{i=0}^n p_i \cdot \mu_i$  and  $\nu \xrightarrow{\tau}_C \sum_{i=0}^n p_i \cdot \nu_i$  such that  $\vec{\mu}_i$  and  $\mu_i \mathcal{R} \nu_i$  for  $i = 1, \dots, n$  and  $\sum_{i=0}^n p_i = 1$ .
3. Symmetrical for  $\nu$ .

The largest of these late weak bisimulations is denoted  $\approx_{lw}$ . Two MAs  $\mathcal{M}_1, \mathcal{M}_2$  are late weak bisimilar, denoted  $\mathcal{M}_1 \approx_{lw} \mathcal{M}_2$ , if the initial distributions of their MLTSs are contained in a late weak bisimulation.

Note that [196] show, that weak bisimulation for IMCs implies BMW-trace distribution equivalence, when restricted to time-abstract deterministic schedulers. The following result is complementary to theirs.

**Theorem 6.20.** *Late weak bisimulation is strictly finer than BMW-trace distributions equivalence, i.e.  $\approx_{lw} \subset =_{BMW}$ .*

*Proof sketch.* For two MLTSs  $\mathfrak{M}_1$  and  $\mathfrak{M}_2$ , we show that  $\mathfrak{M}_1 \approx_{lw} \mathfrak{M}_2$  implies  $\text{Trd}_{BMW}(\mathfrak{M}_1) \subseteq \text{Trd}_{BMW}(\mathfrak{M}_2)$  as the inverse follows by symmetry. Thus, for a

given trace distribution  $\mathcal{D} \in \text{trd}(\mathfrak{M}_1)$  we construct a scheduler  $\mathcal{A}' \in \text{Sched}(\mathfrak{M}_2)$ , such that  $\text{trd}(\mathcal{A}') = \mathcal{D}$ . This is done via induction over trace distribution length. The construction requires careful treatment of classes of late weak bisimilar states, in particular those belonging to abstract paths, which have the same abstract trace. By induction assumption, we know there is a scheduler, that induces the same probabilities for all abstract traces of length  $k$ . The induction step requires us to show that there is a scheduler that continues this property for abstract paths of length  $k + 1$ . The proof hinges on two observations made by [66]: If a distribution  $\mu$  is transition consistent, then so is its late weak bisimilar equivalent  $\nu$ . If  $\mu$  is not transition consistent, then  $\nu$  is also not transition consistent. However, there exists a splitting of  $\mu$  and  $\nu$ , such that both are. Recall that transition consistency of a distribution means that each element in their support has the same enabled action set. With the addition of some corner cases, e.g.  $\tau$  enabled or not, we can thus find a scheduler that assigns all abstract traces the same probability.  $\square$

**Remark 6.21.** *Similar to weak bisimulation ignoring Markovian self-loops, we can make an exception in clause 1 of Definition 6.19 for  $\theta \in \mathbb{R}^+$  and  $\mu \mathcal{R} \nu$ . This gives rise to late weak bisimulation ignoring Markovian self-loops, denoted as  $\approx_{lw}^\circ$ . Then, obviously  $\approx_{lw} \subset \approx_{lw}^\circ$ .*

**Proposition 6.22.** *The inclusion  $\approx_s \subset \approx_w \subset \approx_{lw}$  holds and is strict.*

Song et al. [162] prove that the largest class of schedulers, that guarantees trace distribution equivalence under late weak bisimulation are partial information schedulers. This is not in contrast with our results, since the most general class of schedulers they consider are deterministic (cf. *Example 6* in [162]).

**Theorem 6.23.** *Late weak bisimulation ignoring Markovian self-loops is strictly finer than trace distribution equivalence, i.e.  $\approx_{lw}^\circ \subsetneq_{TD}$ .*

*Proof sketch.* The proof relies on a similar construction like the one of Theorem 6.20. However, the *maximal progress* assumption states that internal actions must be taken without delay. In order to prove the statement, we need an additional lemma. The lemma shows that two late weak bisimilar sub-distributions over states are always late weak bisimilar to sub-distributions over stable states, i.e. states not enabling  $\tau$ -actions. This stable sub-distribution can be reached with probability 1 from the unstable sub-distribution. Additionally, the convex combination of two measures is a measure. Thus, independently of the measure scheduled by  $\mathcal{D} \in \text{trd}(\mathcal{M}_1)$ , a scheduler  $\mathcal{A}' \in \text{Sched}(\mathcal{M}_2)$  can copy it.  $\square$

**Remark 6.24.** *Song et al. [162] point out that weak bisimulation according to [59] and [67] proved to be congruences for time convergent Markov automata. When considering divergent processes [93] suggests relating two states (resp. distributions) if both of them are divergent, or none of them is. Another approach is given by [59] in the form of indefinite delays or passive transitions. Song. et al [162] remark that these aspects can be incorporated into late weak bisimulation.*

### 6.3.3 Hierarchy

The previous two sections represent a cross-cut through the equivalence relations for Markov automata. We present the established hierarchy in Figure 6.5 and point out, that the reverse implications do not hold. To that end, we return to the yardstick examples seen at the beginning of this section to provide an illustration for the various equivalences.

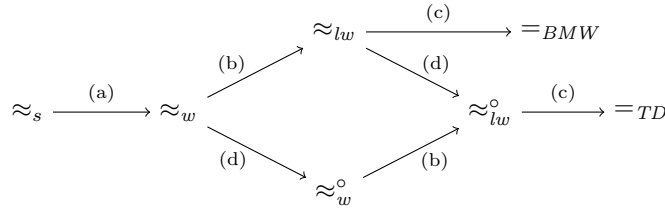


Figure 6.5: Hierarchical overview of equivalence relations, comparing strong bisimulation  $\approx_s$  [67], weak bisimulation  $\approx_w$  [67], weak bisimulation ignoring Markovian self-loops  $\approx_w^\circ$ , late weak bisimulation  $\approx_{lw}$  [162], late weak bisimulation ignoring Markovian self-loops  $\approx_{lw}^\circ$ , trace distribution equivalence by Baier et al.  $=_{BMW}$  [196] and trace distribution equivalence  $=_{TD}$ .

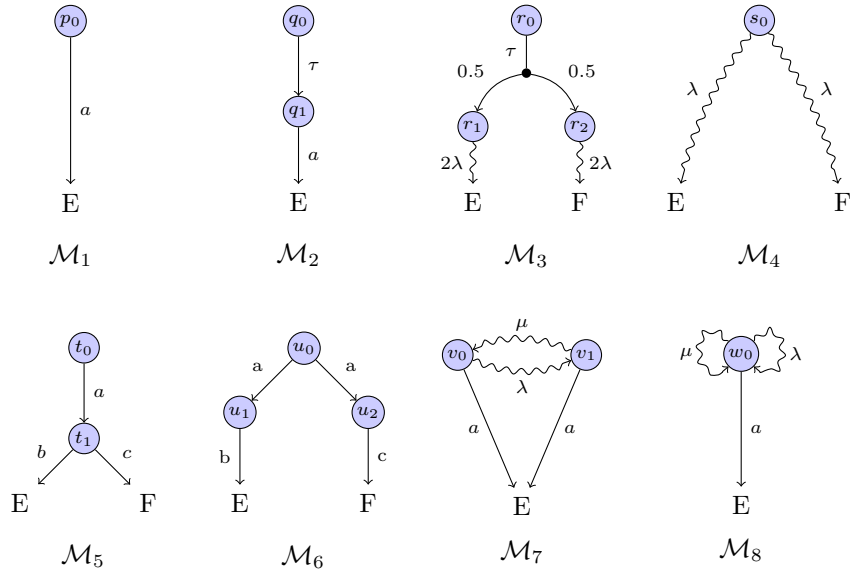


Figure 6.6: Distinguishing examples with respect to the hierarchy of Figure 6.5. For details, see Theorem 6.25.

**Theorem 6.25.** *The reverse implications presented in Figure 6.5 do not hold.*



*Proof.* We present distinguishing examples in Figure 6.6.

- (a) We have  $\mathcal{M}_1 \approx_w \mathcal{M}_2$ , but  $\mathcal{M}_1 \not\approx_s \mathcal{M}_2$ , since strong bisimulation does not abstract away from internal  $\tau$  actions.
- (b) We have  $\mathcal{M}_3 \approx_{lw} \mathcal{M}_4$  and  $\mathcal{M}_3 \approx_{lw}^\circ \mathcal{M}_4$ , but  $\mathcal{M}_3 \not\approx_w \mathcal{M}_4$  and  $\mathcal{M}_3 \not\approx_w^\circ \mathcal{M}_4$ , since weak bisimulation cares for the order of the probabilistic and the Markovian distributions. Late weak bisimulation does not distinguish  $r_0$  and  $s_0$ , because the time and probability to reach  $E$  and  $F$  are the same.
- (c) We have  $\mathcal{M}_5 =_{BMW} \mathcal{M}_6$  and  $\mathcal{M}_5 =_{TD} \mathcal{M}_6$ , but  $\mathcal{M}_5 \not\approx_{lw} \mathcal{M}_6$  and  $\mathcal{M}_5 \not\approx_{lw}^\circ \mathcal{M}_6$ . There is no state in  $\mathcal{M}_6$  that enables the  $b$  and the  $c$  action, hence there is no bisimilar state to  $t_1$ . Trace distribution equivalence cares for the probabilities and times of traces and equates  $\mathcal{M}_5$  and  $\mathcal{M}_6$ .
- (d) We have  $\mathcal{M}_7 \approx_w^\circ \mathcal{M}_8$  and  $\mathcal{M}_7 \approx_{lw}^\circ \mathcal{M}_8$ , but  $\mathcal{M}_7 \not\approx_w \mathcal{M}_8$  and  $\mathcal{M}_7 \not\approx_{lw} \mathcal{M}_8$ , since weak and late weak bisimulation do not ignore Markovian self-loops.

For more details on these examples, we refer to [67], [134], [162] and [196].  $\square$

## 6.4 Conclusions

We presented a novel notion of trace distributions equivalence for Markov automata by augmenting schedulers with the ability to wait before proceeding with scheduled actions. This establishes an equivalence relation for Markov automata, which is coarser than existing frameworks for weak bisimulation. Despite being an extension to similar trace-based approaches, we show that our trace distributions are incomparable to trace distributions not allowing waiting time, due to the maximal progress assumption.

Giving schedulers the opportunity of allowing waiting time lets us relate processes, which emit the same visible behaviour for an external observer. We underlined this notion with a testing scenario in form of a button pushing experiment, and showed that two Markov automata are observationally equivalent, if and only if they have the same trace distributions.

A widely agreed upon opinion, is that general scheduler classes model too powerful behaviour [66]. It is therefore important for future work to study more restricted scheduler classes. Moreover, we see potential in the application of our framework to model-based testing of stochastic systems, where a waiting scheduler models the delayed input of a user. This potential is under the premise that better methods to infer the underlying scheduler of a gathered sample exist in the statistical analysis, e.g. via more restricted scheduler classes.

## 6.5 Proofs

We present the proofs of the theorems within this chapter, alongside some lemmata. Reoccurring theorems are numbered according to their occurrence in

the chapter, while additional content is numbered alphanumerically. Note that some results are proven immediately after they are stated in the main body of the chapter, based on their simplicity or illustration of our methods. As to not impede the reading flow, we present the remaining theorems in this section.

**Lemma A.** *The set of finite trace distributions of a Markov automaton  $\mathcal{M}$  is closed.*

*Proof.* Trace distributions for Markov automata are probability spaces in the sense of Definition 6.7. Note that [40] show that the set of trace distributions for probabilistic automata is closed, i.e. probability spaces based on *Act*. Additionally, the set of closed intervals on  $\mathbb{R}_0^+$  is closed. Since the Cartesian product of two closed sets is a closed set, and we only consider finite trace distributions, we conclude that the set of trace distributions for Markov automata is closed.  $\square$

**Lemma B.** *For a Markov automaton  $\mathcal{M}$  and any given  $\delta > 0$ , there exists  $M \in \mathbb{N}$ , such that for all  $m \geq M$  and  $\mathcal{D} \in \text{trd}(\mathcal{M}, k)^m$ , we have*

$$P_{\mathcal{D}}(\text{freq}^{-1}(B_{\delta}(\mathbb{E}^{\mathcal{D}}))) \geq 1 - \alpha.$$

*Proof.* Let  $M > \frac{1}{4\delta^2\alpha}$ . By Chebychev's inequality [77], we have for all  $m > M$

$$P_{\mathcal{D}}(\{O \in \text{Obs}(\mathcal{M}) \mid \exists \sigma \in O : |\text{freq}(O)(\sigma) - \mathbb{E}^{\mathcal{D}}(\sigma)| \geq \delta\}) \leq \frac{1}{4\delta^2m} \leq \alpha.$$

This yields for all  $m > M$  that

$$\begin{aligned} & P_{\mathcal{D}}(\text{freq}^{-1}(B_{\delta}(\mathbb{E}^{\mathcal{D}}))) \\ &= P_{\mathcal{D}}(\{O \in \text{Obs}(\mathcal{M}) \mid \forall \sigma \in O : |\text{freq}(O)(\sigma) - \mathbb{E}^{\mathcal{D}}(\sigma)| < \delta\}) \\ &= 1 - P_{\mathcal{D}}(\{O \in \text{Obs}(\mathcal{M}) \mid \exists \sigma \in O : |\text{freq}(O)(\sigma) - \mathbb{E}^{\mathcal{D}}(\sigma)| \geq \delta\}) \\ &\geq 1 - \alpha. \end{aligned}$$

$\square$

**Theorem 6.11.** *Let  $\mathcal{M}_1$  and  $\mathcal{M}_2$  be two MAs. Let  $k \in \mathbb{N}$  and  $\alpha \in (0, 1)$ . Then there exists  $M \in \mathbb{N}$  such that for all  $m > M$  we have*

$$\text{trd}(\mathcal{M}_1, k) = \text{trd}(\mathcal{M}_2, k) \iff \text{Obs}(\mathcal{M}_1, \alpha, k, m) = \text{Obs}(\mathcal{M}_2, \alpha, k, m).$$

*Proof.* The “ $\implies$ ” direction is trivial, since observations are based on trace distributions (Definition 6.10). For the converse, assume  $\text{trd}(\mathcal{M}_1, k) \not\subseteq \text{trd}(\mathcal{M}_2, k)$ , then there exists a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{M}_1, k) \setminus \text{trd}(\mathcal{M}_2, k)$ . We need to show that  $\text{Obs}(\mathcal{M}_1, \alpha, k, m) \not\subseteq \text{Obs}(\mathcal{M}_2, \alpha, k, m)$ . By Lemma A the set of finite trace distributions is closed. Hence, there is  $\delta > 0$  such that  $\text{dist}(\text{trd}(\mathcal{M}_1, k), \text{trd}(\mathcal{M}_2, k)) > \delta$ , for some distance function  $\text{dist}$ . Let  $\mathcal{D}' \in \text{trd}(\mathcal{M}_2, k)$  such that  $\text{dist}(P_{\mathcal{D}}, P_{\mathcal{D}'}) = \text{dist}(\text{trd}(\mathcal{M}_1, k), \text{trd}(\mathcal{M}_2, k))$ . By Lemma B, we know that there exist  $M_1, M_2 \in \mathbb{N}$  such that for all  $m > \max(M_1, M_2)$

$$P_{\mathcal{D}}(\text{freq}^{-1}(B_{\frac{\delta}{3}}(\mathbb{E}^{\mathcal{D}}))) \geq 1 - \alpha \quad \text{and} \quad P_{\mathcal{D}'}(\text{freq}^{-1}(B_{\frac{\delta}{3}}(\mathbb{E}^{\mathcal{D}'}))) \geq 1 - \alpha.$$

This yields

$$Obs(\mathcal{D}, \alpha, k, m) \subseteq freq^{-1}(B_{\frac{\delta}{3}}(\mathbb{E}^{\mathcal{D}})) \text{ and } Obs(\mathcal{D}', k, m, \alpha) \subseteq freq^{-1}(B_{\frac{\delta}{3}}(\mathbb{E}^{\mathcal{D}'})).$$

Since  $dist(P_{\mathcal{D}}, P_{\mathcal{D}'}) > \delta$ , we have  $B_{\frac{\delta}{3}}(P_{\mathcal{D}}) \cap B_{\frac{\delta}{3}}(P_{\mathcal{D}'}) = \emptyset$ . From here we deduce that  $Obs(\mathcal{D}, \alpha, k, m) \not\subseteq Obs(\mathcal{M}_2, \alpha, k, m)$ , and hence  $Obs(\mathcal{M}_1, \alpha, k, m) \not\subseteq Obs(\mathcal{M}_2, \alpha, k, m)$ . The other direction holds by symmetry.  $\square$

**Theorem 6.20.** *Late weak bisimulation is strictly finer than BMW-trace distributions equivalence, i.e.  $\approx_{lw} \subsetneq =_{BMW}$ .*

*Proof.* Let  $\mathcal{M}_1$  and  $\mathcal{M}_2$  be two MLTSs. Assume  $\mathcal{M}_1 \approx_{lw} \mathcal{M}_2$ , where  $\mathcal{R}$  is a late weak bisimulation. We have to show that  $\mathcal{M}_1 =_{BMW} \mathcal{M}_2$ . Note that it is sufficient to show that

$$\forall D \in Trd_{BMW}(\mathcal{M}_1) \exists D' \in Trd_{BMW}(\mathcal{M}_2) \forall \Sigma \in (I \times Act)^* : P_{\mathcal{D}}(\Sigma) = P_{\mathcal{D}'}(\Sigma),$$

where  $I \subseteq \mathbb{R}_0^+$  are intervals, since the converse follows by symmetry.

Let  $\mathcal{D} \in Trd_{BMW}(\mathcal{M}_1)$  be a BMW-trace distribution, then there is  $\mathcal{A} \in Sched_{BMW}(\mathcal{M}_1)$ , such that  $trd(\mathcal{A}) = \mathcal{D}$ . We show the existence of  $\mathcal{D}'$  by induction over trace distribution length. Assume  $|\Sigma| = 0$ , then by definition of trace distributions (Definition 6.7), we obviously have  $P_{\mathcal{D}}(\varepsilon) = P_{\mathcal{D}'}(\varepsilon) = 1$ .

Assume we have shown that  $P_{\mathcal{D}}(\Sigma) = P_{\mathcal{D}'}(\Sigma)$  for all  $\Sigma$  with  $|\Sigma| = n$ . Let now  $\Sigma \in (I \times Act)^{n+1}$ . Then

$$\Sigma = I_1 a_1 \dots I_n a_n I_{n+1} a_{n+1} \text{ and } \Sigma' = I_1 a_1 \dots I_n a_n,$$

where  $I_i \subseteq \mathbb{R}_0^+$  are intervals,  $a_i \in Act$ , and  $\Sigma' \sqsubseteq_n \Sigma$ .

Since  $\Sigma$  is an abstract trace, there are distinct abstract paths  $\Pi_1, \dots, \Pi_k$  with  $tr(\Pi_i) = \Sigma$  for  $i = 1, \dots, k$ . Let  $\Pi'_1, \dots, \Pi'_l$  be the distinct abstract paths with  $tr(\Pi'_j) = \Sigma'$  for  $j = 1, \dots, l$  such that for all  $\Pi'_j$  exists  $\Pi_i$  such that  $\Pi'_j \sqsubseteq \Pi_i$ . Note that  $l \leq k$ . Let  $\delta'_j = P_{\mathcal{A}}(\Pi'_j)$  for  $j = 1, \dots, l$ . Then  $P_{\mathcal{A}}(\Pi_i) = \delta'_j \cdot p_i$  for some corresponding  $\delta'_j \in [0, 1]$  and  $p_i \in [0, 1]$ .

Since  $\mathcal{M}_1 \approx_{lw} \mathcal{M}_2$ , there are abstract paths  $\hat{\Pi}'_1, \dots, \hat{\Pi}'_{l'}$  in  $\mathcal{M}_2$ , such that for all  $last(\Pi'_i)$ , there is  $last(\hat{\Pi}'_j)$  such that  $last(\Pi'_i) \mathcal{R} last(\hat{\Pi}'_j)$  (the Dirac distribution of the states). Equivalently, let  $\hat{\Pi}_1, \dots, \hat{\Pi}_{k'}$  be abstract paths in  $\mathcal{M}_2$ , such that for each  $last(\Pi_i)$  there is  $last(\hat{\Pi}_j)$  with  $last(\Pi_i) \mathcal{R} last(\hat{\Pi}_j)$  (the Dirac distribution of the states). By induction assumption, we have

$$\sum_{i=1}^l P_{\mathcal{D}}(tr(\Pi'_i)) = \sum_{j=1}^{l'} P_{\mathcal{D}'}(tr(\hat{\Pi}'_j)).$$

Let  $C'_i = [last(\Pi'_i)]_{\mathcal{R}} \subseteq S_1$  be the equivalence class of (Dirac distributions over) states in  $S_1$  for each  $\Pi'_1, \dots, \Pi'_l$ . It is possible that  $C'_i = C'_j$  for some  $i \neq j$ , so assume there are  $m \leq l$  distinct such  $C'_i$ . Equivalently, let  $\hat{C}'_1, \dots, \hat{C}'_m$  the corresponding equivalence classes in  $\mathcal{M}_2$ . We overload  $tr$  by denoting  $tr(C'_i) =$

$\{\Sigma \in (I \times Act)^* \mid \Sigma = tr(\Pi) \text{ with } last(\Pi) \in C_i\}$ . By induction assumption, we can assume without loss of generality, that

$$P_{\mathcal{D}}(tr(C'_i)) = P_{\mathcal{D}'}(tr(\hat{C}'_i)) \text{ for each } i = 1, \dots, m.$$

It now suffices to show, that there is  $\mathcal{A}' \in Sched(\mathcal{M}_2)$  with  $trd(\mathcal{A}') = \mathcal{D}'$  such that  $P_{\mathcal{D}}(tr(C_i)) = P_{\mathcal{D}'}(tr(\hat{C}_i))$  for each  $i = 1, \dots, w$ , where  $C_i = [last(\Pi_i)]_{\mathcal{R}} \subseteq S_1$  and  $\hat{C}_j = [last(\hat{\Pi}_j)]_{\mathcal{R}} \subseteq S_2$ .

Let  $\mu \in \text{SubDistr}(C'_i)$  for some  $i$ . Since  $\mathcal{M}_1 \approx_{lw} \mathcal{M}_2$ , there is a sub-distribution  $\nu \in \text{SubDistr}(\hat{C}'_i)$  such that  $\mu \mathcal{R} \nu$ . We need to show, that  $\mathcal{D}'$  can assign a total of  $\sum_{i \in \mathcal{I}} p_i$  to states in distribution  $\nu$  to go via the last trace fragment  $I_{n+1}a_{n+1}$  to  $\hat{C}_j$  for some  $j$ . Here,  $\mathcal{I}$  is the set of indices for abstract paths  $\Pi'_r$  such that  $\Pi'_r \sqsubseteq \Pi_s$  for some  $\Pi_s$  with  $last(\Pi_s) \in C_j$ . Indeed, assume  $\mu \xrightarrow{\alpha}_C \mu'$ , then by late weak bisimulation, there is  $\nu \xrightarrow{\alpha}_C \nu'$ , such that  $\mu' \mathcal{R} \nu'$ . If  $\alpha \in Act_{\tau}$ , let

$$p_{\alpha}^i = \frac{1}{|C'_i|} \sum_{j=1}^{|C'_i|} \mathcal{A}(\pi'_j)(0, \alpha, \mu'),$$

where  $\pi'_i$  are the corresponding paths to  $\Pi'_i$  with  $last(\Pi'_i) \in C'_i$ . Intuitively,  $p_{\alpha}^i$  is the probability that scheduler  $\mathcal{A}$  assigns to equivalence class  $C'_i$  to schedule  $\mu'$ . If  $\alpha \in \mathbb{R}^+$ , then we define  $p_{\alpha}^i = 1 - \sum_{a \in Act_{\tau}} p_a^i$  as the remaining probability of a Markovian transition taking place.

- (a) If  $\alpha = \tau$ ,  $\vec{\mu}$  and  $\nu \neq \nu'$ , then  $\vec{\nu}$ . This is a result proven in [66]. Then there exists  $\nu_d \in \text{Distr}(S_2)$  with  $\nu \xrightarrow{\tau}_C \nu_1 \xrightarrow{\tau}_C \dots \xrightarrow{\tau}_C \nu_d \xrightarrow{\tau}_C \nu_{d+1} \xrightarrow{\tau}_C \nu_{\kappa} \xrightarrow{\tau}_C \nu'$  such that there is  $\mathcal{A}' \in \text{Sched}_{BMW}(\mathcal{M}_2)$ , with

$$\sum_{\hat{\pi}'_j \in \hat{C}'_i} \mathcal{A}(\hat{\pi}'_j 0, \tau, \nu_1 s_1, \dots, 0, \tau, \nu_{h-1} s_{h-1})(0, \tau, \nu_h) = \begin{cases} p_{\alpha}^i & \text{if } h = d \\ 1 & \text{if } h \neq d, \end{cases}$$

and  $Trd_{BMW}(\mathcal{A}') = \mathcal{D}'$ . This is possible, due to the transition consistency  $\vec{\nu}_h$  for  $h = 1, \dots, d$  implicitly given by  $\vec{\nu}$  and late weak bisimulation.

- (b) If  $\alpha = \tau$ ,  $\neg \vec{\mu}$  and  $\nu \neq \nu'$ , then  $\neg \vec{\nu}$ . This is a result taken from [66]. However, by late weak bisimulation, there exists a splitting of  $\mu = \sum_{j=1}^z w_j \mu_j$ , such that  $\nu \xrightarrow{\tau}_C \sum_{j=1}^z w_j \nu_j$  with  $\mu_j \mathcal{R} \nu_j$  and  $\vec{\mu}_j$ , and consequently  $\vec{\nu}_j$  for  $j = 1, \dots, z$ . Now, there exists a scheduler  $\mathcal{A}' \in \text{Sched}_{BMW}(\mathcal{M}_2)$  with  $Trd_{BMW}(\mathcal{A}') = \mathcal{D}'$ , that can reach  $\sum_{j=1}^z w_j \nu_j$  with probability 1, because  $\mu \mathcal{R} \nu$ . We proceed with (a) for  $\mu_j$  and  $\nu_j$  for  $j = 1, \dots, z$ .
- (c) If  $\alpha = \tau$  and  $\nu = \nu'$ , then  $\mu' \mathcal{R} \nu$ . Then  $\mathcal{D}'$  stays in  $\nu$  with probability 1.
- (d) If  $\alpha \in Act$  and  $\vec{\mu}$ . Similar to (a).
- (e) If  $\alpha \in Act$  and  $\neg \vec{\mu}$ . Similar to (b).
- (f) If  $\alpha \in \mathbb{R}^+$  and  $\vec{\mu}$ , then all  $s \in \text{supp}(\mu)$  are stable, i.e. no  $\tau$ -transitions are enabled. Consequently, also  $\vec{\nu}$ , and therefore, all states  $s \in \text{supp}(\nu)$  are stable. Since  $\mu \mathcal{R} \nu$  we know that  $\mathcal{D}'$  has  $p_{\alpha}^i$  as remaining probability for Markovian transitions to take place.

(g) If  $\alpha \in \mathbb{R}^+$  and  $\neg \vec{\mu}$ . This case is solved as a combination of (b) and (f).

Note that the choice of  $\mathcal{A}'$  solely depends on decisions of  $\mathcal{D}$  and not individual traces. Since all of the above cases are exclusive, there is a scheduler  $\mathcal{A}' \in \text{Sched}_{BMW}(\mathcal{M}_2)$  that fulfils all of them with  $\text{Trd}_{BMW}(\mathcal{A}') = \mathcal{D}'$ .

Finally, this shows for each  $C_i$  that

$$\begin{aligned} P_{\mathcal{D}}(\text{tr}(C_i)) &= \sum_{j=1}^{m_i} P_{\mathcal{D}}(\text{tr}(C'_j)) \cdot \sum_{k=1}^{|C_i|} p_{jk} \\ &= \sum_{j=1}^{\hat{m}_i} P_{\mathcal{D}'}(\text{tr}(\hat{C}'_j)) \cdot \sum_{k=1}^{|\hat{C}_i|} p_{jk} \\ &= P_{\mathcal{D}'}(\text{tr}(\hat{C}_i)), \end{aligned}$$

where  $p_{jk}$  are the probabilities in  $P_{\mathcal{A}}(\Pi_j) = P_{\mathcal{A}}(\Pi'_z)p_{jk}$  for each  $\text{last}(\Pi_j) \in C_i$  with  $\Pi'_z \sqsubseteq_n \Pi_j$  and  $\text{last}(\Pi'_z) \in C'_j$  for some  $j$  (the same for  $\hat{\Pi}_j$  and  $\hat{C}_i$ ). Summing over all  $C_i$  and  $\hat{C}_i$  respectively, gives  $P_{\mathcal{D}}(\Sigma) = P_{\mathcal{D}'}(\Sigma)$  for each  $\Sigma$ .

Thus, we showed the existence of  $\mathcal{D}' \in \text{Trd}_{BMW}(\mathcal{M}_2)$ , such that all abstract traces of  $\mathcal{M}_2$  get assigned the same probability as in  $\mathcal{M}_1$ . That yields  $\mathcal{M}_1 \subseteq_{BMW} \mathcal{M}_2$ . The converse holds by symmetry.  $\square$

**Lemma C.** *Let  $\mathcal{R}$  be a late weak bisimulation on a non-divergent finite Markov automaton  $\mathcal{M}$ . Let  $\mu, \nu \in \text{SubDistr}(S)$  with  $\mu \mathcal{R} \nu$  with  $\nu$  unstable. Then there exists  $\nu^* \in \text{SubDistr}(S)$  such that  $\mu \mathcal{R} \nu^*$  and  $\nu^*$  stable.*

*Proof.* Let  $\mu, \nu \in \text{SubDistr}(S)$  with  $\mu \mathcal{R} \nu$  and  $\nu$  unstable. There are two cases:

1. Assume  $\tau \in \text{enabled}(\mu)$ . Then there is  $\mu' \in \text{SubDistr}(S)$  with  $\mu \xrightarrow{\tau}_C \mu'$ . Since  $\mathcal{R}$  is a late weak bisimulation, there is  $\nu' \in \text{SubDistr}(S)$ , such that  $\nu \xrightarrow{\tau}_C \nu'$  and  $\mu' \mathcal{R} \nu'$ . Additionally,  $\mu' \mathcal{R} \nu$  and  $\mu \mathcal{R} \nu'$ . Then either  $\tau \in \text{enabled}(\mu')$  or  $\tau \notin \text{enabled}(\mu')$ . In the first case, we repeat case **1**, in the latter we go to case **2**. Note that this procedure can be repeated only finitely many times. Assume we infinitely often execute case **1**. Then either  $\mathcal{M}$  is divergent or  $\mathcal{M}$  is not finite. Both cases violate our assumptions.
2. Assume  $\tau \notin \text{enabled}(\mu)$ . Since  $\nu$  is unstable, there is  $\nu' \in \text{SubDistr}(S)$  with  $\nu \xrightarrow{\tau} \nu'$ . Since  $\mathcal{R}$  is a late weak bisimulation, there is  $\mu' \in \text{SubDistr}(S)$ , such that  $\mu \xrightarrow{\tau} \mu'$  and  $\nu' \mathcal{R} \mu'$ . Since  $\tau \notin \text{enabled}(\mu)$ , we necessarily have  $\mu \equiv \mu'$ . If  $\tau \in \text{enabled}(\nu')$ , then we repeat case **2**. If  $\tau \notin \text{enabled}(\nu')$ , then  $\nu' \equiv \nu^*$ , i.e.  $\nu^*$  is stable. Note that case **2** is repeated only finitely many times. If this were not the case, then  $\mathcal{M}$  would either be divergent or not finite.

$\square$

**Theorem 6.23** *Late weak bisimulation ignoring Markovian self-loops is strictly finer than trace distribution equivalence, i.e.  $\approx_{lw}^{\circ} \subseteq =_{TD}$ .*

*Proof sketch.* The proof arises as a combination of Theorem 6.23 and Lemma C. Instead of assigning probability sub-distributions to trace fragments, the trace distribution  $\mathcal{D}'$  assigns measures depending on the measures used by  $\mathcal{D}$ . By Lemma C a sub-distribution of states can always reach an equivalent and stable sub-distribution of states with probability one. Note that a scheduler can schedule any measures in a stable sub-distribution. Additionally, a finite convex combination of measures is a measure. Hence,  $\mathcal{D}'$  can assign the same measure to all equivalence classes and their corresponding trace fragments as  $\mathcal{D}$ .  $\square$

# CHAPTER 7

---

## Model-Based Testing with Stochastic Automata

---

Stochastic behaviour and requirements are an important aspect of today's complex systems: network protocols extensively rely on randomised algorithms [106, 165], cloud providers commit to service level agreements, the emerging field of probabilistic robotics [168] allows the automation of complex tasks via simple randomised strategies, as seen in e.g. vacuum and lawn mowing robots, and we see a proliferation of probabilistic programming languages [82].

Stochastic systems must satisfy stochastic requirements. Consider the example of exponential back-off in Ethernet studied in Chapter 4: an adapter that, after a collision, sometimes retransmits earlier than prescribed by the standard may not impact the overall functioning of the network, but may well gain an unfair advantage in throughput at the expense of overall network performance. In the case of cloud providers, the service level agreements are inherently stochastic when guaranteeing a certain availability (i.e. *average* uptime) or a certain distribution of maximum response times for different tasks. This has given rise to extensive research in stochastic *model checking* techniques [111]. However, in practice, *testing* remains the dominant technique to evaluate and certify systems outside of a limited area of highly safety-critical applications.

In this chapter, we present an MBT framework based on input-output stochastic automata (IOSA) [51], which are transition systems augmented with discrete probabilistic choices and timers whose expiration is governed by general probability distributions. IOSA mark the pinnacle of modelling formalisms in terms of both complexity and power of this thesis. By using IOSA models, we can quantitatively specify stochastic aspects of a system, in particular with respect to timing. We support discrete as well as continuous probability distributions, so our framework is suitable for both hard and soft real-time requirements. Since IOSA extend transition systems, non-determinism is available for underspecification as usual. We formally define the notions of stochastic ioco (**sa-ioco**), and of test cases as a restriction of IOSA. We then outline practical algorithms for test generation and **sa-ioco** conformance testing. The latter combines per-trace functional verdicts as in standard ioco with a statistical evaluation that builds upon the Kolmogorov-Smirnov test [99] known from non-parametric

statistics. While our theory of IOSA and **sa-ioco** is very general with respect to supported probability distributions and non-determinism, we need to assume some restrictions to arrive at practically feasible algorithms. We finally exemplify our framework’s capabilities and its inherent trade-offs by testing timing aspects of different implementation variants of the Bluetooth device discovery protocol. This illustrates differences to purely Markovian test theory as seen in Chapter 5 in a most practical way.

Since probabilistic automata and Markov automata are special cases of stochastic automata, our new framework generalises both previous ones. However, due to the memoryless assumption, the **Mar-ioco** conformance testing algorithm only had to compare the means of the specified and observed delays. We do not make any such assumption for **sa-ioco** instead we are checking the entire shape of the distribution via the Kolmogorov-Smirnov test matches. It is thus a more precise test, that comes at the slight increase of complexity.

We summarize the main contributions of this chapter:

- The input output stochastic automata (IOSA) model, comprising non deterministic choices, discrete probability distributions, and discretely and continuously distributed clocks for transition,
- a behavioural description for SA based on trace semantics,
- definitions of test cases, execution and verdicts,
- the soundness and completeness results of our framework,
- non-parametric tests for general continuous clock distributions, and
- the Bluetooth discovery protocol revisited with the SA framework.

**Related Work.** Early influential work had only deterministic time [21, 116, 120], later extended with time-outs/quiescence [29]. Probabilistic testing pre-orders and equivalences are well-studied [45, 60, 158]. Probabilistic bisimulation via hypothesis testing was first introduced in [122]. Our work is largely influenced by [40], which introduced a way to compare trace frequencies with collected samples. Closely related is work by [97, 138] on stochastic finite state machines, where stochastic delays are specified similarly, but discrete probability distributions over target states are not included.

**Origins of the chapter.** The work underlying this chapter was performed in collaboration with Arnd Hartmanns and Mariëlle Stoelinga, and appeared in

- Marcus Gerhold, Arnd Hartmanns, and Mariëlle Stoelinga. Model-based testing for general stochastic time. In *Proceedings of the 10th International Symposium on NASA Formal Methods, NFM*, pages 203–219, 2018.

**Organisation of the chapter.** Section 7.1 establishes the underlying automaton model in Stochastic automata, and recalls language theoretic concepts. The theory of testing with stochastic automata is explored in Section 7.2. This



includes the conformance relation, test cases, verdicts and the framework’s correctness. An algorithmic outline and applicable methods for the framework in practice are presented in Section 7.3. The Bluetooth device discovery protocol is revisited in Section 7.4, and differences to the Markov automaton framework are explored. The chapter ends with concluding remarks in Section 7.5.

## 7.1 Stochastic Automata

We introduce the underlying model for the remainder of this chapter in input/output stochastic automata (IOSA). Stochastic automata are transition systems augmented with non-determinism, discrete probability choices, and real-time clocks whose expiration is governed by general discrete or continuous probability distributions. Separating their alphabet into input- and output actions gives rise to IOSA. As such, they are a conservative extension of both probabilistic automata and Markov automata, as seen in Chapters 4 and 5. We refer to Figure 7.1 for an hierarchical overview. SA are the most powerful and complex modelling formalisms studied in this thesis, and thus mark the pinnacle of MBT frameworks in a probabilistic setting discussed here.

The remainder of this section is structured similarly to PA and MA; the automaton model is introduced alongside illustrating examples, and the meaning of paths and traces is explored. We define parallel composition for IOSA to enable communication between multiple components, or with the environment, and introduce schedulers. As before, schedulers ensure the quantification of probabilities of traces with respect to a probability measure, and induce *trace distributions*.

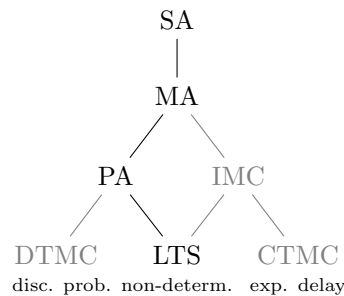


Figure 7.1: Traversing the automata formalism hierarchy shown in Figure 1.3.

### 7.1.1 Definition

SA [50] are transition systems augmented with non-determinism, discrete probability choices, and clocks whose expiration is governed by general probability distributions, which are either discrete or continuous. The action alphabet of SA is separated into inputs, outputs and internal actions, as we have done for other modelling formalisms. Working in an open setting, where an implementation may interact with other components, or the environment, warrants this distinction. This gives rise to input output stochastic automata (IOSA) [51].

While strictly being an extension of Markov automata, working in general continuous time not restricted to exponential distributions means IOSA lose the desirable *memoryless property*. To account for the loss, SA use *clocks*; real-valued

variables that increase synchronously with rate 1 over time and expire some random amount of time after they have been *restarted*. Their expiration time is drawn according to an underlying probability distribution. Therefore, we define SA to comprise *locations* rather than *states*; each location keeps track of all real valued clocks, while a single state comprises a location, an *evaluation* of all clocks, as well as their respective expiration times. The complexity of the state space of an SA becomes evident, as there are uncountably many clock valuations in each location. Similarly, each transition is a discrete probability distribution (potentially Dirac). We point out that IOSAs are *not* defined as input-reactive and output-generative this time around for technical reasons.

What sets SA apart further from previous models is the existence of both guards and clock resets on transitions; a transition may only be taken once all clocks in the guard sets are expired. This is similar to timed automata, except expiration times are randomly distributed according to a discrete or continuous probability distribution. The eponymous clock resets ensures clocks are properly set back to zero whenever it is desired.

**Definition 7.1.** An input-output stochastic automaton (IOSA) is a 6-tuple  $\mathcal{I} = \langle Loc, \mathcal{C}, Act, E, F, \ell_0 \rangle$  where

- $Loc$  is a countable set of locations, with  $\ell_0 \in Loc$  as the initial location,
- $\mathcal{C}$  is a finite set of clocks,
- $Act = Act_I \sqcup Act_O \sqcup Act_H$  is the finite action alphabet partitioned into inputs  $Act_I$ , outputs  $Act_O$  containing the distinct element  $\delta$  denoting quiescence, and internal actions  $Act_H$ ,
- $E : Loc \rightarrow \mathcal{P}(Edges)$  with  $Edges \stackrel{\text{def}}{=} \mathcal{P}(\mathcal{C}) \times Act \times Distr(T)$  and  $T \stackrel{\text{def}}{=} \mathcal{P}(\mathcal{C}) \times Loc$  is the edge function mapping each location to a finite set of edges, that in turn consist of a guard set, a label, and a distribution over targets in  $T$  consisting of a restart set of clocks and target locations, and
- $F : \mathcal{C} \rightarrow Meas(\mathbb{R}_0^+)$  is the delay measure function that maps each clock to a probability measure.

Intuitively, a stochastic automaton starts its execution in the initial location with all clocks expired. An edge may be taken only if all clocks in its guard set  $G$  are expired. If *any* output edge is enabled, *some* edge must be taken, i.e. all outputs are *urgent*. When an edge  $(\ell, \langle C, a, \mu(R, \ell') \rangle)$  is taken, 1. its action is  $a$ , 2. we select a target  $\langle R, \ell' \rangle \in T$  randomly according to the discrete distribution  $\mu$ , 3. all clocks in  $R$  are restarted and other expired clocks remain expired, and 4. we move to successor location  $\ell'$ . There, another edge may be taken immediately or we may need to wait until some further clocks expire etc. When a clock  $c$  is restarted, the time until it expires is chosen randomly according to the probability measure  $F(c)$ . We point out that each clock has its own distribution, which is *independent* from locations, as well as independent from other clocks.

**Example 7.2.** Figure 7.2a shows an example IOSA specifying the behaviour of a file server with archival storage. We omit empty restart sets and the empty

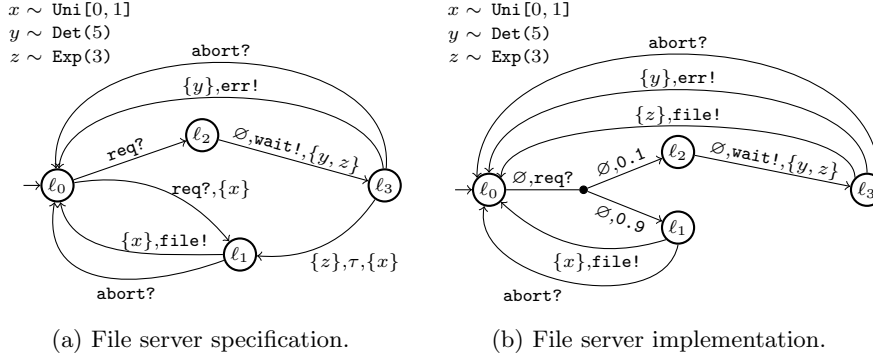


Figure 7.2: Specification and a possible implementation IOSA of a file server with archival storage. Upon receiving a request, a server either directly transfers the file, or needs to fetch it from the archives, resulting in different waiting times. As usual, inputs are suffixed with “?” and outputs with “!”. Note that we omit empty restart sets and the empty guard sets of inputs for readability.

*guard sets of inputs. Upon receiving a request in the initial location  $\ell_0$ , an implementation may either move to  $\ell_1$  or  $\ell_2$ . The latter represents the case of a file in archive: the server must immediately deliver a *wait!* notification and then attempt to retrieve the file from the archive. Clocks  $y$  and  $z$  are restarted, and used to specify that retrieving the file shall take on average  $\frac{1}{3}$  of a time unit, exponentially distributed, but no more than 5 time units. In location  $\ell_3$ , there is a race between retrieving the file and a deterministic time-out. In case of time-out, an error message *err!* is returned; otherwise, the file can be delivered as usual from location  $\ell_1$ . Clock  $x$  is used to specify the transmission time of the file: it is supposed to be uniformly distributed between 0 and 1 time units.*

*In Figure 7.2b, we show an implementation of this specification. One out of ten files randomly requires to be fetched from the archive. This is allowed by the specification: it is one particular resolution of the non-determinism, i.e. underspecification, defined in  $\ell_0$ . The implementation also manages to transmit files from archive directly while fetching them, as evidenced by the direct edge from  $\ell_3$  back to  $\ell_0$  labelled *file!*. This violates the timing prescribed by the specification, and must be detected by an MBT procedure for IOSA.*

**Notation.** By convention, we use the following notations and concepts:

- As usual, we suffix inputs with “?”, and outputs with “!”. Throughout the chapter  $\tau$  denotes an internal action, and  $\delta$  denotes the distinguished quiescence label contained in  $Act_O$ . Further, we use  $\mu$  and  $\nu$  to denote discrete probability distributions used in edges.
- We write  $\ell \xrightarrow{G,a}_E \mu$  if  $\langle G, a, \mu \rangle \in E(\ell)$ .
- A multi-set is written as  $\{\! \{ \dots \} \}$ .

- $Val = V \rightarrow \mathbb{R}_0^+$  is the set of valuations for an (implicit) set  $V$  of (non-negative real-valued) variables.  $\mathbf{0} \in Val$  assigns value zero to all variables.
- Given  $X \subseteq V$  and  $v \in Val$ , we write  $v[X]$  for the *valuation* defined by  $v[X](x) = 0$  if  $x \in X$  and  $v[X](y) = v(y)$  otherwise. This notation is usually used for clock resets. For  $t \in \mathbb{R}_0^+$ ,  $v + t$  is the valuation defined by  $(v + t)(x) = v(x) + t$  for all  $x \in V$ .
- We write  $pdf(c)$  for the probability density function associated to the cumulative distribution function  $F(c)$ .
- We call an IOSA *input-enabled* if all input actions are available in every location at every time, i.e.  $\forall \ell \in Loc, a \in Act_I \exists \mu: \langle \emptyset, a, \mu \rangle \in E(\ell)$

**Parallel composition.** We define parallel composition for IOSAs in order to enable communication among various components, or with the environment. Parallel composition is defined in the standard fashion [9] by synchronizing on shared actions, and evolving independently on others. Since transitions in IOSAs are stochastically independent, we multiply the probabilities when taking shared actions.

**Definition 7.3.** *Given two IOSAs*

$$\mathcal{I} = \langle Loc, \mathcal{C}, Act, E, F, \ell_0 \rangle \quad \text{and} \quad \mathcal{I}' = \langle Loc', \mathcal{C}', Act', E', F', \ell'_0 \rangle$$

with  $\mathcal{C} \cap \mathcal{C}' = \emptyset$ ,  $Act_H \cap Act'_H = \emptyset$ , and  $M \subseteq Act \times Act'$ , their parallel composition is defined as  $\mathcal{I} \parallel \mathcal{I}' \stackrel{\text{def}}{=} \langle Loc'', \mathcal{C}'', Act'', E'', F'', \langle \ell_{0_1}, \ell_{0_2} \rangle \rangle$  where

- $Loc'' = Loc \times Loc'$ ,
- $\mathcal{C}'' = \mathcal{C} \cup \mathcal{C}'$
- $Act'' \stackrel{\text{def}}{=} Act''_I \sqcup Act''_O \sqcup Act''_H$  with outputs  $Act''_O = Act_O \cup Act'_O$ , internal actions  $Act''_H = Act_H \cup Act'_H$ , and inputs  $Act''_I = (Act_I \cup Act'_I) \setminus (\{a_I \in Act_I \mid \exists a_O \in Act'_O: \langle a_I, a_O \rangle \in M\} \cup \{a_I \in Act'_I \mid \exists a_O \in Act_O: \langle a_O, a_I \rangle \in M\})$ ,
- $E''$  is the smallest edge function satisfying the inference rules

$$\frac{\ell_1 \xrightarrow{G, a}_{E_1} \mu \quad a = \tau \vee \nexists a_2 \in Act_2: \langle a, a_2 \rangle \in M}{\langle \ell_1, \ell_2 \rangle \xrightarrow{G, a}_{E''} \{ \langle R, \langle \ell'_1, \ell_2 \rangle \rangle \mapsto \mu(\langle R, \ell'_1 \rangle) \mid R \subseteq \mathcal{C}, \ell'_1 \in Loc_1 \}} \quad (indep_1)$$

$$\frac{\ell_1 \xrightarrow{G_1, a_1}_E \mu \quad \ell_2 \xrightarrow{G_2, a_2}_{E'} \nu \quad a_1 \in Act_O \wedge \langle a_1, a_2 \rangle \in M}{\langle \ell_1, \ell_2 \rangle \xrightarrow{G_1 \cup G_2, a_1}_{E''} \{ \langle R_1 \cup R_2, \langle \ell'_1, \ell'_2 \rangle \rangle \mapsto \mu(\langle R_1, \ell'_1 \rangle) \cdot \nu(\langle R_2, \ell'_2 \rangle) \}} \quad (sync_1)$$

plus symmetric rules  $indep_2$  and  $sync_2$  for the corresponding steps of  $\mathcal{I}_2$ , and lastly

- $F = F \cup F'$ .

We use the convention that two actions  $a_1$  and  $a_2$  match, i.e.  $\langle a_1, a_2 \rangle \in M$ , if they are the same except for the suffix (e.g.  $\mathbf{a}!$  matches  $\mathbf{a}?$  but not  $\mathbf{b}?$  or  $\mathbf{a}!$ ).

The definition of parallel composition for IOSA is straightforward; since clocks are required to be independent from each other, we simply take the union of all clocks, as well as corresponding probability distributions attached to them. The same holds for the action alphabet, except for synchronising actions. Here, only inputs that do not have a corresponding synchronising output in the other component are carried over. Perhaps the most involved item is the edge function; an edge is carried over if its label is  $\tau$ , it does not synchronise, or if both IOSAs enable the same action that is synchronised upon. Note that the latter requires to carry over both guard sets, clock reset sets, and to account for the correct discrete probability over target states by multiplying their independent distributions  $\mu$  and  $\nu$ .

### 7.1.2 Language Theoretic Concepts

IOSAs comprise non-negative real-valued clocks that keep track of the time since their last respective resets. Since there is no memoryless assumption for IOSAs, they comprise *locations*, rather than *states*. Each state  $\langle \ell, v, x \rangle \in S$  consists of the current location  $\ell$  and the values  $v$  and expiration times  $x$  of all clocks. Consequently, the resulting *state space* of an IOSA consists of uncountably many states, but countably many locations as required by Definition 7.1.

**Definition 7.4.** *The states of IOSA  $\mathcal{I}$  are  $S \stackrel{\text{def}}{=} \text{Loc} \times \text{Val} \times \text{Val}$ .*

With the notion of states in place, we are now able to define *paths* induced by an IOSA. In order to do so, we let  $\text{Ex}(G, v, x) \stackrel{\text{def}}{=} \forall c \in G: v(c) \geq x(c)$  be a Boolean that checks whether all clocks in  $G$  are expired.

**Definition 7.5.** *The set of all paths of an IOSA  $\mathcal{I}$  is*

$$\text{paths}(\mathcal{I}) \stackrel{\text{def}}{=} S \times (\mathbb{R}_0^+ \times \text{Edges} \times \mathcal{P}(\mathcal{C}) \times S)^\omega$$

where the first state of each path is  $\langle \ell_0, \mathbf{0}, \mathbf{0} \rangle$ , and contains precisely the sequences  $\pi$  of the form

$$\pi = \langle \ell_0, v_0, x_0 \rangle \langle t_1, e_1, R_1, \langle \ell_1, v_1, x_1 \rangle \rangle \dots,$$

where  $v_0 = x_0 = \mathbf{0}$ , and for all  $i \geq 1$ , we have  $e_i = \langle G_i, a_i, \mu_i \rangle \in E(\ell_{i-1})$ ,  $\text{Ex}(G_i, v_{i-1} + t, x_{i-1})$ ,  $\langle R_i, \ell_i \rangle \in \text{support}(\mu_i)$ ,  $v_i = (v_{i-1} + t)[R_i]$ ,  $x_i \in \{x \in \text{Val} \mid \bigwedge_{c \in \mathcal{C} \setminus R_i} x(c) = x_{i-1}(c) \wedge \bigwedge_{c \in R_i} x(c) \geq 0\}$ , and if  $a_i \notin \text{Act}_I$ , then additionally

$$\nexists t' \in [0, t[: \bigvee_{\ell_{i-1} \xrightarrow{G, a} \mu} \text{Ex}(G, v_{i-1} + t', x_{i-1}).$$

We write  $\text{paths}^{\text{fin}}(\mathcal{I})$  as the set of finite paths. For a finite path  $\pi \in \text{paths}^{\text{fin}}(\mathcal{I})$  we denote  $\text{last}(\pi)$  as its last state, and its length  $|\pi|$  is the number of edges with non-internal action labels.

While Definition 7.5 certainly looks more technically involved compared to previously seen counterparts, it is completely standard. We require that every (finite) path starts in the initial location with all clocks and expiration times set to zero. Every edge occurring on the path needs to be an edge in the IOSA. An edge may only be taken if all its guarded clocks are expired, denoted by the Boolean  $\text{Ex}(G, v, x)$ . Moreover, we require that only valid valuations of clocks and expiration times are recorded in a (finite) path. Lastly, we ensure the urgency of non-input actions, i.e. no other action is enabled before time  $t$ . If presented with a path  $\pi$ , our thorough Definition 7.5 lets us precisely reconstruct an execution of the underlying IOSA. That is, we can track the precise behaviour exhibited by an IOSA up to clock valuations and expiration times in every location. The additional requirements ensure that only valid paths are contained in the eponymous set, and no urgency assumptions or guard constraints are violated.

**Remark 7.6.** *We define states and paths of an IOSA explicitly, by setting states as triples of locations, valuations and expirations. An alternative approach lies in providing the semantics of a stochastic automaton by presenting its underlying timed probabilistic transition system (TPTS). A TPTS treats states as “first class citizens”, and language theoretic concepts can be defined similarly.*

*We opt for the explicit treatment of the semantics of IOSA in this chapter, and make use of TPTS semantics in Chapter 8. In the latter, we require restrictions of the information available in states, and working in TPTS ensures a clean treatment of that.*

Traces represent the behaviour of an automaton visible to an external observer. In particular, they cannot see the values of individual clocks, but only the time passed since the run started. As such, traces are (finite) sequences of time values followed by externally visible actions. Since we intend to quantify the probability of traces, we define abstract traces that comprise time intervals as opposed to single time values. Note that this is a necessity when working in a continuous real-time environment.

**Definition 7.7.** *The trace of a (finite) path  $\pi$  is the overloaded projection*

$$\text{tr}(\pi) : \text{paths}(\mathcal{I}) \rightarrow (\mathbb{R}_0^+ \times \text{Act})^\omega \text{ and } \text{tr}(\pi) : \text{paths}^{\text{fin}}(\mathcal{I}) \rightarrow (\mathbb{R}_0^+ \times \text{Act})^* \text{ resp.}$$

*to the delays in  $\mathbb{R}_0^+$ , and the actions in Act.  $\tau$ -steps are omitted and their delays are added to that of the next visible action. The set of traces of  $\mathcal{I}$  is  $\text{traces}(\mathcal{I})$ , and the set of finite traces is  $\text{traces}^{\text{fin}}(\mathcal{I})$ , respectively.  $\text{traces}^{\text{com}}(\mathcal{I})$  is the set of complete finite traces for  $\mathcal{I}$  based on paths ending in terminal locations.*

*An abstract trace in  $\text{AbsTraces}(\mathcal{I})$  is a sequence  $\Sigma = I_1 a_1 I_2 a_2 \dots$  where  $I_i \subseteq \mathbb{R}_0^+$  are closed intervals. Finite abstract traces are defined analogously, and denoted  $\text{AbsTraces}^{\text{fin}}(\mathcal{I})$ . A finite abstract trace  $\Sigma$  is a prefix of an abstract trace  $\Sigma'$ , denoted  $\Sigma \sqsubseteq \Sigma'$ , if  $a_i = a'_i$  and  $I_i = I'_i$  for all  $i = 1, \dots, n$ .*

As before, we point out, that there is a one-to-one mapping between traces and their abstract counterparts due to our convention to identify time value  $t$  with the interval  $[0, t]$ .

### 7.1.3 Schedulers and Trace Distributions

Input/Output stochastic automata comprise non-deterministic choices, discrete probability distributions, as well as continuously distributed real-valued clocks. Due to the non-determinism in an IOSA, it is not directly possible to assign probabilities to (abstract) paths and traces directly. Rather, we resort to *schedulers* that resolve non-determinism, and consequently yield a purely probabilistic system. This construction was used in previous chapters, too.

Given any finite history leading to a *state*, a scheduler returns a discrete probability distribution over the set of next transitions. In order to model termination, we define schedulers such that they can continue paths with a halting extension  $\perp$ , after which only quiescence is observed. We point out that non-determinism in IOSA always has the form of timed non-determinism, i.e. the choice between multiple transition only arises if the clocks of all guards on transitions expire simultaneously. This includes the empty guard sets on inputs.

**Definition 7.8.** *A scheduler of an IOSA  $\mathcal{I} = \langle Loc, \mathcal{C}, Act, E, F, \ell_0 \rangle$  is a measurable function*

$$\mathcal{A} : \text{paths}^{\text{fin}}(\mathcal{I}) \rightarrow \text{Distr}(\text{Edges} \cup \{\perp\})$$

such that  $\mathcal{A}(\pi)(\langle G, a, \mu \rangle) > 0$  with  $\text{last}(\pi) = \langle \ell, v, x \rangle$  implies  $\ell \xrightarrow{G, a} \mu$  and  $\text{Ex}(G, v + t, x)$  where  $t \in \mathbb{R}_0^+$  is the minimal delay for which no other transition was available before, i.e.

$$\nexists t' \in [0, t] : \bigvee_{\ell \xrightarrow{G', a'} \mu'} \text{Ex}(G, v + t', x).$$

$\mathcal{A}(\pi)(\perp)$  is the probability to halt.  $\mathcal{A}$  halts on  $\pi$  if  $\mathcal{A}(\pi)(\perp) = 1$ .  $\mathcal{A}$  is of length  $k \in \mathbb{N}$  if  $\mathcal{A}(\pi)(\perp) = 1$  for all paths  $\pi$  of length  $\geq k$ . Lastly,  $\text{Sched}(\mathcal{I}, k)$  is the set of all schedulers of  $\mathcal{I}$  of length  $k$ , and  $\text{Sched}(\mathcal{I})$  is the set of all finite schedulers.

A scheduler can only choose between the edges enabled at the points where *any* edge just became enabled in an IOSA. Schedulers remove all non-determinism, and transform an IOSA to a purely probabilistic system. As is usual, we restrict to schedulers that let time diverge with probability 1 (*Zenoness*).

The probability of each step on a path is then given by the *step probability function*:

**Definition 7.9.** *Given IOSA  $\mathcal{I}$  and  $\mathcal{A} \in \text{Sched}(\mathcal{I})$ , the step probability function*

$$Q^{\mathcal{A}} : \text{paths}^{\text{fin}}(\mathcal{I}) \rightarrow \text{Meas}((\mathbb{R}_0^+ \times \text{Edges} \times \mathcal{P}(\mathcal{C}) \times S) \cup \{\perp\})$$

is defined by  $Q^{\mathcal{A}}(\pi)(\perp) = \mathcal{A}(\pi)(\perp)$  and, for  $\pi$  with  $\text{last}(\pi) = \langle \ell, v, x \rangle$ ,  $Q^{\mathcal{A}}(\pi)([t_1, t_2] \times E_Q \times R_Q \times S_Q) =$

$$\mathbb{1}_{t \in [t_1, t_2]} \sum_{e = \langle G, a, \mu \rangle \in E_Q} \left[ \mathcal{A}(\pi)(e) \cdot \sum_{R \in R_Q, \ell' \in Loc} \left( \mu(\langle R, \ell' \rangle) \cdot \int_{\langle \ell', v', x' \rangle \in S_Q} X_R^x(v', x') \right) \right]$$

where  $t$  is the minimal delay in  $\ell$  as in Definition 7.5 and

$$X_R^x(v', x') = \mathbb{1}_{v'=(v+t)[R]} \prod_{c \in R} \begin{cases} 1 & \text{if } c \notin R \wedge x(c) = x'(c) \\ 0 & \text{if } c \notin R \wedge x(c) \neq x'(c) \\ pdf(c)(x'(c)) & \text{if } c \in R. \end{cases}$$

For a given scheduler  $\mathcal{A} \in \text{Sched}(\mathcal{I}, k)$ , the step probability function ensures that the probability of each abstract path is quantified. Note that abstract paths for IOSA consist of time-intervals, and *state intervals*, where each state consists of a location, the current clock valuations, and expiration time valuations. Since the latter two are uncountable, we require intervals to construct the  $\sigma$ -field over abstract paths, as seen in Chapter 5. The step probability function  $Q^{\mathcal{A}}$  induces a unique probability measure  $P_{\mathcal{A}}$  over  $\text{AbsPaths}^{\text{fin}}(\mathcal{I})$ :  $Q^{\mathcal{A}}$  defines, for every path  $\pi$ , a measure quantifying the probability to continue from state  $\text{last}(\pi) = \langle \ell, v, x \rangle$  by incurring a delay in the interval  $I \subseteq \mathbb{R}_0^+$ , taking an edge in  $E_Q$ , resetting a set of clocks in  $R_Q$ , and ending up in a state in  $S_Q$ . Note that we require sets of states  $S_Q$ , since a single state  $\langle \ell, v, x \rangle$  has probability zero to occur.

The probability to halt precisely after  $\pi$  is inferred from the probability a scheduler assigns to the halting extension  $\perp$ . The probability to see an action and the transition to a new location including expiration times and clock valuations is defined as follows: At first, the indicator function ensures that only time values contained in the interval  $[t_1, t_2]$  are accounted for. Next, the probability of all scheduled edges is considered. Here, the discrete probabilities assigned to every respective edge is multiplied with the outcome of their inherent discrete probability distribution  $\mu$ , i.e. the distribution deciding on reset clocks and the next location. Lastly, the Lebesgue integral [178] over all states in  $S_Q$  accounts for clock valuations and expiration times; if a clock is not in the reset set, it contributes with factor 1. On the contrary, it contributes factor 0 if the clock was not reset, and yet new expiration times are sampled – a case that should not occur in practice, that hence sets the probability to 0. Only the presence of a clock  $c$  in the set of reset clocks  $R$  allows for newly sampled expiration times.

The step probability function allows the construction of probability spaces based on maximal abstract paths, as we have seen in Chapter 5. We construct trace distributions based on these probability spaces.

**Trace Distributions.** We can define the *trace distribution* for an IOSA  $\mathcal{I}$  and a scheduler as the probability measure over traces (using abstract traces to construct the corresponding  $\sigma$ -field) induced by the probability measure over paths in the usual way. Then, the probability assigned to a set of abstract traces  $X$  is the probability of all paths whose trace is an element of  $X$ .

**Definition 7.10.** *The trace distribution  $\mathcal{D}$  of a scheduler  $\mathcal{A}$  of IOSA  $\mathcal{I}$ , denoted as  $\mathcal{D} = \text{trd}(\mathcal{A})$ , is the probability space  $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}}, P_{\mathcal{D}})$  given by*

- $\Omega_{\mathcal{D}} = \text{AbsTraces}(\mathcal{I})$ ,
- $\mathcal{F}_{\mathcal{D}}$  is the smallest  $\sigma$ -field generated by the sets  $\{C_{\Sigma} \mid \Sigma \in \text{AbsTraces}^{\text{fin}}(\mathcal{M})\}$ , where  $C_{\Sigma} = \{\Sigma' \in \Omega_{\mathcal{D}} \mid \Sigma \sqsubseteq \Sigma'\}$ , and



- $P_{\mathcal{D}}$  is the unique probability measure on  $\mathcal{F}_{\mathcal{D}}$ , such that

$$P_{\mathcal{D}}(X) = P_{\mathcal{A}}(\text{tr}^{-1}(X)) \text{ for } X \in \mathcal{F}_{\mathcal{D}}.$$

**Trace distribution equivalence.** A trace distribution is of length  $k \in \mathbb{N}$ , if it is based on a scheduler of length  $k$ . The set of all such trace distributions is  $\text{trd}(\mathcal{I}, k)$ . Conversely, the set of all finite trace distributions is denoted  $\text{trd}(\mathcal{I})$ . This induces an equivalence relation  $=_{TD}$ ; two IOSA  $\mathcal{I}$  and  $\mathcal{S}$  are trace distribution equivalent, written  $\mathcal{I} =_{TD} \mathcal{S}$ , if and only if  $\text{trd}(\mathcal{I}) = \text{trd}(\mathcal{S})$ .

## 7.2 Stochastic Testing Theory

Model-based testing entails the automatic test case generation, execution, and evaluation based on a requirements model. Our goal is to establish this three-step procedure for input-output stochastic automata. As a first step, we define what formal conformance between two models means. To that end, we define a conformance relation akin to **ioco** aptly called **sa-ioco**.

In the next step, we formally define test cases, and present how tests are generated based on an IOSA system specification. Their annotations evaluating single traces are based on the previously established conformance relation. The next step involves the execution of these tests on an implementation. Lastly, we demonstrate how samples are evaluated both functionally, and statistically. The functional evaluation is comparable to established procedures from the literature [174], and was described multiple times before in previous chapters. Informally, we require all outputs offered by the SUT to be predictable by the specification. This property is culminated in the **ioco** framework by Tretmans [174], and therefore utilized here. Conversely, similar to how a single coin toss cannot evaluate whether the coin is fair or not, we require a sufficiently large sample to apply statistical tests, that establish probabilistic correctness.

Alongside the functional correctness, we present evaluation procedures for a separate probabilistic verdict. This probabilistic verdict accounts both for implemented probabilities, as well as stochastically distributed time. We define the set of *acceptable outcomes* of an IOSA, which aids us in relating two IOSAs to each other.

### 7.2.1 The Conformance Relation $\sqsubseteq_{ioco}^{sa}$

Trace distribution equivalence  $=_{TD}$  is the probabilistic counterpart of *trace equivalence* for transition systems: it shows that there is a way for the traces of two different models, e.g. the IOSA  $\mathcal{I}$  and  $\mathcal{S}$ , to all have the same probability via *some* resolution of non-determinism. However, trace equivalence or inclusion is too fine as a conformance relation for testing, as studied in Section 3.2. The **ioco** relation [174] for *functional* conformance solves this problem by allowing underspecification of functional behaviour: an implementation  $\mathcal{I}$  is conforming

to a specification  $\mathcal{S}$  if every experiment derived from  $\mathcal{S}$  executed on  $\mathcal{I}$  leads to an output that was foreseen in  $\mathcal{S}$ . Formally:

$$\mathcal{I} \sqsubseteq_{ioco} \mathcal{S} \Leftrightarrow \forall \sigma \in \text{traces}^{fn}(\mathcal{S}): \text{out}_{\mathcal{I}}(\sigma) \subseteq \text{out}_{\mathcal{S}}(\sigma)$$

where  $\text{out}_{\mathcal{I}}(\sigma)$  is the set of outputs in  $\mathcal{I}$  that is enabled after trace  $\sigma$ .

To extend ioco testing to IOSA, we need two auxiliary concepts that mirror trace prefixes, and the set *out* stochastically:

- Given a trace distribution  $\mathcal{D}$  of length  $k$ , and a trace distribution  $\mathcal{D}'$  of length greater or equal to  $k$ , we say  $\mathcal{D}$  is a *prefix* of  $\mathcal{D}'$ , written  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$ , if both assign the same probability to all abstract traces of length  $k$ .
- For an IOSA  $\mathcal{I}$  and a trace distribution  $\mathcal{D}$  of length  $k$ , the output continuation of  $\mathcal{D}$  in  $\mathcal{I}$  contains all trace distributions  $\mathcal{D}'$  of length  $k+1$ , such that  $\mathcal{D} \sqsubseteq_k \mathcal{D}'$ , that assign every trace of length  $k+1$  ending in input probability 0. We set

$$\begin{aligned} \text{outcont}_{\mathcal{I}}(\mathcal{D}) &\stackrel{\text{def}}{=} \{ \mathcal{D}' \in \text{trd}(\mathcal{I}, k+1) \mid \\ &\quad \mathcal{D} \sqsubseteq_k \mathcal{D}' \wedge \forall \sigma \in [\mathbb{R}_0^+ \text{Act}]^k \mathbb{R}_0^+ \text{Act}_{\mathcal{I}} : P_{\mathcal{D}'}(\Sigma) = 0 \}. \end{aligned}$$

**Definition 7.11.** *Let  $\mathcal{S}$  and  $\mathcal{I}$  be two IOSA over the same action signature, and let  $\mathcal{I}$  be input enabled. We say  $\mathcal{I}$  is **sa-ioco-conforming** to  $\mathcal{S}$ , written  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$ , if and only if*

$$\forall k \in \mathbb{N} \forall \mathcal{D} \in \text{trd}(\mathcal{S}) : \text{outcont}_{\mathcal{I}}(\mathcal{D}) \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}).$$

Intuitively,  $\mathcal{I}$  is conforming if, no matter how it resolves non-determinism,  $\mathcal{S}$  can mimic its behaviour by resolving non-determinism accordingly, such that all abstract traces have the same probability. Like the original **ioco**, Definition 7.11 neglects traces ending in input.

Recall Theorem 4.16, and the fact that a stochastic automaton is an input output transition system iff the set of clocks and clock resets is empty, and every discrete probability distribution  $\mu$  is the Dirac distribution. This shows that **sa-ioco** conservatively extends **ioco**.

**Theorem 7.12.** *Let  $\mathcal{I}$  and  $\mathcal{S}$  be two IOTS with  $\mathcal{I}$  input enabled, then*

$$\mathcal{I} \sqsubseteq_{ioco} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}.$$

*Proof sketch.* Since an IOTS is an IOSA without any clocks, guard or reset sets and only Dirac distributions on edges, the notions of schedulers for such IOSAs (Definition 7.8) coincides with the notion of schedulers for pIOTSs only containing Dirac distributions (Definition 4.9). Hence, both definitions induce the same trace distributions on IOTSs. The proof then immediately follows from Theorem 4.16.  $\square$

### 7.2.2 Test Cases

A test case describes the possible behaviour of a tester. The advantage of MBT over manual testing is that test cases can be automatically generated from the specification, and automatically executed on an implementation. In each step of the execution, the tester may either 1. stop testing, 2. wait to observe output, or 3. send an input to the implementation. A single test may provide multiple options, giving rise to multiple concrete testing sequences. It may also prescribe different reactions to different outputs.

Formally, test cases for a specification  $\mathcal{S}$  are IOSA whose inputs are the outputs of  $\mathcal{S}$  and vice-versa. More specifically, we define test cases as IOTSS, i.e. IOSAs with an empty set of clocks and only the Dirac distribution on edges. This ensures that the model of an implementation under test is internally deterministic. A test can also react to *no* output being supplied, modelled by *quiescence*  $\delta$ , and check if that was expected. Since we can only judge the correctness of *specified* behaviour, we limit the outputs a test case can give to those occurring in the specification (cf. *input-minimal tests* [169]).

**Definition 7.13.** *A test case  $t$  over an alphabet  $(Act_I, Act_O)$  is an IOSA  $\langle Loc, \emptyset, Act_I^t \sqcup Act_O^t, E, \emptyset, \ell_0 \rangle$  that has the alphabet's outputs as inputs and vice-versa, i.e.  $Act_I^t = Act_O \cup \{\delta\}$  and  $Act_O^t = Act_I \setminus \{\delta\}$ , and that is a finite, internally deterministic and connected tree. Additionally, we require for every  $\ell \in Loc$ :*

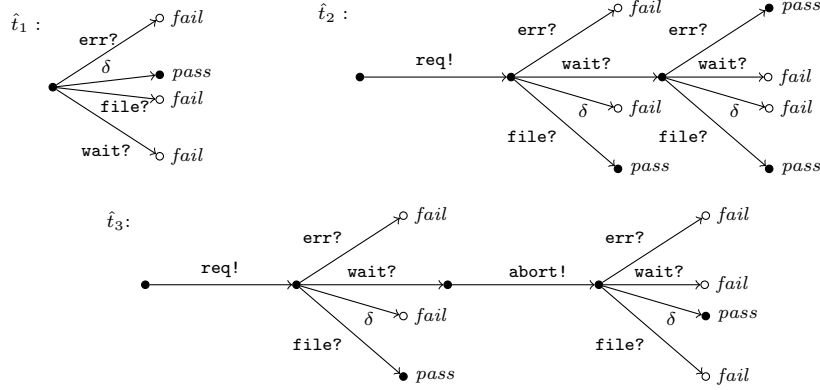
- $E(\ell) = \emptyset$ ,
- For all  $a \in Act_I^t$  there is  $\ell \xrightarrow{\emptyset, a} \mu$  where  $\mu$  the Dirac distribution, or
- There is exactly one  $a \in Act_O^t$ , such that  $\ell \xrightarrow{\emptyset, a} \mu$  where  $\mu$  the Dirac distribution.

*A test suite  $T$  is a set of test cases. A test case (suite resp.) for an IOSA  $\mathcal{S}$ , is a test case (suite resp.) over its action signature, and whenever  $t$  sends an input, this input must be present in  $\mathcal{S}$ , too, i.e.*

$$\forall \sigma \in \text{traces}^{fn}(t) \text{ with } \sigma = \sigma_1 \mathfrak{t} a \sigma_2 \text{ and } a \in Act_I: \sigma_1 \mathfrak{t} a \in \text{traces}^{fn}(\mathcal{S}).$$

Even though test cases are formally defined as IOSAs, their structure is reminiscent of an IOTS. This is due to the fact that test cases do not make use of clocks, or any non-Dirac distributions on their edges. A test case over an action alphabet or specification respectively has a reversed interface: specification inputs are test outputs and vice versa. This is to ensure proper synchronization with respect to parallel composition. To take into account the history, tests are formalized as internally deterministic trees.

**Remark 7.14.** *We point out that allowing non-empty guard sets on input transitions is an appealing approach, as it would allow to model the behaviour of a tester to wait a certain amount of time before supplying inputs. This approach, however, poses some difficulties: The correctness of our framework*

Figure 7.3: Three test cases  $\hat{t}_1, \hat{t}_2, \hat{t}_3$  for the file server specification.

crucially depends on test cases to be internally deterministic. This means that we use test cases according to [173], as opposed to the more recent, input-enabled ones [174]. The latter allow a test to catch output of the implementation, even though it decided to give a stimulus itself. Not having this property means that non-empty guards on test stimuli essentially block output of the implementation. While this also holds for the predecessors of the SA framework, working with clocks which may remain expired due to blocking obfuscates the behaviour of the implementation in undesired ways. We therefore opt to not incorporate non-empty guard sets on stimuli.

**Annotations.** To assess whether observed behaviour is functionally correct, each complete trace of a test is annotated with a verdict: all leaf locations of test cases are labelled with either *pass* or *fail*. We annotate exactly the traces that are present in the specification with the *pass* verdict; formally:

**Definition 7.15.** For a given test  $t$  a test annotation is a function

$$\text{ann} : \text{traces}^{\text{com}}(t) \longrightarrow \{\text{pass}, \text{fail}\}.$$

A pair  $\hat{t} = (t, \text{ann})$  consisting of a test and a test annotation is called an annotated test. A set of such  $\hat{t}$ , denoted by  $\hat{T} = \{(t_i, \text{ann}_i)_{i \in \mathcal{I}}\}$  for some index set  $\mathcal{I}$ , is called an annotated test suite. If  $t$  is a test case for a specification  $\mathcal{S}$  with alphabet  $(\text{Act}_I, \text{Act}_O)$ , we define  $\text{ann}_{\text{saioco}}^{\mathcal{S}} : \text{traces}^{\text{com}}(t) \longrightarrow \{\text{pass}, \text{fail}\}$  by

$$\text{ann}_{\text{saioco}}^{\mathcal{S}} = \begin{cases} \text{fail} & \text{if } \exists \varrho \in \text{traces}^{\text{fin}}(\mathcal{S}), \mathbf{t} \in \mathbb{R}_0^+, a \in \text{Act}_O : \\ & \varrho \mathbf{t} a! \sqsubseteq \sigma \wedge \varrho \mathbf{t} a! \notin \text{traces}^{\text{fin}}(\mathcal{S}) \\ \text{pass} & \text{otherwise.} \end{cases}$$

Annotations decide functional correctness only. The correctness of discrete probability choices and stochastic clocks is assessed in a separate second step.

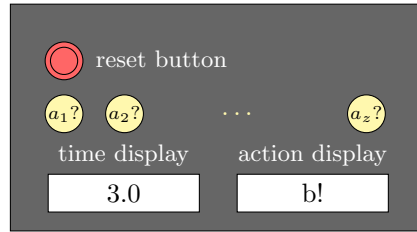


Figure 7.4: Black-box timed trace machine assumed to operate based on an underlying IOSA. The black-box possesses a reset button, and various input buttons, alongside an action window showing the most recently executed action, and a time window showing the relative time since the last observable action.

**Example 7.16.** *Figure 7.3 presents three test cases for the file server specification of Figure 7.2. Test case  $\hat{t}_1$  uses the quiescence observation  $\delta$  to assure no output is given in the initial state. The test  $\hat{t}_2$  checks for eventual delivery of the file, which may be in archive, requiring the intermediate *wait!* notification, or may be sent directly. Lastly,  $\hat{t}_3$  tests the *abort?* transition.*

### 7.2.3 Test Execution and Sampling

We test stochastic systems, hence executing a test case  $t$  once is insufficient to establish **sa-ioco** conformance. We need many executions for a probabilistic verdict about the stochastic behaviour in addition to the functional verdict obtained from the annotation on each execution. As establishing the functional verdict is the same as in standard ioco testing, cf. Chapter 3, we focus on the statistical evaluation here.

**Sampling.** We perform a statistical hypothesis test on the implementation based on the outcome of a push-button experiment in the sense of [134]. Before the experiment, we fix the parameters for sample length  $k \in \mathbb{N}$  (the length of the individual test executions), sample size  $m \in \mathbb{N}$  (how many test executions to observe), and level of significance  $\alpha \in (0, 1)$ . The latter is a limit for the *statistical error of first kind*. The statistical analysis is performed after collecting the sample for the chosen parameters, while functional correctness is checked during the sampling process.

As always, we assume implementations to be black-boxes; all we observe are (timed) traces as in Definition 7.7, and all possible inputs are given by the input alphabet  $Act_I = \{a_0?, \dots, a_n?\}$  of the specification, plus the ability to reset the implementation to its initial state. Such a black-box is depicted in Figure 7.4.

**Frequencies.** Our goal is to determine the deviation of a sample of traces  $O = \{\sigma_1, \dots, \sigma_m\}$  taken from  $\mathcal{I}$  compared to the results expected for the specification  $\mathcal{S}$ . If it is too large,  $O$  was likely not generated by an IOSA conforming to  $\mathcal{S}$  and we reject  $\mathcal{I}$ . If the deviation is within bounds depending

on  $k$ ,  $m$  and  $\alpha$ , we have no evidence to suspect an underlying IOSA other than  $\mathcal{S}$  and accept  $\mathcal{I}$  as a conforming implementation. We compare the frequencies of traces in  $O$  with their probabilities according to  $\mathcal{S}$ .

Since  $\mathcal{I}$  is a concrete implementation, the scheduler is assumed to be the same for all executions, resulting in trace distribution  $\mathcal{D}$  for  $\mathcal{I}$  and the expected probability of abstract trace  $\Sigma$  is given by  $\mathbb{E}^{\mathcal{D}}(\Sigma) = P_{\mathcal{D}}(\Sigma)$ . We define the frequency measure

$$\text{freq}(O)(\Sigma) \stackrel{\text{def}}{=} \frac{|\{\varrho \in O \mid \forall i : t_i^{\varrho} \in I_i^{\Sigma}\}|}{m},$$

i.e. the fraction of traces in  $O$  that are in  $\Sigma$ . Specifically, we require all time stamps to be contained in the intervals given in  $\Sigma$ . Contrary to Chapter 5, this frequency function does *not* assume the independence of clock valuations from locations. This is due to the delay not being *memoryless* as it was for Markov automata.  $\mathcal{I}$  is rejected on statistical evidence if the distance of the measures  $\mathbb{E}^{\mathcal{D}}$  and  $\text{freq}(O)$  exceeds a threshold based on  $\alpha$ . Like before  $\mathbb{E}^{\mathcal{D}}$  denotes the probability measure of abstract traces for a given trace distribution.

**Acceptable outcomes.** We accept a sample  $O$  if  $\text{freq}(O)$  lies within some radius, say  $r_{\alpha}$ , around  $\mathbb{E}^{\mathcal{D}}$ . To minimise the error of false acceptance, we choose the smallest  $r_{\alpha}$  that guarantees that the error of false rejection is not greater than  $\alpha$ , i.e.

$$r_{\alpha} \stackrel{\text{def}}{=} \inf \{ r \in \mathbb{R}^+ \mid P_{\mathcal{D}}(\text{freq}^{-1}(B_r(\mathbb{E}^{\mathcal{D}}))) > 1 - \alpha \}, \quad (7.1)$$

where  $B_y(x)$  is the closed ball centred at  $x \in X$  with radius  $y \in \mathbb{R}^+$  and  $X$  a metric space. The set of all measures defines a metric space together with the total variation distance of measures

$$\text{dist}(u, v) \stackrel{\text{def}}{=} \sup_{\sigma \in (\mathbb{R}_0^+ \times \text{Act})^{\leq k}} |u(\Sigma) - v(\Sigma)|.$$

**Definition 7.17.** For  $k, m \in \mathbb{N}$  and an IOSA  $\mathcal{I}$ , the acceptable outcomes under a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{I}, k)$  of level of significance  $\alpha \in (0, 1)$  are given by the set

$$\text{Obs}(\mathcal{D}, \alpha, k, m) = \{ O \in (\mathbb{R}_0^+ \times \mathcal{A})^{\leq k \times m} \mid \text{dist}(\text{freq}(O), \mathcal{D}) \leq r_{\alpha} \}.$$

The set of acceptable outcomes of  $\mathcal{I}$  with  $\alpha \in (0, 1)$  is then given by

$$\text{Obs}(\mathcal{I}, \alpha, k, m) = \bigcup_{\mathcal{D} \in \text{trd}(\mathcal{I}, k)} \text{Obs}(\mathcal{D}, \alpha, k, m).$$

These sets limit the statistical error of first and second kind as follows: if a sample was generated under a trace distribution of  $\mathcal{I}$  or a trace distribution equivalent IOSA, we accept it with probability higher than  $1 - \alpha$ ; and for all samples generated under a trace distribution by non-equivalent IOSA, the chance of erroneously accepting it is smaller than some  $\beta_m$ , where  $\beta_m$  is unknown but minimal by construction, cf. Equation (7.1). Note that  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ , i.e. the error of accepting an erroneous sample decreases as sample size increases.

**Remark 7.18.** *The set of acceptable outcomes comprises samples of the form  $O \in [\mathbb{R}_0^+ Act]^{\leq k \times m}$ . In order to align observations with the **sa-ioco** relation, we define the set of acceptable output outcomes like follows*

$$OutObs(\mathcal{D}, \alpha, k, m) = \{O \in ([\mathbb{R}_0^+ Act]^{\leq k-1} \mathbb{R}_0^+ Act_O)^m \mid dist(freq(O), \mathbb{E}^{\mathcal{D}}) \leq r_\alpha\}.$$

Test annotations together with the set of *acceptable outcomes* let us define mathematical functions, telling us precisely when an implementation under test passes a test case or test suite respectively.

**Definition 7.19.** *Given an IOSA  $\mathcal{S}$ , an annotated test case  $\hat{t}$ ,  $k$  and  $m \in \mathbb{N}$ , and a level of significance  $\alpha \in (0, 1)$ , we define the functional verdict as given by  $v_{func} : IOSA \times IOSA \rightarrow \{pass, fail\}$  where*

$$v_{func}(\mathcal{I}, \hat{t}) = \begin{cases} pass, & \text{if } \forall \sigma \in traces^{com}(\mathcal{I} \parallel \hat{t}) : ann_{saioco}^{\mathcal{S}}(\sigma) = pass, \\ fail, & \text{otherwise,} \end{cases}$$

and the statistical verdict as given by  $v_{prob} : IOSA \times IOSA \rightarrow \{pass, fail\}$  where

$$v_{prob}(\mathcal{I}, \hat{t}) = \begin{cases} pass, & \text{if } \forall \mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}) \exists \mathcal{D}' \in trd(\mathcal{S}, k) : \\ & P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k, m)) > 1 - \alpha \\ fail, & \text{otherwise,} \end{cases}$$

and lastly the overall verdict  $V(\mathcal{I}, \hat{t}) = pass$  iff  $v_{func}(\mathcal{I}, \hat{t}) = v_{prob}(\mathcal{I}, \hat{t}) = pass$ .  $\mathcal{I}$  passes an annotated test suite  $\hat{T}$ , if  $V(\mathcal{I}, \hat{t}) = pass$  for all annotated tests  $\hat{t} \in \hat{T}$ .

## 7.2.4 Correctness of the Framework

Ideally, only **sa-ioco** correct implementations pass a test suite. However, due to the stochastic nature of IOSA, there remains a degree of uncertainty upon giving verdicts. This is phrased as *errors of first and second kind* in hypothesis testing: the probability to reject a true hypothesis, and to accept a false one, respectively. They are reflected as the probability to reject a correct implementation, or to accept an erroneous one, in the context of probabilistic MBT. The relevance of these errors becomes evident when we consider the correctness of our test framework. Correctness comprises *soundness* and *completeness*: every conforming implementation passes, and there is a test case to expose every non-conforming one. A test suite can only be considered correct with a guaranteed (high) probability  $1 - \alpha$  (as inherent from Definition 7.19).

**Definition 7.20.** *Let  $\mathcal{S}$  be an IOSA over the action signature  $(Act_I, Act_O)$ ,  $\alpha \in (0, 1)$  be level of significance, and  $\hat{T}$  an annotated test suite for  $\mathcal{S}$ . Then*

- $\hat{T}$  is sound for  $\mathcal{S}$  with respect to  $\sqsubseteq_{ioco}^{sa}$  for every  $\alpha \in (0, 1)$ , iff for every input enabled IOSA  $\mathcal{I}$  and every  $\hat{t} \in \hat{T}$  we have that

$$\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = pass.$$

- $\hat{T}$  is complete for  $\mathcal{S}$  with respect to  $\sqsubseteq_{ioco}^{sa}$  for every  $\alpha \in (0, 1)$ , iff for every input-enabled IOSA  $\mathcal{I}$  there exists  $\hat{t} \in \hat{T}$  such that for sufficiently large  $m \in \mathbb{N}$

$$\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S} \implies V(\mathcal{I}, \hat{t}) = fail.$$

For given  $\alpha \in (0, 1)$  soundness expresses that there is a  $1 - \alpha$  chance that a correct system passes an annotated test suite. This relates to false rejection of a correct hypothesis in statistical hypothesis testing, or the rejection of a correct implementation, respectively.

**Theorem 7.21.** *Each annotated test for an IOSA  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{ioco}^{sa}$ .*

*Proof sketch.* For a test case  $\hat{t}$  for  $\mathcal{S}$  and an input enabled IOSA  $\mathcal{I}$  we need to show that  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{t}) = pass$ . By the definition of verdicts (Definition 7.19) this is done in two steps relating to the functional- and the probabilistic verdict. The proof is reminiscent of the ones for pIOTS or IOMA.

- Functional correctness requires that all  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$  have the *pass* annotation. This is shown by considering a prefix  $\sigma'$  of  $\sigma$  in the finite traces of  $\mathcal{S}$ . The intuition is showing that  $\sigma' \tau a$  is a trace in  $\mathcal{S}$  for all outputs  $a$  present after  $\sigma'$  in  $\mathcal{I}$ . There is a trace distribution  $\mathcal{D}$  of  $\mathcal{S}$  that assigns  $\sigma'$  positive probability. Without loss of generality, we choose a trace distribution in  $outcont_{\mathcal{I}}(\mathcal{D})$  that assigns  $\sigma' \tau a$  positive probability. Together with the  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$  assumption, this shows that  $\sigma' \tau a$  gets assigned positive probability in  $\mathcal{S}$  under this trace distribution. Hence  $\sigma' \tau a$  is in the finite traces of  $\mathcal{S}$ . Since this holds for all such prefixes,  $\sigma$  consequently gets assigned the *pass* annotation. This, in turn, yields  $v_{func}(\mathcal{I}, \hat{t}) = pass$ .
- Probabilistic correctness requires that all observations of  $\mathcal{I} \parallel \hat{t}$  get assigned a measure greater or equal to  $1 - \alpha$  for a trace distribution of  $\mathcal{S}$ , i.e. they are acceptable outcomes of  $\mathcal{S}$ . The proof encompasses to choose  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, k)$ , and showing that then also  $\mathcal{D} \in trd(\mathcal{S})$ . This is sufficient by merit of the definition of observations (Definition 7.17), i.e. we always have  $P_{\mathcal{D}}(Obs(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha$  for any  $\mathcal{D}$ . This is done in three steps 1.  $\mathcal{D}$  might still schedule positive probability to input actions in the  $k$ -th step; we choose a new scheduler that assigns all this probability mass to halting instead. Note that the measure of *OutObs* is unaffected by this change, since it comprises traces ending in outputs only. 2. We show that  $\mathcal{D}$  is a trace distribution of  $trd(\mathcal{I})$ . This is guaranteed to hold by construction of test cases (Definition 7.13). In particular, they are constructed in such a way, that  $\mathcal{I} \parallel \hat{t}$  is internally deterministic. A scheduler of  $\mathcal{I}$  can thus directly copy the behaviour of the scheduler for  $\mathcal{I} \parallel \mathcal{I}$ . 3. We apply  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$  to show  $\mathcal{D} \in trd(\mathcal{S})$ . Finally, this yields  $v_{prob}(\mathcal{I}, \hat{t}) = pass$ .

Since both sub-verdicts were shown to yield *pass* we conclude that the overall verdict is *pass* i.e. all tests are sound with respect to  $\sqsubseteq_{ioco}^{sa}$ .  $\square$



Completeness of a test suite is inherently a theoretical result. Infinite behaviour of the implementation, for instance caused by loops, hypothetically requires a test suite of infinite size. Moreover, there remains the possibility of accepting an erroneous implementation by chance, i.e. committing a type II error. However, the latter is bounded from above and decreases with increasing sample size.

**Theorem 7.22.** *The set of all annotated test cases for an IOSA  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{ioco}^{sa}$  for sufficiently large sample size.*

*Proof sketch.* The proof is similar to the ones for pIOTS and IOMA. We assume  $\mathcal{I}$  to be an input enabled IOSA and  $\hat{T}$  to be the test suite containing *all* annotated test cases for  $\mathcal{S}$ . The statement is proven by showing that  $\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = fail$ , i.e. there is a test case  $\hat{t}$  such that either the functional, or the probabilistic verdict fails. Assuming  $\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S}$  implies the existence of a trace ending in output in  $\mathcal{I}$ , such that its probability cannot be matched under any trace distribution in  $\mathcal{S}$ . Generally, this can have two causes: 1. the mismatch arises due to the fact that the trace is simply not present in  $\mathcal{S}$ , or 2. all such traces are present in  $\mathcal{S}$ , and the mismatch arises due to different inherent probability distributions in  $\mathcal{I}$  and  $\mathcal{S}$ . We show that the first implies  $v_{func}(\mathcal{I}, \hat{t}) = fail$ , and the second implies  $v_{prob}(\mathcal{I}, \hat{t}) = fail$  for some test  $\hat{t} \in \hat{T}$ .

- Showing that the functional verdict yields *fail* is straightforward, and requires us to show that there is a test case for which the trace not present in  $\mathcal{S}$  has the *fail* annotation. The proof follows directly from the definition of test cases and test annotations (Definition 7.13 and Definition 7.15), and the fact that  $\hat{T}$  contains *all* test cases for  $\mathcal{S}$ .
- In order to show that the probabilistic verdict yields *fail*, we need to show that there is a test case  $\hat{t}$  and a trace distribution of  $\mathcal{I} \parallel \hat{t}$ , under which all observations get assigned a measure smaller than  $1 - \alpha$  for all trace distributions of  $\mathcal{S}$  for sufficiently large  $m$ . It is clear by the definition of acceptable outcomes (Definition 7.17) that  $P_{\mathcal{D}}(OutObs(\mathcal{D}', \alpha, k, m)) < \beta_m$  for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$  whenever  $\mathcal{D} \neq \mathcal{D}'$ . By the assumption  $\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S}$ , we know this holds for all  $\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)$  with  $\mathcal{D}^* \in trd(\mathcal{S}, k)$ . This estimation does not change, if we increase the search space to all  $\mathcal{D}' \in trd(\mathcal{S}, k + 1)$  instead, as the measure of the *OutObs* set is maximised for trace distributions of *outcont*.

It remains to be shown that for a trace distribution  $\mathcal{D} \in outcont_{\mathcal{I}}(\mathcal{D}^*)$ , there is a test case  $\hat{t} \in \hat{T}$ , such that  $\mathcal{D} \in trd(\mathcal{I} \parallel \hat{t})$ . However, by assumption, we know that all abstract traces getting assigned positive probability under  $\mathcal{D}$  are present in  $\mathcal{S}$ . Thus, we can choose  $\hat{t}$  as the test containing all those traces. In particular, all such traces end in output, which means that the last step of every branch in  $\hat{t}$  requires it to observe the system. By construction of test cases (Definition 7.13)  $\mathcal{I} \parallel \hat{t}$  is internally deterministic. This means a scheduler of  $\mathcal{I} \parallel \hat{t}$  can copy the behaviour of the underlying

scheduler of  $\mathcal{D}$  step-by-step. This shows that  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t})$ , for which all  $\mathcal{D}' \in \text{trd}(\mathcal{S})$  assign all of  $\mathcal{D}$ 's observations a measure smaller than  $1 - \alpha$  for sufficiently large  $m$ . This yields exactly  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$ .

Since the mismatch of probability of an abstract trace has to belong to one of the two cases, we have thus shown that  $V(\mathcal{I}, \hat{T}) = \text{fail}$ .  $\square$

### 7.3 Implementing Stochastic Testing

The previous section laid the theoretical foundations of our IOSA-based testing framework. Several aspects were specified very abstractly, for which we now provide practical procedures. There are several ways to generate, annotate and execute test cases in batch or on-the-fly in the classic **ioco** setting [174], e.g. Chapters 3 and 4. These can be directly transferred to our framework, and are therefore left undiscussed here.

The statistical analysis of gathered sample data in MBT, on the other hand, is largely unexplored since few frameworks include probabilities or even stochastic timing. Determining verdicts according to Definition 7.19 requires concrete procedures to implement the statistical tests described in Subsection 7.2.3 with level of significance  $\alpha$ . We now present practical methods to evaluate test cases in line with this theory. In particular, we discuss how the goodness-of-fit method that was first applied in Chapter 4 is adjusted in order to cope with real-time.

In Chapter 5 we saw the application of simple interval estimations for Markovian parameters. Since IOSA are not limited to exponential distributions anymore, we need more powerful ways to infer if a sample was drawn from a particular distribution. In order to establish a *generally applicable* framework, we resort to non-parametric statistical procedures. In particular, we apply the Kolmogorov-Smirnov test (KS-test), which is capable to infer general probability distributions. Applying various KS-tests alongside Pearson's  $\chi^2$ -test means we are simultaneously performing a multitude of statistical hypothesis tests on the same sample. In order to cope with the inflation of the error of first kind, we apply  $\alpha$  correction. We end the section by giving an algorithmic outline of the model-based testing procedure for IOSA.

#### 7.3.1 Goodness of Fit

We establish practically applicable methods to decide about the verdicts given in Definition 7.19. While the functional verdict is solely based on test annotations, we lack applicable procedures to decide about the probabilistic correctness of an implementation. In Chapter 5 we used Pearson's  $\chi^2$  test alongside a multitude of interval estimations. Working with IOSA means distributions are not limited to the exponential distribution anymore. Since our models neither comprise only *one specific distribution*, nor *one specific parameter* to test for, we resort to non-parametric goodness of fit tests. Non-parametric statistical procedures allow to test hypotheses that were designed for ordinal or nominal data [99], matching our intention of 1. testing the overall distribution of trace frequencies

in a sample  $O = \{\sigma_1, \dots, \sigma_m\}$ , and 2. validating that the observed delays were drawn from the specified clocks and distributions. We use Pearson's  $\chi^2$  test for the first and multiple Kolmogorov-Smirnov tests for the latter.

Like before, our method is based on a theorem known from the literature [40], relating trace distributions on the one hand, to the set of acceptable outcomes on the other hand. However, neither set is readily available to us in case of a real black-box implementation, and only experiments and samples give evidence about its inner workings. Therefore, we pose a null-hypothesis test based on a gathered sample of the implementation. Should the sample turn out to be an acceptable outcome of the specification, too, we accept the hypothesis that all observations of the implementation under test are also observations of the specification. In tandem with the theorem by Cheung, Stoelinga & Vaandrager [40], this implies an embedding on the set of trace distributions. Consequently, the resulting probabilistic verdict in Definition 7.19 is *pass*.

**Pearson's  $\chi^2$  test.** The  $\chi^2$  test compares empirical sample data to its expectations (Appendix A.2). It allows us to check the hypothesis that observed data indeed originates from a specified distribution. The cumulative sum of normalised squared errors is compared to a critical value, and the hypothesis is rejected if the empiric value exceeds the threshold. We can thus check whether trace frequencies correspond to a specification under a certain trace distribution. For a finite trace  $\sigma = t_1 a_1 t_2 a_2 \dots t_k a_k$ , we define its timed closure as  $\bar{\sigma} \stackrel{\text{def}}{=} \mathbb{R}^+ a_1 \dots \mathbb{R}^+ a_k$ . Applying Pearson's  $\chi^2$  in our case results in

$$\chi^2 \stackrel{\text{def}}{=} \sum_{\bar{\sigma} \in \{\bar{\sigma} \mid \sigma \in O\}} \frac{(|\{\bar{\rho} \mid \rho \in O \wedge \bar{\rho} = \bar{\sigma}\}| - m \cdot \mathbb{E}^{\mathcal{D}}(\bar{\sigma}))^2}{m \cdot \mathbb{E}^{\mathcal{D}}(\bar{\sigma})}. \quad (7.2)$$

We need to find a trace distribution  $\mathcal{D} \in \text{trd}(\mathcal{S}, k)$  that gives a high likelihood to a sample, such that  $\chi^2 < \chi_{crit}^2$ , where  $\chi_{crit}^2$  depends on  $\alpha$  and the degrees of freedom. The latter is given by the number of different timed closures in  $O$  minus one, as the probability of one item is set, if we know all other. The critical values can be calculated or found in standard tables (Appendix A.2).

Recall that a trace distribution is based on a scheduler that resolves non-deterministic choices randomly. This turns (7.2) into a satisfaction problem of a probability vector  $p$  over a rational function  $f(p)/g(p)$ , where  $f$  and  $g$  are polynomials. Finding a resolution such that  $\chi^2 < \chi_{crit}^2$  ensures that the error of rejecting a conforming implementation is at most  $\alpha$ . This can be done via SMT solving, or optimisation with feasible constraints for  $p$ , and does account for *trace frequencies*.

**The Kolmogorov-Smirnov test.** While Pearson's  $\chi^2$  test assesses the existence of a scheduler that explains the observed trace frequencies, it does not take into account the observed delays. For this purpose, we use the non-parametric Kolmogorov-Smirnov test (KS-test). We give a short introduction, but refer to Appendix A.2 for further reading.

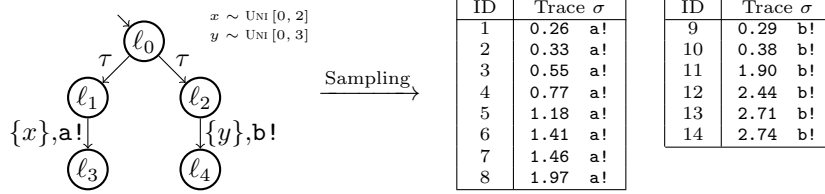


Figure 7.5: Tiny example specification IOSA and sample observation.

The KS-test assesses whether observed data matches a hypothesised *continuous* probability measure. We thus restrict the practical application of our approach to IOSA where the  $F(c)$  for all clocks  $c$  are continuous distributions. Let  $t_1, \dots, t_n$  be the delays observed for a certain edge over multiple traces in ascending order and  $F_n$  be the resulting step function, i.e. the right-continuous function  $F_n$  defined by  $F_n(t) = 0$  for  $t < t_1$ ,  $F_n(t) = n_i/n$  for  $t_i \leq t < t_{i+1}$ , and  $F_n(t) = 1$  for  $t \geq t_n$  where  $n_i$  is the number of  $t_j$  that are smaller or equal to  $t_i$ . Further, let  $c$  be a clock with CDF  $F_c$ . Then the  $n$ -th KS statistic is given by

$$K_n \stackrel{\text{def}}{=} \sup_{t \in \mathbb{R}_0^+} |F_c(t) - F_n(t)|. \quad (7.3)$$

If the sample values  $t_1, \dots, t_n$  are truly drawn from the CDF  $F_c$ , then  $K_n \rightarrow 0$  almost surely as  $n \rightarrow \infty$  by the Glivenko-Cantelli Theorem [77]. Hence, for given  $\alpha$  and sample size  $n$ , we accept the hypothesis that the  $t_i$  were drawn from  $F_c$  iff  $K_n \leq K_{crit}$ , where  $K_{crit}$  is a critical value given by the Kolmogorov distribution. Again, the critical values can be calculated or found in tables (Appendix A.2).

**Example 7.23.** We revisit Example 5.32 first encountered in Chapter 5. There, the example only contained exponential distributions with parameters  $\lambda_1$  and  $\lambda_2$ . Now, the left-hand side of Figure 7.5 shows a tiny example specification IOSA with clocks  $x$  and  $y$ . The expiration times of both are uniformly distributed with different parameters. In  $\ell_0$  there is a non-deterministic choice to either take the left or the right branch.

The right-hand side depicts a sample from this IOSA. There are two steps to assess whether the observed data is a truthful sample of the specification with a confidence of  $\alpha = 0.05$ : 1. find a trace distribution that minimises the  $\chi^2$  statistic, and 2. evaluate two KS tests to assess whether the observed time data is a truthful sample of  $\text{UNI}[0, 2]$  and  $\text{UNI}[0, 3]$ , respectively.

There are two classes of traces solely based on the action signature: ID 1-8 with **a!** and ID 9-14 with **b!**. Let  $p$  be the probability that a scheduler assigns to taking the left branch in  $\ell_0$  and  $1 - p$  that it assigns to taking the right branch. Drawing a sample of size  $m$ , we expect  $p \cdot m$  times **a!** and  $(1 - p) \cdot m$  times **b!**. The empirical  $\chi^2$  value is therefore calculated as

$$\chi^2 = \frac{(8 - 14 \cdot p)^2}{(14 \cdot p)} + \frac{(6 - 14 \cdot (1 - p))^2}{(14 \cdot (1 - p))},$$

which is minimal for  $p = 8/14$ . Since it is smaller than  $\chi_{crit}^2 = 3.84$ , we found a scheduler that maximises the likelihood of the observed frequencies.

As for the second step, let  $t_1 = 0.26, \dots, t_8 = 1.97$  be the data associated with clock  $x$  and  $t'_1 = 0.29, \dots, t'_6 = 2.74$  be the data associated with clock  $y$ . Since there is no time that was recorded twice, observe that the step function of the  $t_i$  for these values is given as

$$F_8(t) = \begin{cases} 0 & t < t_0 \\ \frac{k}{8} & t_k \leq t < t_{k+1}, k = 1, \dots, 7 \\ 1 & t \geq t_8 \end{cases}$$

$D_8 = 0.145$  is the maximal distance between the empirical step function of the  $t_i$  and  $\text{UNI}[0, 2]$ . The critical value of the Kolmogorov distribution for  $n = 8$  and  $\alpha = 0.05$  is  $K_{crit} = 0.46$ . The value was taken from a standard table (Appendix A.3). With  $K_8 < K_{crit}$ , the empiric value is below the given threshold. Hence, the inferred measure is sufficiently close to the specification. The KS test for  $t'_i$  and  $\text{UNI}[0, 3]$  can be performed analogously.

Note that the overall statistical acceptance of the implementation based on the sample data at  $\alpha = 0.05$  requires  $\alpha$ -correction first, to account for the multiple statistical hypothesis tests performed on the same sample data.

Our intention is to provide universally applicable statistical tests. The KS-test is conservative for general distributions, but can be made precise [47]. More specialised and thus efficient tests exist for specific distributions, e.g. the Lilliefors test [99] for Gaussian distributions, and parametric tests are generally preferred due to higher power at equal sample size. The KS-test requires a comparably large sample size, and e.g. the Anderson-Darling test [99] is an alternative.

**Remark 7.24.** *The connection of two non-parametric tests is made immensely more difficult in the presence of internal non-determinism in a specification, cf. Example 7.23 with  $\mathbf{a}! = \mathbf{b}!$ . Time values can no longer be unambiguously addressed to unique distributions, and no confidence bound for the measured time data can be given. In this case, the scheduler probability decisions  $p$  are used as parameters for mixture distributions, e.g.  $F(p) \stackrel{\text{def}}{=} p \cdot F_x + (1 - p) \cdot F_y$  in Figure 7.5. The parametrized distribution can then be used in the iterative expectation-maximization algorithm [135], and confidence can be given upon convergence.*

However, for the sake of simplicity, we assume that the specification is internally deterministic, i.e. there are no two paths that result in the same trace. While this largely decreases the space of potential specifications, we deem it necessary to compromise in order to come up with feasible practical methods.

**Multiple comparisons problem.** A level of significance  $\alpha \in (0, 1)$  limits type 1 error by  $\alpha$ . Performing several statistical experiments inflates this probability: if one experiment is performed at  $\alpha = 0.05$ , there is a 5% probability to incorrectly reject a true hypothesis. Performing 100 experiments, we expect to see a type 1

error 5 times. If all experiments are independent, the chance is thus 99.4%. This is the *family-wise error rate* (FWER). There are two approaches to control the FWER: *single step* and *sequential* adjustments. The most prevalent example for the first is Bonferroni correction, while a prototype of the latter is Holm's method. Both methods aim at limiting the *global* type I error in the statistical testing process. We refer to Appendix A.2 for a more detailed discussion on possibly applicable methods.

For the remainder of this section, we use the straightforward Bonferroni correction, i.e.

$$\alpha_{local} = \frac{\alpha_{global}}{l}$$

where  $l$  is the number of hypothesis tests to be performed.

### 7.3.2 Algorithmic Outline

The overall practical procedure to perform MBT for **sa-ioco** is then as follows:

1. Generate an annotated test case  $t$  of length  $k$  for the specification IOSA  $\mathcal{S}$ .
2. Execute  $t$  on the implementation  $\mathcal{I}$   $m$  times. If the *functional* fail verdict is encountered in any of the  $m$  test executions then fail  $\mathcal{I}$  for functional reasons.
3. Calculate the number of hypothesis tests and adjust  $\alpha$  to avoid error propagation.
4. Use SMT solving to find a scheduler such that the  $\chi^2$  statistic of the sample is below the critical value. If no scheduler is found, fail  $\mathcal{I}$  for *probabilistic* reasons.
5. Group all time stamps assigned to the same clock and perform a KS-test for each clock. If any of them fails, reject  $\mathcal{I}$  for *probabilistic reasons*.
6. Otherwise, accept  $\mathcal{I}$  as conforming to  $\mathcal{S}$  according to  $t$ .

Step 5 has the potential to vastly grow in complexity if traces cannot be uniquely identified in the specification model. Recall Figure 7.5 and assume  $\mathbf{a!} = \mathbf{b!}$ : it is now infeasible to differentiate between time values belonging to the left and the right branch. To avoid this, we have to avoid this scenario at the time of modelling, check *all combinations* of time value assignments, or resort to the iterative expectation-maximization algorithm [135] of parametrized distributions.

## 7.4 Bluetooth Device Discovery Revisited

We revisit the Bluetooth device discovery case study from Chapter 5 with our new IOSA-based test framework. Bluetooth is a wireless communication protocol for low-power devices communicating over short distances. Its devices organise in small networks consisting of one *master* and up to seven *slave* devices. In this initialisation period, Bluetooth uses a frequency hopping scheme to cope with

inferences. To illustrate our framework, we study the initialisation for one master and one slave device. It is inherently stochastic due to the initially random unsynchronised states of the devices. We give a high level overview and refer the reader to [64] for a detailed description and formal analysis of the protocol in a more general scenario. Alternatively, we refer to Chapter 5 for a slightly broader introduction.

**Device discovery protocol.** Master and slave try to connect via 32 prescribed frequencies. Both have a 28-bit clock that ticks every  $312.5\ \mu\text{s}$ . The master broadcasts on two frequencies for two consecutive ticks, followed by a two-tick listening period on the same frequencies, which are selected according to

$$freq = [CLK_{16-12} + off + (CLK_{4-2,0} - CLK_{16-12}) \bmod 16] \bmod 32$$

where  $CLK_{i-j}$  marks the bits  $i, \dots, j$  of the clock and  $off \in \mathbb{N}$  is an offset. The master switches between two *tracks* every 2.56 s. When the 12th bit of the clock changes, i.e. every 1.28 s, a frequency is *swapped* between the tracks. We use  $off = 1$  for track 1 and  $off = 17$  for track 2, i.e. the tracks initially comprise frequencies 1-16 and 17-32. The slave scans the 32 frequencies and is either in sleeping or listening state. The Bluetooth standard leaves some flexibility with respect to the length of the former. For our study, the slave listens for 11.25 ms every 0.64 s and sleeps for the remaining time. It picks the next frequency after 1.28 s, enough for the master to repeatedly cycle through 16 frequencies.

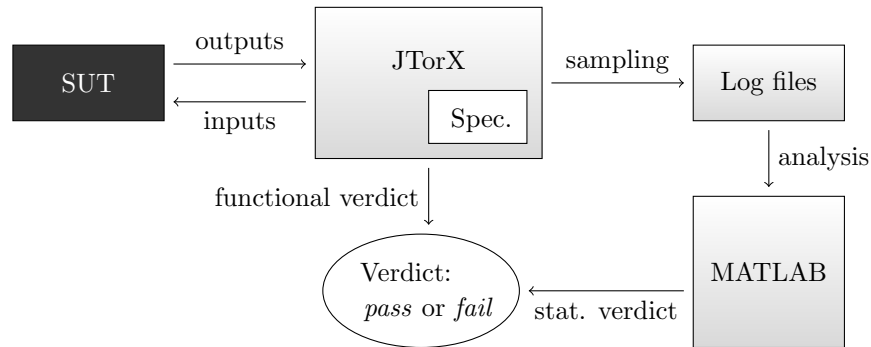


Figure 7.6: Experimental set up entailing the system under test, the MBT tool JTorX [15] and MATLAB [86]. Logs are gathered during the conformance test, and analysed later for a verdict on the implemented probabilities.

**Experimental setup.** Our tool-chain is depicted in Figure 7.6. The implementation is tested on-the-fly via the MBT tool JTorX [15], which generates tests with respect to a transition system abstraction of our IOSA specification modelling the protocol described above. JTorX returns the functional *fail* verdict if unforeseen output or a time-out (quiescence) is observed at any time throughout

the test process. We chose a time-out of approximately 5.2s in accordance with the specification. JTorX's log files comprise the sample, which were evaluated in MATLAB [86]. We implemented the protocol and three mutants in Java 7:

- $\mathcal{M}_1$  Master mutant  $\mathcal{M}_1$  never switches tracks 1 and 2, slowing the coverage of frequencies: new frequencies are only added in the swap every 1.28s.
- $\mathcal{M}_2$  Master mutant  $\mathcal{M}_2$  never swaps frequencies, and only switches between tracks 1 and 2. The expected time to connect will therefore be around 2.56s.
- $\mathcal{S}_1$  Slave mutant  $\mathcal{S}_1$  has its listening period halved: it is only in a receiving state for 5.65 ms every 0.64s.

We expect an increase in waiting time until connection establishment for  $\mathcal{M}_1$  and  $\mathcal{S}_1$ , and slightly lower times for  $\mathcal{M}_2$  when compared to the correct implementation. We anticipate that the increase leads to functional *fail* verdicts due to time-outs or to probabilistic *fail* verdicts based on differing connection time distributions compared to the specification. We collected  $m = 100$ ,  $m = 1000$  and  $m = 10000$  test executions for each implementation, and used  $\alpha = 0.05$ . We used MATLABs [86] `kstest2` to execute a two sample Kolmogorov-Smirnov test to analyse the samples with respect to the correct time distribution.

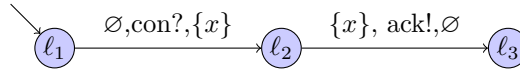


Figure 7.7: Specification of the parallel composition of mutant and slave device. The clock  $x$  is distributed according to the CDF  $F_x$ .

An IOSA specification is presented in Figure 7.7. The clock  $x$  is distributed according to the CDF  $F_x$ . Note that  $F_x$  can be specified as the exponential distribution with parameter  $\lambda = 0.755$  used in Chapter 5, or the exact time distribution given in the Bluetooth specification [161]. This illustrates the modelling power exhibited by IOSA, and shows that the framework of Chapter 5 is subsumed by the one presented here. For the sake of this section, we used the precise time distribution, i.e. the distribution resulting in 32 steps in the step function CDF with respect to the initial locations of both slave and master.

**Results.** Table 7.1 shows the verdicts and the observed KS statistics  $K_m$  alongside the corresponding critical values  $K_{crit}$  for our experiments. The statistical verdict **Accept** was given if  $K_m < K_{crit}$ , and **Reject** otherwise. Note that the critical values depend on the level of significance  $\alpha$  and the sample size  $m$ . The correct implementation was accepted in all three experiments. During the sampling of  $\mathcal{M}_1 \parallel \mathcal{S}$ , we observed several time-outs leading to a functional *fail* verdict. It would also have failed the KS test in all three experiments.  $\mathcal{M}_2 \parallel \mathcal{S}$  passed the test for  $m = 100$ , but was rejected with increased sample size.  $\mathcal{M} \parallel \mathcal{S}_1$



	correct	mutants		
	$\mathcal{M} \parallel \mathcal{S}$	$\mathcal{M}_1 \parallel \mathcal{S}$	$\mathcal{M}_2 \parallel \mathcal{S}$	$\mathcal{M} \parallel \mathcal{S}_1$
$k = 2$	Accept	Reject	Accept	Accept
$m = 100$	$K_m = 0.065$	—	$K_m = 0.110$	$K_m = 0.065$
	$K_{crit} = 0.136$	—	$K_{crit} = 0.136$	$K_{crit} = 0.136$
Timeouts	0	40	0	0
$k = 2$	Accept	Reject	Reject	Accept
$m = 1000$	$K_m = 0.028$	—	$K_m = 0.05$	$K_m = 0.020$
	$K_{crit} = 0.045$	—	$K_{crit} = 0.045$	$K_{crit} = 0.045$
Timeouts	0	399	0	0
$k = 2$	Accept	Reject	Reject	Reject
$m = 10000$	$K_m = 0.006$	—	$K_m = 0.043$	$K_m = 0.0193$
	$K_{crit} = 0.019$	—	$K_{crit} = 0.019$	$K_{crit} = 0.0192$
Timeouts	0	3726	0	0

Table 7.1: Verdicts and Kolmogorov-Smirnov test results for Bluetooth initialisation. An implementation under test passed the KS-test if  $K_m < K_{crit}$

is the most subtle of the three mutants and was only rejected with  $m = 10000$  at a narrow margin.

We point out that no  $\chi^2$  test was necessary due to the structure of the specification. Only one KS-test was required, and consequently no  $\alpha$  correction was performed, as only one statistical hypothesis test was necessary. Lastly, the critical values were taken from a standard KS-test table (Appendix A.2).

**Discussion.** Observe that the critical value decreases faster than the observed KS statistic in all three faulty implementations. We conjecture that an even larger sample is expected to have a clearer verdict, as this is in line with the decrement of the type 2 error for increasing sample size pointed out in Section 7.2. This is especially desirable in the case of  $\mathcal{M} \parallel \mathcal{S}_1$ , where a sample of size  $m = 10000$  was needed to refute the mutant. It is therefore not entirely unrealistic that gathering another sample of the same size would yield a statistical *pass* verdict.

The Bluetooth discovery protocol highlights the advantage of non-parametric hypothesis tests, such as the KS-test. The time distribution to establish a connection does not follow a specific parameter  $\theta$ , and is generally a non-standard distribution unlike e.g. the Gaussian-, exponential-, or uniform distribution. Hence, parametric hypothesis tests are generally not applicable here.

We point out that an alternate specification to the one given in Figure 7.7 is possible. For instance, the entire specification could comprise a probabilistic branching distribution over 32 distinct locations with deterministic guard sets according to the step values of the distribution of the Bluetooth specification. This illustrates the flexibility of the modelling capabilities in the IOSA test framework, and goes to show there is no *unique best* model. Its effectiveness highly depends on the metrics of interest, e.g. is the focus on functional, probabilistic, or stochastically timed behaviour – all three are always covered, but a posteriori analysis might differ in complexity.

## 7.5 Conclusions

We presented an MBT setup based on stochastic automata that combines non-deterministic- and probabilistic choices, as well as continuous stochastic time. The framework marks the pinnacle of this thesis, as far as modelling complexity and power are concerned. Like in previous chapters, we use schedulers and the resulting trace distributions to resolve non-determinism. This yields a purely probabilistic system, and conformance is defined in the **sa-ioco** relation akin to the original **ioco** framework of Tretmans [174]. Test cases were defined as IOTSs (degenerate IOSAs), and annotated to check functional correctness. The statistical verdict requires a sampling process as in earlier chapters.

We instantiated the theoretical framework with a concrete procedure using two statistical tests: Pearson’s  $\chi^2$  to check trace frequencies, and the Kolmogorov-Smirnov test to test clock distributions. The latter is a non-parametric test, allowing to test for arbitrary probability distributions, as opposed to mere confidence interval checks for mean values as seen in Chapter 5. We had to assume some limitations, like internally deterministic models, to end up with a practically feasible algorithm. Lastly, we explored the frameworks’ applicability by revisiting the Bluetooth device discovery protocol. All ingredients of the **sa-ioco** framework are summarized in Table 7.2.

Physical Ingredients:	Formal Ingredients:
<ul style="list-style-type: none"> <li>• Informal requirements</li> <li>• Black-box implementation</li> <li>• Observations: Definition 7.17, <math>Obs(\mathcal{I} \parallel \hat{t}, \alpha, k, m)</math></li> </ul>	<ul style="list-style-type: none"> <li>• Model: Definition 7.1, IOSA</li> <li>• Conformance: Definition 7.11, <math>\sqsubseteq_{ioco}^{sa}</math></li> <li>• Test verdicts: Definition 7.19</li> </ul>
Tooling:	Objectives:
<ul style="list-style-type: none"> <li>• MBT tool: JTorX [15]</li> <li>• Test adapter: Implementable</li> <li>• Test generation method: Random testing</li> </ul>	<ul style="list-style-type: none"> <li>• Soundness: Theorem 7.21</li> <li>• Completeness: Theorem 7.22</li> </ul>
Assumptions:	
<ul style="list-style-type: none"> <li>• Every physical implementation has a corresponding IOSA model</li> <li>• Specification is internally deterministic</li> </ul>	

Table 7.2: The MBT ingredients instantiated by the **sa-ioco** framework.

## 7.6 Proofs

Below we present the proofs for the theorems in this chapter. Reoccurring theorems are numbered according to their occurrence in the chapter.

**Theorem 7.12.** *Let  $\mathcal{I}$  and  $\mathcal{S}$  be two IOTS with  $\mathcal{I}$  input enabled, then*

$$\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S} \iff \mathcal{I} \sqsubseteq_{ioco} \mathcal{S}.$$

*Proof.* Observe that an IOSA  $\mathcal{I} = \langle Loc, \mathcal{C}, Act, E, F, \ell_0 \rangle$  is an IOTS according to Definition 3.2, if  $\mathcal{C} = \emptyset$ ,  $E = Loc \times Act \times Loc$  and  $F = \emptyset$ . Since there are no clocks in IOTSs, there are naturally no guard sets on transitions. Consequently, the definition of schedulers of IOSA (Definition 7.8) coincides with the definition of schedulers for pIOTSs (Definition 4.6), where every probability distribution is the Dirac distribution. Hence, both definitions of schedulers induce the same trace distributions on IOTS. The proof is then an immediate result of Theorem 4.16  $\square$

**Theorem 7.21.** *Each annotated test for an IOSA  $\mathcal{S}$  is sound for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{ioco}^{sa}$ .*

*Proof.* Let  $\mathcal{I}$  be an input enabled IOSA and  $\hat{t}$  be a test for  $\mathcal{S}$ . Further assume that  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$ . Then we want to show  $V(\mathcal{I}, \hat{t}) = pass$ , i.e. we want to show that a **sa-ioco** correct implementation passes an annotated test case. By the definition of verdicts (Definition 7.19) we have  $V(\mathcal{I}, \hat{t}) = pass$  if and only if

$$v_{func}(\mathcal{I}, \hat{t}) = v_{prob}(\mathcal{I}, \hat{t}) = pass.$$

We proceed by showing that  $v_{func}(\mathcal{I}, \hat{t}) = pass$ , and  $v_{prob}(\mathcal{I}, \hat{t}) = pass$  in two separate steps:

1. In order for  $v_{func}(\mathcal{I}, \hat{t}) = pass$ , we need to show that

$$ann_{saioco}^{\mathcal{S}}(\sigma) = pass \text{ for all } \sigma \in traces^{com}(\mathcal{I} \parallel \hat{t}),$$

according to the definition of verdicts (Definition 7.19). Therefore, let  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ . We need to show  $ann_{saioco}^{\mathcal{S}}(\sigma) = pass$  by the definition of annotations (Definition 7.15). Assume  $\sigma' \in traces^{fm}(\mathcal{S})$  and  $a! \in Act_O$  such that  $\sigma' \text{ t } a! \sqsubseteq \sigma$  for some  $\text{t} \in \mathbb{R}_0^+$ .

We observe two things:

- Since  $\varepsilon \in traces^{fm}(\mathcal{S})$ , i.e. the empty trace is a trace and is in  $traces^{fm}(\mathcal{S})$ ,  $\sigma'$  always exists.
- If no such  $a! \in Act_O$  exists, then  $\sigma$  only consists of inputs. By definition of annotations consequently  $ann_{saioco}^{\mathcal{S}}(\sigma) = pass$ .

By construction of  $\sigma$  we have  $\sigma' \uparrow a! \in \text{traces}^{fin}(\mathcal{I} \parallel \hat{t})$  and therefore also  $\sigma' \uparrow a! \in \text{traces}^{fin}(\mathcal{I})$ . Note in particular that the parallel composition with a test case does not alter the guard sets on transitions. We conclude,  $\sigma' \in \text{traces}^{fin}(\mathcal{I}) \cap \text{traces}^{fin}(\mathcal{S})$ . Our goal is to show  $\sigma' \uparrow a! \in \text{traces}^{fin}(\mathcal{S})$ .

Let  $l = |\sigma'|$  be the length of  $\sigma'$ . Without loss of generality, we can now choose  $\mathcal{D} \in \text{trd}(\mathcal{S}, l)$ , such that  $P_{\mathcal{D}}(\Sigma') > 0$ . In particular, we point out that this choice is not invalidated by urgent transitions. If a transition has a guard set, whose clock can never expire in a location due to another urgent output, then this transition is never part of a path (Definition 7.5). Together with the previous observation, this yields  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \neq \emptyset$ . Again, without loss of generality, we choose  $\mathcal{D}' \in \text{outcont}_{\mathcal{I}}(\mathcal{D})$ , such that  $P_{\mathcal{D}'}(\Sigma' [0, \uparrow] a!) > 0$ .

Lastly, we assumed  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$ , hence  $\text{outcont}_{\mathcal{I}}(\mathcal{D}) \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D})$ . We conclude  $\mathcal{D}' \in \text{trd}(\mathcal{S}, l + 1)$ , and  $P_{\mathcal{D}'}(\Sigma' [0, \uparrow] a!) > 0$ . By definition of trace distributions (Definition 7.10), this implies that  $\sigma' \uparrow a! \in \text{traces}^{fin}(\mathcal{S})$ . If additionally  $\sigma' \uparrow a! \in \text{traces}^{com}(\mathcal{I} \parallel \hat{t})$ , then  $\sigma = \sigma' \uparrow a$ . Consequently  $\text{ann}_{saioco}^{\mathcal{S}}(\sigma) = \text{pass}$  by definition of annotations (Definition 7.15). This ultimately yields  $v_{func}(\mathcal{I}, \hat{t}) = \text{pass}$ .

2. In order for  $v_{prob}(\mathcal{I}, \hat{t}) = \text{pass}$  we need to show that

$$\forall \mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k) \exists \mathcal{D}' \in \text{trd}(\mathcal{S}, k) : P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha,$$

according to the definition of verdicts (Definition 7.19). Therefore, let  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$ . By definition of output-observations (Remark 7.18), we have

$$\text{OutObs}(\mathcal{D}, \alpha, k, m) = \{O \in ([\mathbb{R}_0^+ \text{Act}]^{\leq k-1} \mathbb{R}_0^+ \text{Act}_O)^m \mid \text{dist}(\text{freq}(O), \mathbb{E}^{\mathcal{D}}) \leq r_{\alpha}\}.$$

There exists  $\mathcal{D}' \in \text{trd}(\mathcal{I} \parallel \hat{t}, k)$  with

$$P_{\mathcal{D}'}(\Sigma) = \begin{cases} 0 & \text{if } \sigma \in [\mathbb{R}_0^+ \text{Act}]^{k-1} \mathbb{R}_0^+ \text{Act}_I \\ P_{\mathcal{D}}(\Sigma) & \text{if } \sigma \in [\mathbb{R}_0^+ \text{Act}]^{\leq k-1} \mathbb{R}_0^+ \text{Act}_O. \end{cases} \quad (7.4)$$

To see why, consider the scheduler that assigns all probability to halting instead of inputs for traces of length  $k$ , while assigning the same probability to outputs as the scheduler of  $\mathcal{D}$ . By construction of the set  $\text{OutObs}$  (Remark 7.18), observe that

$$\begin{aligned} P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

since only traces ending in output are measured.

It is now sufficient to show that  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k)$ . However, as an intermediate step, we first show that  $\mathcal{D}' \in \text{trd}(\mathcal{I}, k)$ , as this will let us make use of the assumption  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$ .

Consider the mapping  $f$  from the finite paths of  $\mathcal{I} \parallel \hat{t}$  to the finite paths of  $\mathcal{I}$ , i.e.  $f : \text{paths}^{fin}(\mathcal{I} \parallel \hat{t}) \rightarrow \text{paths}^{fin}(\mathcal{I})$ , where for every fragment of the path we have

$$\begin{aligned} & f(\dots \langle (\ell, q), v_0, x_0 \rangle \langle \mathfrak{t}_1, e_1, R_1, |(\ell', q'), v_1, x_1 \rangle \dots) \\ = & \dots \langle \ell, \bar{v}_0, \bar{x}_0 \rangle \langle \bar{\mathfrak{t}}_1, \bar{e}_1, \bar{R}_1 \langle \ell', \bar{v}_1, \bar{x}_1 \rangle \dots \end{aligned}$$

This is possible, because test cases do not contain clocks and parallel composition thus does not change guard sets, reset sets, or expiration times (Definition 7.3). In particular this implies  $v_i = \bar{v}_i$ ,  $x_i = \bar{x}_i$  for  $i = 0, 1$ ,  $\mathfrak{t}_1 = \bar{\mathfrak{t}}_1$  and  $R_1 = \bar{R}_1$ . For  $\bar{e}_1$  consider  $g : E_{\mathcal{I} \parallel \hat{t}} \rightarrow E_{\mathcal{I}}$  such that

$$g(e) = g(C, a, \mu(R, (\ell, q))) = (C, a, \bar{\mu}(R, \ell)) = \bar{e},$$

where  $\mu(R, (\ell, q)) = \bar{\mu}(R, \ell)$  for all  $\ell$ . This construction of  $\mu$  is possible, because tests only contain Dirac distributions, and discrete probabilities thus directly transfer. Hence,  $q$  is uniquely determined by parallel composition (Definition 7.3). Since  $\hat{t}$  is internally deterministic, it is easy to see that  $f$  is an injective mapping, i.e.  $f(\pi_1) = f(\pi_2) \Rightarrow \pi_1 = \pi_2$ .

By definition of trace distributions (Definition 7.10) there is a scheduler, say  $\mathcal{A}' \in \text{Sched}(\mathcal{I} \parallel \hat{t}, k)$ , such that  $\text{trd}(\mathcal{A}') = \mathcal{D}'$ . With the help of  $f$  we show the existence of a scheduler  $\mathcal{A}'' \in \text{Sched}(\mathcal{I})$ , such that for all traces  $\sigma$  we have  $P_{\text{trd}(\mathcal{A}')}(\Sigma) = P_{\text{trd}(\mathcal{A}'')}(\Sigma)$ , i.e.  $\text{trd}(\mathcal{A}'') = \mathcal{D}'$

For every path  $\pi \in \text{paths}^{fin}(\mathcal{I})$  with  $f^{-1}(\pi) \in \text{paths}^{fin}(\mathcal{I} \parallel \hat{t})$ , we define  $\mathcal{A}''$  as

$$\mathcal{A}''(\pi)(\bar{e}) \stackrel{\text{def}}{=} \mathcal{A}'(f^{-1}(\pi))(e)$$

Note that  $P_{\mathcal{A}''}(\Pi) = 0$  if  $\pi \notin \text{paths}^{fin}(\mathcal{I} \parallel \hat{t})$ .

The construction of  $\mathcal{A}''$  is straightforward: Due to the construction of test cases,  $\mathcal{I} \parallel \hat{t}$  is internally deterministic. In particular, there is no interleaving. This means that  $\mathcal{A}''$  can copy the behaviour of  $\mathcal{A}'$  step by step. We set  $\mathcal{D}'' = \text{trd}(\mathcal{A}'')$  and conclude  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, k)$ . By construction it is easy to check that  $P_{\mathcal{D}''}(\Sigma) = P_{\mathcal{D}'}(\Sigma)$  for all traces  $\sigma$ . Further, we have

$$\begin{aligned} P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}'', \alpha, k, m)) &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}', \alpha, k, m)) \\ &= P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}'}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &= P_{\mathcal{D}}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \\ &\geq 1 - \alpha \end{aligned}$$

We proceed to show that  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$ .

The proof is by induction over trace distribution length of prefixes of  $\mathcal{D}''$  up to  $k$ . Trivially, if  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, 0)$ , then also  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, 0)$ . Assume this has been shown for length  $n$ . We proceed by showing that the statement holds for  $n + 1 \leq k$ .

Let  $\mathcal{D}'' \in \text{trd}(\mathcal{I}, n + 1)$  and take  $\mathcal{D}''' \sqsubseteq_n \mathcal{D}''$ . By induction assumption  $\mathcal{D}''' \in \text{trd}(\mathcal{S}, n)$ . Together with the assumption  $\mathcal{I} \sqsubseteq_{ioco}^{sa} \mathcal{S}$ , we have

$$\text{outcont}_{\mathcal{I}}(\mathcal{D}''') \subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}''').$$

Since  $\mathcal{D}'' \in \text{outcont}_{\mathcal{I}}(\mathcal{D}''')$  (Equation (7.4)) we have  $\mathcal{D}'' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}''')$ , and consequently  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, n + 1)$ .

We showed  $\mathcal{D}'' \in \text{trd}(\mathcal{S}, k)$  and conclude

$$P_{\mathcal{D}''}(\text{OutObs}(\mathcal{D}, \alpha, k, m)) \geq 1 - \alpha.$$

Ultimately, this yields  $v_{prob}(\mathcal{I}, \hat{t}) = \text{pass}$  by the definition of verdicts (Definition 7.19).

Both parts together give  $V(\mathcal{I}, \hat{t}) = \text{pass}$ . This means that each annotated test for  $\mathcal{S}$  is sound with respect to  $\sqsubseteq_{ioco}^{sa}$  for every  $\alpha \in (0, 1)$ .  $\square$

**Theorem 7.22.** *The set of all annotated test cases for an IOSA  $\mathcal{S}$  is complete for every level of significance  $\alpha \in (0, 1)$  with respect to  $\sqsubseteq_{ioco}^{sa}$  for sufficiently large sample size.*

*Proof.* In order to show the completeness of test suite  $\hat{T}$  consisting of all annotated tests for  $\mathcal{S}$ , assume that  $\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S}$ . Our goal is to show that  $V(\mathcal{I}, \hat{T}) = \text{fail}$ . By the definition of verdicts (Definition 7.19) this is the case iff  $v_{func}(\mathcal{I}, \hat{t}) = \text{fail}$  or  $v_{prob}(\mathcal{I}, \hat{t}) = \text{fail}$  for some  $\hat{t} \in \hat{T}$ .

Since  $\mathcal{I} \not\sqsubseteq_{ioco}^{sa} \mathcal{S}$ , there is  $k \in \mathbb{N}$ , such that there is  $\mathcal{D}^* \in \text{trd}(\mathcal{S}, k)$ , for which

$$\text{outcont}_{\mathcal{I}}(\mathcal{D}^*) \not\subseteq \text{outcont}_{\mathcal{S}}(\mathcal{D}^*).$$

More specifically

$$\exists \mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*) \forall \mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*) \exists \sigma \in \mathfrak{C} : P_{\mathcal{D}}(\Sigma) \neq P_{\mathcal{D}'}(\Sigma), \quad (7.5)$$

where  $\mathfrak{C} \stackrel{\text{def}}{=} \text{traces}^{fn}(\mathcal{I}) \cap [\mathbb{R}_0^+ \text{Act}]^k \mathbb{R}_0^+ \text{Act}_O$ , and  $\Sigma$  is the corresponding abstract trace of  $\sigma$ . Without loss of generality, we can assume  $k$  to be minimal. There are two cases to consider:

1.  $\exists \sigma \in \mathfrak{C} : \sigma \notin \text{traces}^{fn}(\mathcal{S})$ , or
2.  $\forall \sigma \in \mathfrak{C} : \sigma \in \text{traces}^{fn}(\mathcal{S})$ ,

We will relate the two cases to the functional and the probabilistic verdict (Definition 7.19), respectively. We prove that item 1. implies  $v_{func}(\mathcal{I}, \hat{T}) = \text{fail}$ , and item 2. implies  $v_{prob}(\mathcal{I}, \hat{T}) = \text{fail}$ . Therefore, let  $\mathcal{D} \in \text{outcont}_{\mathcal{I}}(\mathcal{D}^*)$  such that Equation (7.5) holds for all  $\mathcal{D}' \in \text{outcont}_{\mathcal{S}}(\mathcal{D}^*)$ .

1. In order for  $v_{func}(\mathcal{I}, \hat{t}) = fail$ , we need to show

$$\exists \sigma \in traces^{com}(\mathcal{I} \parallel \hat{t}) : ann_{saioco}^S(\sigma) = fail$$

for some  $\hat{t} \in \hat{T}$ , according to the definition of verdicts (Definition 7.19). Assume there is  $\sigma \in \mathfrak{C}$ , such that  $\sigma \notin traces^{fin}(\mathcal{S})$ . Our goal is to show that there is  $\hat{t} \in \hat{T}$  for which  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ , and  $ann_{saioco}^S(\sigma) = fail$ .

Without loss of generality we can assume  $P_{\mathcal{D}}(\Sigma) > 0$ . To see why, assume  $P_{\mathcal{D}}(\Sigma) = 0$ . Then we can find a trace distribution in  $outcont_{\mathcal{S}}(\mathcal{D}^*)$  with an underlying scheduler  $Sched(\mathcal{S})$  that does not assign probability to the last action in  $\sigma$  to obtain probability 0. This violates the assumption that  $P_{\mathcal{D}}(\Sigma) \neq P_{\mathcal{D}'}(\Sigma)$  for all  $\mathcal{D}' \in trd(\mathcal{S})$ . We conclude  $\sigma = \sigma' \mathbf{t} a$ , for some  $\sigma' \in [\mathbb{R}_0^+ Act]^k$ ,  $a \in Act_O$  and  $\mathbf{t} \in \mathbb{R}_0^+$ .

The prefix  $\sigma'$  is in  $traces^{fin}(\mathcal{S})$ , because it is of length  $k$ , and since  $\mathcal{D}^* \in trd(\mathcal{S}, k)$ . Since  $\mathcal{D}$  and all  $\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)$  are continuations of  $\mathcal{D}^*$ , we conclude  $P_{\mathcal{D}^*}(\Sigma') = P_{\mathcal{D}}(\Sigma') = P_{\mathcal{D}'}(\Sigma')$ , i.e. all trace distributions of the respective sets assign every prefix of  $\sigma$  the same probability by merit of *outcont*. We conclude  $\sigma' \in traces^{fin}(\mathcal{S})$ , but  $\sigma' \mathbf{t} a \notin traces^{fin}(\mathcal{S})$ .

By initial assumption  $\hat{T}$  contains *all* annotated test cases. Hence, let  $\hat{t} \in \hat{T}$  such that  $\sigma \in traces^{com}(\hat{t})$ . This is possible because  $\sigma' \in traces^{fin}(\mathcal{S})$ . By the definition of annotations (Definition 7.15) we have  $ann_{saioco}^S(\sigma) = fail$ .

Recall that the set of clocks in test cases is empty. Since  $\sigma \in traces^{fin}(\mathcal{I})$  and  $\sigma \in traces^{com}(\hat{t})$ , we consequently also have  $\sigma \in traces^{com}(\mathcal{I} \parallel \hat{t})$ , as no guard sets or reset sets are changed under parallel composition with a test case. Ultimately, this yields  $v_{func}(\mathcal{I}, \hat{t}) = fail$ .

2. In order for  $v_{prob}(\mathcal{I}, \hat{t}) = fail$ , we need to show

$$\exists \mathcal{D} \in trd(\mathcal{I} \parallel \hat{t}, l) \forall \mathcal{D}' \in trd(\mathcal{S}, l) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, l, m)) < 1 - \alpha,$$

for some  $\hat{t} \in \hat{T}$  and some  $l \in \mathbb{N}$ , according to the definition of verdicts (Definition 7.19).

Together with Equation (7.5) and the definition of acceptable outcomes (Definition 7.17), we conclude

$$\forall \mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) < \beta_m \quad (7.6)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ . Observe that

$$\begin{aligned} & \sup_{\mathcal{D}' \in trd(\mathcal{S}, k+1)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) \\ &= \sup_{\mathcal{D}' \in outcont_{\mathcal{S}}(\mathcal{D}^*)} P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)), \end{aligned} \quad (7.7)$$

by definition of *OutObs* (Remark 7.18). *OutObs* only comprises traces ending in output, thus its measure under any trace distribution of  $trd(\mathcal{S}, k+1)$  cannot be larger than the ones already contained in  $outcont_{\mathcal{S}}(\mathcal{D}^*)$ . Together with Equation (7.6) this yields

$$\forall \mathcal{D}' \in trd(\mathcal{S}, k+1) : P_{\mathcal{D}'}(OutObs(\mathcal{D}, \alpha, k+1, m)) < \beta_m \quad (7.8)$$

for some  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$ . We are left to show that  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k+1)$  for some  $\hat{t} \in \hat{T}$ . Let  $\mathfrak{K} = \{\sigma \in \text{traces}^{\text{fin}}(\mathcal{I}) \mid P_{\mathcal{D}}(\Sigma) > 0\}$ , i.e. all traces getting assigned positive probability under  $\mathcal{D}$ . Obviously  $\mathfrak{C} \subseteq \mathfrak{K}$ . By initial assumption we know that all  $\sigma \in \mathfrak{C}$  are contained in  $\text{traces}^{\text{fin}}(\mathcal{S})$ . Hence all  $\sigma \in \mathfrak{K}$  are necessarily in  $\text{traces}^{\text{fin}}(\mathcal{S})$ . That means, there is a test case  $\hat{t}$  for  $\mathcal{S}$ , such that all  $\sigma \in \mathfrak{K}$  are in  $\text{traces}^{\text{com}}(\hat{t})$ . In particular, observe that all  $\sigma$  end in output by assumption. Hence, the last stage of every test case is the second bullet in the definition of test cases (Definition 7.13). We now construct a scheduler  $\mathcal{A}' \in \text{Sched}(\mathcal{I} \parallel \hat{t}, k+1)$  such that  $\text{trd}(\mathcal{A}') = \mathcal{D}$ . Consider the mapping  $f : \text{tr}^{-1}(\mathfrak{K}) \rightarrow \text{paths}^{\text{fin}}(\mathcal{I} \parallel \hat{t})$ , where for every path fragment

$$\begin{aligned} & f(\dots \langle \ell, v_0, x_0 \rangle \langle \mathfrak{t}_1, e_1, R_1 \langle \ell', v_1, x_1 \rangle \rangle \dots) \\ &= \dots \langle (\ell, q), \bar{v}_0, \bar{x}_0 \rangle \langle \bar{\mathfrak{t}}_1, \bar{e}_1, \bar{R}_1, \langle (\ell', q'), \bar{v}_1, \bar{x}_1 \rangle \rangle \dots \end{aligned}$$

By definition of test cases (Definition 7.13) observe that  $v_i = \bar{v}_i$ ,  $x_i = \bar{x}_i$  for  $i = 0, 1$ ,  $\mathfrak{t}_1 = \bar{\mathfrak{t}}_1$ ,  $R_1 = \bar{R}_1$ , because the clock set of test cases is empty. Further, we define  $g : E_{\mathcal{I}} \rightarrow E_{\mathcal{I} \parallel \hat{t}}$ , such that

$$g(e) = g(C, a, \mu(R, \ell)) = (C, a, \bar{\mu}(R, \ell, q)) = \bar{e},$$

where  $\mu(R, (\ell, k)) = \bar{\mu}(R, \ell)$  for all  $\ell$ . The location  $q$  is uniquely determined, because tests are internally deterministic and every distribution is the Dirac distribution. Thus, discrete probabilities carry over from  $\mu$  to  $\bar{\mu}$ . In particular  $q = q'$  if  $a = \tau$ . It is then easy to see that  $f$  is an injection.

We proceed to construct  $\mathcal{A}'$ . Let  $\mathcal{A} \in \text{trd}(\mathcal{I})$  be the underlying scheduler that induces  $\mathcal{D}$  by definition of trace distributions (Definition 7.10). For every  $\pi \in \text{tr}^{-1}(\mathfrak{K})$  we define

$$\mathcal{A}'(\pi)(\bar{e}) \stackrel{\text{def}}{=} \mathcal{A}(f^{-1}(\pi))(e).$$

The construction of  $\mathcal{A}'$  is straightforward: Since  $\hat{t}$  is internally deterministic, and every of its discrete distributions is the Dirac distribution, there is no interleaving in  $\mathcal{I} \parallel \hat{t}$ . Hence, a scheduler of  $\mathcal{I} \parallel \hat{t}$  may copy the decisions of  $\mathcal{A}$  step-by-step. Note in particular that  $P_{\text{trd}(\mathcal{A}')}(\Sigma) = 0$  for  $\sigma \notin \mathfrak{K}$ . We conclude  $\text{trd}(\mathcal{A}') = \mathcal{D}$  and therefore  $\mathcal{D} \in \text{trd}(\mathcal{I} \parallel \hat{t}, k+1)$ .

Together with Equation (7.7), we have found a scheduler  $\mathcal{A}'$  such that  $\text{trd}(\mathcal{A}') \in \text{trd}(\mathcal{I} \parallel \hat{t}, k+1)$ , and for all  $\mathcal{D}' \in \text{trd}(\mathcal{S}, k+1)$  we have

$$P_{\mathcal{D}'}(\text{OutObs}(\text{trd}(\mathcal{A}'), \alpha, k+1, m)) < \beta_m. \quad (7.9)$$

Now iff  $\alpha \leq 1 - \beta_m$ , we estimate (7.9) further to

$$P_{\mathcal{D}'}(\text{OutObs}(\text{trd}(\mathcal{A}'), \alpha, k+1, m)) < \beta_m \leq 1 - \alpha.$$

However, the inequality  $\alpha \leq 1 - \beta_m$  always holds for sufficiently large  $m$ , since  $\beta_m \rightarrow 0$  as  $m \rightarrow \infty$  by the definition of acceptable outcomes (Definition 7.17). Ultimately, this yields  $v_{\text{prob}}(\mathcal{I}, \hat{t}) = \text{fail}$ .

Together, the two cases yield  $\mathcal{I} \not\sqsubseteq_{\text{icco}}^{\text{sa}} \mathcal{S}$  implies  $V(\mathcal{I}, \hat{T}) = \text{fail}$ .  $\square$



## CHAPTER 8

---

### Scheduler Hierarchy for Stochastic Automata

---

Stochastic automata are a formal compositional model for concurrent stochastic timed systems, with general distributions and non-deterministic choices. In this chapter we investigate the power of various theoretically and practically motivated classes of schedulers, considering the classic complete-information view [30] and a restriction to non-prophetic schedulers [91].

The need to analyse continuous-time stochastic models arises in many practical contexts, including critical infrastructures [8], railway engineering [153], space mission planning [17], and security [94]. Non-determinism arises through inherent concurrency of independent processes [33], but may also be deliberate underspecification. Modelling such uncertainty with probability is convenient for simulation, but not always adequate [7, 117]. Various models and formalisms have thus been proposed to extend continuous-time stochastic processes with non-determinism [22, 31, 67, 89, 93, 167]. It is then possible to *verify* such systems by considering the extremal probabilities of a property. These are the supremum and infimum of the probabilities of the property in the purely stochastic systems induced by classes of *schedulers* that resolve all non-determinism. If the non-determinism is considered controllable, one may alternatively be interested in the *planning* problem of synthesising a scheduler that satisfies certain probability bounds.

Unlike in the previous chapter, we consider closed stochastic automata (SA) [50]. Numerical verification algorithms exist for very limited subclasses of SA only: Buchholz et al. [35] restrict to phase-type or matrix-exponential distributions, such that non-determinism cannot arise (as each edge is guarded by a single clock). Bryans et al. [34] propose two algorithms that require an a priori fixed scheduler, continuous bounded distributions, and that all active clocks be reset when a location is entered. The latter forces regeneration on every edge, making it impossible to use clocks as memory between locations. Regeneration is central to the work of Paolieri et al. [11], however they again exclude non-determinism. The only approach that handles non-determinism is the region-based approximation scheme of Kwiatkowska et al. [119] for a model closely related to SA, but restricted to bounded continuous distributions.

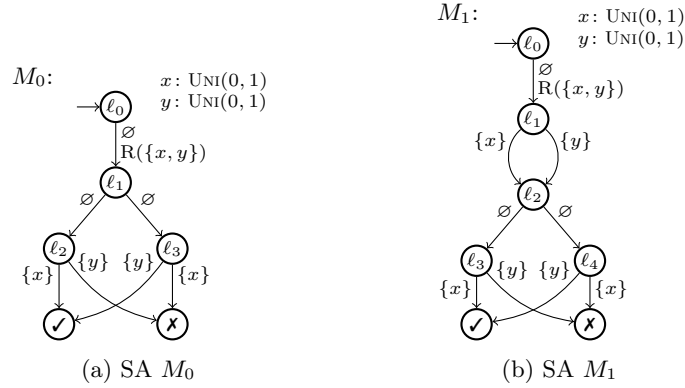


Figure 8.1: Illustration for maximal and minimal unbounded reachability probabilities for schedulers. A scheduler of  $M_0$  that has the information of the expiration times of clocks  $x$  and  $y$  in  $\ell_1$  can make a choice that ensures to reach location  $\checkmark$  with probability 1: go to  $\ell_2$  if  $x$  expires first, and go to  $\ell_3$  if  $y$  expires first. On the same note, a history dependent scheduler of  $M_1$  with access to the information whether the left or right branch was taken in  $\ell_1$ , i.e. which clock already expired, reaches  $\checkmark$  with probability 1. In this chapter, we investigate which information is relevant and separates scheduler classes

Without that restriction [88], error bounds and convergence guarantees are lost.

Evidently, the combination of non-determinism and continuous probability distributions is a challenging one. In this chapter, we take on the underlying problem from a fundamental perspective: we investigate the power of different classes of schedulers for SA as illustrated in Figure 8.1, and prove a hierarchy of scheduler classes with respect to unbounded probabilistic reachability. Our motivation is, on the one hand, that a clear understanding of scheduler classes is crucial to design verification algorithms. For example, Markov decision process (MDP) model checking works well because memoryless schedulers suffice for reachability, and the efficient time-bounded analysis of continuous-time MDP (CTMDP) exploits a relationship between two scheduler classes that are sufficiently simple, but on their own do not realise the desired extremal probabilities [36]. On the other hand, practitioners in planning problems desire *simple* solutions, i.e. schedulers that need little information and limited memory, to be explainable and suitable for implementation on e.g. resource-constrained embedded systems. Understanding the capabilities of scheduler classes helps decide the trade-off between simplicity and the ability to get optimal results.

We use two perspectives on schedulers from the literature: the classic complete-information *residual lifetimes* semantics [30], where optimality is defined via history-dependent schedulers that see the entire current state, and *non-prophetic* schedulers [91] that cannot observe the timing of *future* events. Within each perspective, we define classes of schedulers whose views of the state and history are variously restricted. We prove their relative ordering with respect to achieving

optimal reachability probabilities. We find that SA distinguish most classes. In particular, memoryless schedulers suffice in the complete-information setting (as is implicit in the method of Kwiatkowska et al. [119]), but turn out to be suboptimal in the more realistic non-prophetic case. Considering only the relative order of clock expiration times, as suggested by the first algorithm of Bryans et al. [34], surprisingly leads to partly suboptimal, partly incomparable classes. Our distinguishing SA are small and employ a common non-deterministic gadget. They precisely pinpoint the crucial differences and how schedulers interact with the various features of SA, providing deep insights into the formalism itself.

Our study furthermore forms the basis for the application of *lightweight scheduler sampling* (LSS) to SA. LSS is a technique to use Monte Carlo simulation/statistical model checking with non-deterministic models. On every LSS simulation step, a pseudo-random number generator (PRNG) is re-seeded with a hash of the identifier of the current scheduler and the (restricted) information about the current state (and previous states, for history-dependent schedulers) that the scheduler’s class may observe. The PRNG’s first iteration then determines the scheduler’s action deterministically. LSS has been successfully applied to MDP [52, 125, 126] and probabilistic timed automata [49, 92]. Using only constant memory, LSS samples uniformly from a selected scheduler class to find “near-optimal” schedulers that conservatively approximate the true extremal probabilities. Its principal advantage is that it is largely indifferent to the size of the state space and of the scheduler space – finding near-optimal schedulers for huge state spaces may be very fast, while finding them in smaller ones may take sufficiently more time. In general, sampling efficiency depends only on the likelihood of selecting near-optimal schedulers. However, the mass of *near-optimal* schedulers in a scheduler class that also includes the optimal scheduler may be *less* than the mass in a class that does *not* include it. Given that the mass of optimal schedulers may be vanishingly small, it may be advantageous to sample from a class of less powerful schedulers to achieve better results *on average*.

We summarize the main contributions of this chapter

- a hierarchy of scheduler classes of closed SA with respect to unbounded probabilistic reachability,
- a distinction of *classic-* and *non-prophetic* schedulers therein, and
- experiments on lightweight scheduler sampling to explore the trade-off between powerful and efficient scheduler classes, and confirmation for our theoretical results.

While the work within this chapter is not related to model-based testing *per se*, we encountered schedulers in previous chapters. Even though we only considered history dependent randomised schedulers, the restriction to more refined scheduler classes is widely acknowledged to be of major importance [18, 66]. We deem a hierarchical classification of schedulers imperative for future research related to the topic of this thesis, especially when considering stochastic automata as extension to all modelling formalisms contained therein.

**Related work.** Brázdil et al. [33] show that non-determinism (“unstable behaviour”) is introduced even in generalised semi-Markov processes (GSMP) as soon as there is more than one fixed-delay event, highlighting the importance of properly treating non-determinism in any analysis approach. Alur et al. first mention non-deterministic stochastic systems similar to SA in [5]. Markov automata (MA [67]), interactive Markov chains (IMC [93]) and CTMDP are special cases of SA restricted to exponential distributions. Song et al. [162] look into partial information distributed schedulers for MA, combining earlier works of De Alfaro [54], and Giro and D’Argenio [79] for MDP. Their focus is on information flow and hiding in parallel specifications. Wolf et al. [196] investigate the power of classic (time-abstract, deterministic and memoryless) scheduler classes for IMC. They establish (non-strict) subset relationships for almost all classes with respect to trace distribution equivalence – a very strong measure. Wolovick and Johr [198] show that the class of measurable schedulers for CTMDP is complete and sufficient for reachability problems.

**Origins of the chapter.** This work is the result of a collaboration with Pedro D’Argenio, Arnd Hartmanns, and Sean Sedwards that appeared in

- Pedro R. D’Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. A hierarchy of scheduler classes for stochastic automata. In *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FOSSACS*, pages 384–402, 2018.

**Organisation of the chapter.** Section 8.1 lays the foundations in defining closed stochastic automata, and their underlying semantics in timed probabilistic transition systems. We define various scheduler classes motivated by the literature in Section 8.2. These include classical notions of schedulers, as well as non-prophetic ones. Consequently, a hierarchy among the classes is established for both notions of schedulers in Section 8.3. We validate the theoretical observations by using lightweight scheduler sampling in Section 8.4. Section 8.5 concludes with a small discussion, and final remarks.

## 8.1 Preliminaries

We define stochastic automata according to [50]. As opposed to the previous chapter, we now treat SA as *closed systems*. In particular, there are no delayable actions, and all actions are urgent, i.e. as soon as *any* edge becomes enabled, *some* edge has to be taken. Moreover, in contrast to Chapter 7, we give the semantics of an SA via their underlying timed probabilistic transition systems (TPTS). This allows us to store the valuation, as well as expiration times of clocks in every location explicitly as *states* of the TPTS. Lastly, rather than defining schedulers for SA directly, we define them for their underlying TPTSs based on *runs*/paths.

### 8.1.1 Closed Stochastic Automata

The authors of [50] extend labelled transition systems with stochastic *clocks*: real-valued variables that increase synchronously with rate 1 over time and expire some random amount of time after having been *restarted*. As opposed to the previous chapter, we define *closed* stochastic automata.

**Definition 8.1.** A stochastic automaton (SA) is a tuple  $\langle Loc, \mathcal{C}, Act, E, F, \ell_0 \rangle$ , where

- $Loc$  is a countable set of locations, with  $\ell_0 \in Loc$  as the initial location,
- $\mathcal{C}$  is a finite set of clocks,
- $Act$  is the finite action alphabet, and
- $E: Loc \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{C}) \times Act \times \mathcal{P}(\mathcal{C}) \times \text{Dist}(Loc))$  is the edge function, which maps each location to a finite set of edges that in turn consist of a guard set of clocks, a label, a restart set of clocks and a discrete distribution over target locations, and
- $F: \mathcal{C} \rightarrow \text{Meas}(\mathbb{R}_0^+)$  is the delay measure function that maps each clock to a probability measure.

We write  $\ell \xrightarrow{G,a,R}_E \mu$  for  $\langle G, a, R, \mu \rangle \in E(\ell)$ . Without loss of generality, we restrict to SA where edges are fully characterised by source and action label, i.e. whenever  $\ell \xrightarrow{G_1,a,R_1}_E \mu_1$  and  $\ell \xrightarrow{G_2,a,R_2}_E \mu_2$ , then  $G_1 = G_2$ ,  $R_1 = R_2$  and  $\mu_1 = \mu_2$ . This is a technical restriction to improve the presentation. In all examples we thus assume each edge of a location to have a unique label.

Intuitively, an SA starts in  $\ell_0$  with all clocks expired. An edge  $\ell \xrightarrow{G,a,R}_E \mu$  may be taken only if all clocks in  $G$  are expired. If any edge is enabled, some edge must be taken (i.e. all actions are *urgent* and thus the SA is *closed*). When an edge is taken, its action is  $a$ , all clocks in  $R$  are restarted, other expired clocks remain expired, and we move to successor location  $\ell'$  with probability  $\mu(\ell')$ . There, another edge may be taken immediately or we may need to wait until some further clocks expire etc. When a clock  $c$  is restarted, the time until it expires is chosen randomly according to the probability measure  $F(c)$ . We point out that  $F$  is defined independently from the current location, i.e. clock probability measures are global. Formally, a clock counts up and expires as soon as it hits its expiration time.

**Example 8.2.** We show an example SA,  $M_0$ , in Figure 8.2a. Its initial location is  $\ell_0$ . It has two clocks,  $x$  and  $y$ , with  $F(x)$  and  $F(y)$  both being the continuous uniform distribution over the interval  $[0, 1]$ . No time can pass in locations  $\ell_0$  and  $\ell_1$ , since they have outgoing edges with empty guard sets. We omit action labels and assume every edge to have a unique label. On entering  $\ell_1$ , both clocks are restarted. The choice of going to either  $\ell_2$  or  $\ell_3$  from  $\ell_1$  is non-deterministic, since the two edges are always enabled at the same time. In  $\ell_2$ , we have to wait until the first of the two clocks expires. If that is  $x$ , we have to move to location  $\checkmark$ ; if it is  $y$ , we have to move to  $\times$ . The probability that both expire at the exact

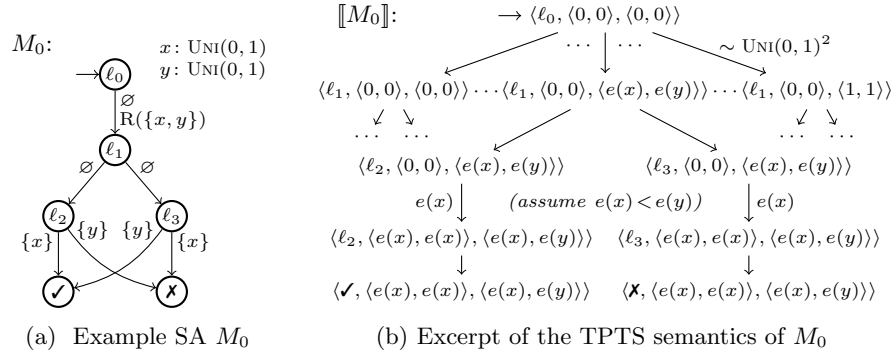


Figure 8.2: Example SA  $M_0$  and corresponding excerpt of its TPTS semantics. The use of clocks results in uncountably many *states*, i.e. locations, clock valuations and expiration times, as opposed to the countably many *locations*.

same time is zero. Location  $\ell_3$  behaves analogously, but with the target states interchanged.

**Remark 8.3.** For compositional modelling, SA can be equipped with a standard parallel composition operator that synchronises edges with matching action labels as seen in Definition 7.3. The guard set of a synchronising edge is the union of the guard sets of the component edges (i.e. actions are patient). In this chapter, we assume that all parallel composition has been unrolled, i.e. we work with a single SA and focus on closed systems. In open systems, some actions could additionally be delayable: When an edge with a delayable action becomes enabled, more time may pass before it or other edges are taken. Markov automata, and thus MDP, are special cases of SA: An SA is an MA if  $F(c)$  is an exponential distribution for all clocks  $c$ , every guard set is either empty (an immediate edge) or singleton (a Markovian edge), and all clocks are restarted on all edges.

**Remark 8.4.** We point out the difference of SA to timed automata (TA) [6]: While clock resets on edges are defined in a similar manner, guard sets behave differently. The guard sets on edges TA generally consists of an a priori fixed time interval. An edge may be taken iff the corresponding clock valuation remains in this time interval. Naïvely, guard sets on edges in an SA are unions of intervals of the form  $[x, \infty)$ , where  $x$  is a realisation of a random variable distributed according to  $F(c)$ . Note that each clock  $c$  has its own  $F(c)$ . An edge must be taken if all clocks of a single guard set of an outgoing edge reach their respective intervals.

### 8.1.2 Timed Probabilistic Transition Systems

Timed probabilistic transition systems (TPTS) form the semantics of SA. We define TPTS with uncountable state and action spaces. This is needed since we

need to store the current valuation of real-valued clocks and expirations in each state. They are finitely-non-deterministic uncountable-state transition systems.

**Definition 8.5.** A (finitely non-deterministic) timed probabilistic transition system (TPTS) is a tuple  $\langle S, Act', T, s_0 \rangle$ , where

- $S$  is a measurable set of states, with the unique initial state  $s_0 \in S$ ,
- $Act' = \mathbb{R}^+ \sqcup Act$  is the alphabet, partitioned into delays in  $\mathbb{R}^+$  and jumps in  $Act$ .
- $T: S \rightarrow \mathcal{P}(Act' \times Meas(S))$  is the transition function, which maps each state to a finite set of transitions, each consisting of a label in  $Act'$  and a measure over target states.

All  $s \in S$  admitting delays are required to be deterministic, i.e.  $|T(s)| = 1$  if there is  $\langle t, \mu \rangle \in T(s)$  such that  $t \in \mathbb{R}^+$ .

The requirement for states admitting delay to be deterministic ensures that a delay cannot be split into two or more sub-delays. We write  $s \xrightarrow{a}_T \mu$  for  $\langle a, \mu \rangle \in T(s)$ . A run is an infinite alternating sequence  $s_0 a_0 s_1 a_1 \dots \in (S \times Act')^\omega$ . A history is a finite prefix of a run ending in a state, i.e. an element of  $(S \times Act')^* \times S$ . We write  $last(h)$  to denote the last state of a run  $h$ . Runs resolve all non-deterministic and probabilistic choices. A scheduler resolves only the non-determinism:

**Definition 8.6.** A scheduler is a measurable function

$$\mathfrak{s}: (S \times Act')^* \times S \rightarrow dist(Act' \times Meas(S))$$

such that for all histories  $h \in (S \times Act')^* \times S$ ,  $\langle a, \mu \rangle \in \text{support}(\mathfrak{s}(h))$  implies  $last(h) \xrightarrow{a}_T \mu$ .

Once a scheduler has chosen  $s_i \xrightarrow{a}_T \mu$ , the successor state  $s_{i+1}$  is picked randomly according to  $\mu$ . Every scheduler  $\mathfrak{s}$  defines a probability measure  $\mathbb{P}_\mathfrak{s}$  on the space of all runs. For a formal definition, see [197]. As is usual, we restrict to *non-Zeno* schedulers that make time diverge with probability one: we require  $\mathbb{P}_\mathfrak{s}(\Pi_\infty) = 1$ , where  $\Pi_\infty$  is the set of runs where the sum of delays is  $\infty$ . This prevents performing infinite actions in a finite amount of time.

### 8.1.3 Semantics of Closed Stochastic Automata

We present the residual lifetimes semantics of [30], simplified for closed SA. A closed SA induces a TPTS, if we require that any delay step must be the minimum delay that enables some edge. First, we require some notation.

**Notation.** Let  $Val: V \rightarrow \mathbb{R}_0^+$  be the valuations for a set  $V$  of (non-negative real-valued) variables.  $\mathbf{0} \in Val$  assigns value zero to all variables. For  $X \subseteq V$  and  $v \in Val$ , we write  $v[X]$  for the valuation defined by  $v[X](x) = 0$  if  $x \in X$  and  $v[X](y) = v(y)$  otherwise. This notation is mainly used for clock resets. We

write  $Val$  if  $V$  is clear from the context. For  $t \in \mathbb{R}_0^+$ ,  $v + t$  is defined by  $(v + t)(x) = v(x) + t$  for all  $x \in V$ . We define  $\text{Ex}(G, v, e) \stackrel{\text{def}}{=} \forall x \in G: v(x) \geq e(x)$  to be the Boolean that characterises enabled edges, i.e. the value  $\text{Ex}(G, v, e)$  is *true* iff all clocks  $x$  of a guard set  $G$  have a valuation  $v(x)$  that is greater or equal to their expiration time  $e(x)$ . Given probability measures  $\mu_1$  and  $\mu_2$ , we denote by  $\mu_1 \otimes \mu_2$  the *product measure* (cf. Appendix A.1). For a collection of measures  $(\mu_i)_{i \in I}$ , we analogously denote the product measure by  $\bigotimes_{i \in I} \mu_i$ .

**Definition 8.7.** *The semantics of an SA  $M = \langle \text{Loc}, \mathcal{C}, \text{Act}, E, F, \ell_0 \rangle$  is the timed probabilistic transition system (TPTS)*

$$\llbracket M \rrbracket = \langle \text{Loc} \times \text{Val} \times \text{Val}, \text{Act} \sqcup \mathbb{R}^+, T_M, \langle \ell_0, \mathbf{0}, \mathbf{0} \rangle \rangle$$

where the states are triples  $\langle \ell, v, e \rangle$  of the current location  $\ell$ , a valuation  $v$  assigning to each clock its current value, and a valuation  $e$  keeping track of all clocks' expiration times.  $T_M$  is the smallest transition function satisfying the inference rules

$$\frac{\ell \xrightarrow{G, a, R}_E \mu \quad \text{Ex}(G, v, e)}{\langle \ell, v, e \rangle \xrightarrow{a}_{T_M} \mu \otimes \text{Dirac}(v[R]) \otimes \text{Sample}_e^R}$$

$$\frac{t \in \mathbb{R}^+ \quad \exists \ell \xrightarrow{G, a, R}_E \mu: \text{Ex}(G, v + t, e) \quad \forall t' \in [0, t), \ell \xrightarrow{G, a, R}_E \mu: \neg \text{Ex}(G, v + t', e)}{\langle \ell, v, e \rangle \xrightarrow{t}_{T_M} \text{Dirac}(\langle \ell, v + t, e \rangle)}$$

where  $\langle G, a, R, \mu \rangle \in E(\ell)$ , i.e.  $G \in \mathcal{P}(\mathcal{C})$  is a guard set,  $a \in \text{Act}$  is an action,  $R \in \mathcal{P}(\mathcal{C})$  is the set of reset clocks and  $\mu$  is the discrete distribution over target locations. Further

$$\text{Sample}_e^R \stackrel{\text{def}}{=} \bigotimes_{c \in \mathcal{C}} \begin{cases} F(c) & \text{if } c \in R \\ \text{Dirac}(e(c)) & \text{if } c \notin R. \end{cases}$$

The second rule creates *delay* steps of  $t$  time units if no edge is enabled from now until just before  $t$  time units have elapsed (third premise) but then, after exactly  $t$  time units, some edge becomes enabled (second premise). The first rule applies if an edge  $\ell \xrightarrow{G, a, R}_E \mu$  is enabled: a transition is taken with the edge's label, the successor state's location is chosen by the discrete distribution  $\mu$ ,  $v$  is updated by resetting the clocks in the reset set  $R$  to zero, and the expiration times for the reset clocks  $c \in R$  are re-sampled according to their distribution  $F(c)$ . All other expiration times remain unchanged.

**Example 8.8.** *Figure 8.2b outlines the semantics of  $M_0$ . The first step from  $\ell_0$  to all the states in  $\ell_1$  is a single transition. Its probability measure is the product of  $F(x)$  and  $F(y)$ , sampling the expiration times of the two clocks. We exemplify the behaviour of all of these states by showing it for the case of expiration times  $e(x)$  and  $e(y)$ , with  $e(x) < e(y)$ .*



## 8.2 Classes of Schedulers

We now define several classes of schedulers that are restricted in the information available to them to make their choices. This includes the standard restriction to memoryless schedulers, and schedulers that only see some part of the states, hiding clock values and expiration times to various degrees. All definitions consider TPTS as in Definition 8.5 with states  $\langle \ell, v, e \rangle$ , and we require for all schedulers  $\mathfrak{s}$  that only available transitions are scheduled, i.e.  $\langle a, \mu \rangle \in \text{support}(\mathfrak{s}(h))$  implies  $\text{last}(h) \xrightarrow{a}_T \mu$ , as in Definition 8.6.

### 8.2.1 Classic Schedulers

We first consider the “classic” complete-information setting where schedulers can in particular see expiration times [30]. Recall the definition of schedulers in Definition 8.6. This scheduler class is history dependent, and has access to both clock valuations and clock expiration times. We denote this set by  $\mathfrak{S}_{\ell, v, e}^{\text{hist}}$ .

We proceed by restricting the information accessible by history-dependent schedulers. We generally denote the information available to a scheduler by subscripts on the set of states, e.g.  $S|_{\ell, v, e} \stackrel{\text{def}}{=} \text{Loc} \times \text{Val} \times \text{Val}$  denotes that a scheduler has access to the current location, clock valuation, and clock expiration times. Our first restriction hides the *values* of all clocks, only revealing the total time since the start of the history. This is inspired by the step-counting or time-tracking schedulers needed to obtain optimal step-bounded or time-bounded reachability probabilities on MDP or Markov automata:

**Definition 8.9.** *A classic history-dependent global-time scheduler is a measurable function*

$$\mathfrak{s}: (S|_{\ell, t, e} \times \text{Act}')^* \times S|_{\ell, t, e} \rightarrow \text{dist}(\text{Act}' \times \text{Meas}(S)),$$

where  $S|_{\ell, t, e} \stackrel{\text{def}}{=} \text{Loc} \times \mathbb{R}_0^+ \times \text{Val}$  with the second component being the total time  $t$  elapsed since the start of the history to the corresponding state, and the third component being the clock expiration times  $e$ . We write  $\mathfrak{S}_{\ell, t, e}^{\text{hist}}$  for the set of all such schedulers.

We next hide the values of all clocks, revealing only their expiration times. This class is considered mainly for the sake of completion of the hierarchy.

**Definition 8.10.** *A classic history-dependent location-based scheduler is a measurable function*

$$\mathfrak{s}: (S|_{\ell, e} \times \text{Act}')^* \times S|_{\ell, e} \rightarrow \text{dist}(\text{Act}' \times \text{Meas}(S)),$$

where  $S|_{\ell, e} \stackrel{\text{def}}{=} \text{Loc} \times \text{Val}$ , with the second component being the clock expiration times  $e$ . We write  $\mathfrak{S}_{\ell, e}^{\text{hist}}$  for the set of all such schedulers.

Having defined three classes of classic history-dependent schedulers,  $\mathfrak{S}_{\ell, v, e}^{\text{hist}}$ ,  $\mathfrak{S}_{\ell, t, e}^{\text{hist}}$  and  $\mathfrak{S}_{\ell, e}^{\text{hist}}$ , we also consider them with the restriction that they only see

the relative *order* of clock expiration, instead of the exact expiration times: for each pair of clocks  $c_1, c_2$ , these schedulers see the relation  $\sim \in \{<, =, >\}$  in  $e(c_1) - v(c_1) \sim e(c_2) - v(c_2)$ . Knowing the expiration order is incomparable to knowing *only* the expiration times  $e$ : The former is determined by the difference  $e(c) - v(c)$  for each clock  $c$ , while current clock *valuations* are unknown in the latter. Since the relation is defined for each pair of clocks, it induces a total order. To illustrate, in  $\ell_1$  of Example 8.8, the scheduler would not see  $e(x)$  and  $e(y)$ , but only whether  $e(x) < e(y)$  or vice-versa (since  $v(x) = v(y) = 0$ , and equality of expiration times has probability 0 here). We consider this case because the expiration order is sufficient for the first algorithm of Bryans et al. [34], and would allow optimal decisions in  $M_0$  of Figure 8.2. We denote the relative order information by  $o$ , and the corresponding scheduler classes by  $\mathfrak{S}_{\ell, v, o}^{hist}$ ,  $\mathfrak{S}_{\ell, t, o}^{hist}$  and  $\mathfrak{S}_{\ell, o}^{hist}$ .

We now define memoryless schedulers, which only see the current state and are at the core of e.g. MDP model checking. They suffice on some important formalisms to obtain optimal reachability probabilities, e.g. MDP [9].

**Definition 8.11.** *A classic memoryless scheduler only sees the current state, i.e. it is a measurable function*

$$s: S \rightarrow \text{dist}(\text{Act}' \times \text{Meas}(S)).$$

We write  $\mathfrak{S}_{\ell, v, e}^{ml}$  for the set of all such schedulers.

We apply the same restrictions as for history-dependent schedulers:

**Definition 8.12.** *A classic memoryless global-time scheduler is a measurable function*

$$s: S|_{\ell, t, e} \rightarrow \text{dist}(\text{Act}' \times \text{Meas}(S)),$$

with  $S|_{\ell, t, e}$  as in Definition 8.9. We write  $\mathfrak{S}_{\ell, t, e}^{ml}$  for the set of all such schedulers.

**Definition 8.13.** *A classic memoryless location-based scheduler is a measurable function*

$$s: S|_{\ell, e} \rightarrow \text{dist}(\text{Act}' \times \text{Meas}(S)),$$

with  $S|_{\ell, e}$  as in Definition 8.10. We write  $\mathfrak{S}_{\ell, e}^{ml}$  for the set of all such schedulers.

Again, we also consider memoryless schedulers that only see the expiration order, so we have memoryless scheduler classes  $\mathfrak{S}_{\ell, v, e}^{ml}$ ,  $\mathfrak{S}_{\ell, t, e}^{ml}$ ,  $\mathfrak{S}_{\ell, e}^{ml}$ ,  $\mathfrak{S}_{\ell, v, o}^{ml}$ ,  $\mathfrak{S}_{\ell, t, o}^{ml}$  and  $\mathfrak{S}_{\ell, o}^{ml}$ . Class  $\mathfrak{S}_{\ell, o}^{ml}$  is particularly attractive because it has a finite domain.

## 8.2.2 Non-Prophetic Schedulers

Consider the SA  $M_0$  in Figure 8.2. No matter which of the previously defined scheduler classes we choose, we always find a scheduler that achieves probability 1 to reach  $\checkmark$ , and a scheduler that achieves probability 0. This is because they can all see the expiration times, or the expiration order of  $x$  and  $y$  when in  $\ell_1$  and thus resolve the non-deterministic choice in that location optimally. This

is somewhat counter-intuitive: When in  $\ell_1$ ,  $x$  and  $y$  have not yet expired—this will only happen later, in  $\ell_2$  or  $\ell_3$ —yet the schedulers already know which clock expires first. The classic schedulers can thus be seen to make decisions based on the timing of *future* events. This *prophetic* scheduling has already been observed in [30], where a “fix” in the form of the *spent lifetimes* semantics was proposed. Hartmanns et al. [91] have shown that this not only still permits prophetic scheduling, but even admits *divine* scheduling, where a scheduler can not only see, but *change* the future behaviour. The authors propose a complex *non-prophetic* semantics that provably removes all prophetic and divine behaviour, but is conceptually involved.

Much of the complication of the non-prophetic semantics of [91] is due to it being specified for open SA that include delayable actions. For the closed SA setting of this paper, prophetic scheduling can be more easily excluded by hiding from the schedulers all information about what happens in the future of the system’s evolution. This information is only contained in the expiration times  $e$  or the expiration order  $o$ . The current location  $\ell$  and the current values of all clocks  $v$  do not permit any prophetic behaviour. We can thus keep the semantics of Section 8.1.3 and modify the definition of schedulers to exclude prophetic behaviour by construction.

In what follows, we thus also consider all scheduler classes of Section 8.2.1 with the added constraint that the expiration times, resp. the expiration order, are not visible, resulting in the *non-prophetic* classes  $\mathfrak{S}_{\ell,v}^{hist}$ ,  $\mathfrak{S}_{\ell,t}^{hist}$ ,  $\mathfrak{S}_{\ell}^{hist}$ ,  $\mathfrak{S}_{\ell,v}^{ml}$ ,  $\mathfrak{S}_{\ell,t}^{ml}$  and  $\mathfrak{S}_{\ell}^{ml}$ . Any non-prophetic scheduler can only reach  $\checkmark$  of  $M_0$  in Figure 8.2 with probability  $\frac{1}{2}$ . The last three classes are the most interesting from a control theory perspective, where we want to synthesise schedulers and implement them: The only information that is likely to be readily available for an embedded controller is the current state of the system. Clearly, it cannot know the timing of future events, and will likely not be able to keep a long history of previous states. Whether it may observe the values of some timers, or of a global one, will depend on the specific case.

### 8.3 The Power of Schedulers

Now that we have defined a number of classes of schedulers, we need to determine what the effect of the restrictions is on the ability to optimally control an SA. We thus evaluate the power of scheduler classes with respect to unbounded reachability probabilities, on the semantics of SA. We shall see that this simple setting already suffices to reveal interesting differences between scheduler classes. In the remainder of this section we consider extremal probabilities of reaching a set of goal locations  $G$ :

**Definition 8.14.** For  $G \subseteq \text{Loc}$ , let  $J_G \stackrel{\text{def}}{=} \{ \langle \ell, v, e \rangle \in S \mid \ell \in G \}$ . Let  $\mathfrak{S}$  be a set of schedulers. Then  $P_{\min}^{\mathfrak{S}}(G)$  and  $P_{\max}^{\mathfrak{S}}(G)$  are the minimum and maximum reachability probabilities for  $G$  under  $\mathfrak{S}$ , defined as  $P_{\min}^{\mathfrak{S}}(G) = \inf_{s \in \mathfrak{S}} \mathbb{P}_s(\Pi_{J_G})$  and  $P_{\max}^{\mathfrak{S}}(G) = \sup_{s \in \mathfrak{S}} \mathbb{P}_s(\Pi_{J_G})$ , where  $\Pi_{J_G}$  are runs traversing  $J_G$ .

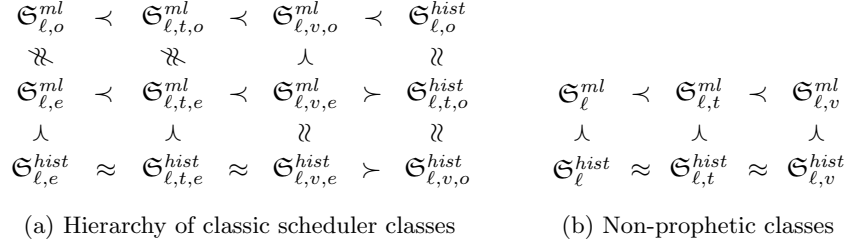


Figure 8.3: Scheduler hierarchy established in this section. For the difference between classical- and prophetic schedulers, we refer to Section 8.2.

**Example 8.15.** *To maximise the probability of reaching  $\checkmark$  of  $M_0$  in Figure 8.2 assuming  $e(x) < e(y)$  we should take the transition to the state in  $\ell_2$ . If a scheduler  $\mathfrak{s}$  can see the expiration times, noting that only their order matters here, it can always make the optimal choice and achieve  $P_{\max}^{\{\mathfrak{s}\}}(\{\checkmark\}) = 1$ .*

For two scheduler classes  $\mathfrak{S}_1$  and  $\mathfrak{S}_2$ , we write  $\mathfrak{S}_1 \succcurlyeq \mathfrak{S}_2$  if, for all SA and all sets of goal locations  $G$ ,  $P_{\min}^{\mathfrak{S}_1}(G) \leq P_{\min}^{\mathfrak{S}_2}(G)$  and  $P_{\max}^{\mathfrak{S}_1}(G) \geq P_{\max}^{\mathfrak{S}_2}(G)$ . We write  $\mathfrak{S}_1 \succ \mathfrak{S}_2$  if additionally there exists at least one SA and set  $G'$  where  $P_{\min}^{\mathfrak{S}_1}(G') < P_{\min}^{\mathfrak{S}_2}(G')$  or  $P_{\max}^{\mathfrak{S}_1}(G') > P_{\max}^{\mathfrak{S}_2}(G')$ . Finally, we write  $\mathfrak{S}_1 \approx \mathfrak{S}_2$  for  $\mathfrak{S}_1 \succcurlyeq \mathfrak{S}_2 \wedge \mathfrak{S}_2 \succcurlyeq \mathfrak{S}_1$ , and  $\mathfrak{S}_1 \not\approx \mathfrak{S}_2$ , i.e. the classes are incomparable, for  $\mathfrak{S}_1 \not\prec \mathfrak{S}_2 \wedge \mathfrak{S}_2 \not\prec \mathfrak{S}_1$ . Unless noted otherwise, we omit proofs for  $\mathfrak{S}_1 \succcurlyeq \mathfrak{S}_2$  when it is obvious that the information available to  $\mathfrak{S}_1$  includes the information available to  $\mathfrak{S}_2$ . All our distinguishing examples are based on the resolution of a single non-deterministic choice between two actions to eventually reach one of two locations. We therefore prove only with respect to the maximum probability,  $p_{\max}$ , for these examples since the minimum probability is given by  $1 - p_{\max}$  and an analogous proof for  $p_{\min}$  can be made by relabelling locations. We may write  $P_{\max}(\mathfrak{S}_x^y)$  for  $P_{\max}^{\mathfrak{S}_x^y}(\{\checkmark\})$  to improve readability.

### 8.3.1 The Classic Hierarchy

We first establish that all classic history-dependent scheduler classes are equivalent. This is not surprising, as knowing the *entire* history always lets us reconstruct all clock valuations and expiration times, respectively.

**Theorem 8.16.**  $\mathfrak{S}_{\ell,v,e}^{hist} \approx \mathfrak{S}_{\ell,t,e}^{hist} \approx \mathfrak{S}_{\ell,e}^{hist}$ .

*Proof.* From the transition labels in  $Act' = Act \sqcup \mathbb{R}^+$  in the history  $(S' \times Act')^*$ , with  $S' \in \{S, S|_{\ell,t,e}, S|_{\ell,e}\}$  depending on the scheduler class, we can reconstruct the total elapsed time as well as the values of all clocks: to obtain the total elapsed time, sum the labels in  $\mathbb{R}^+$  up to each state; to obtain the values of all clocks, do the same per clock and perform the resets of the edges identified by the actions.  $\square$

The same argument applies among the expiration-order history-dependent classes.

**Theorem 8.17.**  $\mathfrak{S}_{\ell,v,o}^{hist} \approx \mathfrak{S}_{\ell,t,o}^{hist} \approx \mathfrak{S}_{\ell,o}^{hist}$ .

However, the expiration-order history-dependent schedulers are strictly less powerful than the classic history-dependent ones.

**Theorem 8.18.**  $\mathfrak{S}_{\ell,v,e}^{hist} \succ \mathfrak{S}_{\ell,v,o}^{hist}$ .

*Proof.* Consider the SA  $M_1$  in Figure 8.4a. Note that the history does not provide any information for making the choice in  $\ell_1$ : we always arrive after having spent zero time in  $\ell_0$  and then having taken the single edge to  $\ell_1$ . We can analytically determine that  $P_{\max}(\mathfrak{S}_{\ell,v,e}^{hist}) = \frac{3}{4}$  by going from  $\ell_1$  to  $\ell_2$  if  $e(x) \leq \frac{1}{2}$  and to  $\ell_3$  otherwise. We would obtain a probability equal to  $\frac{1}{2}$  by always going to either  $\ell_2$  or  $\ell_3$  or by picking either edge with equal probability. This is the best we can do if  $e$  is not visible, and thus  $P_{\max}(\mathfrak{S}_{\ell,v,o}^{hist}) = \frac{1}{2}$ : in  $\ell_1$ ,  $v(x) = v(y) = 0$  and the expiration order is always “ $y$  before  $x$ ” because  $y$  has not yet been started, so the scheduler has no basis to make an informed choice.  $\square$

Just like for MDP and unbounded reachability probabilities, the classic history-dependent and memoryless schedulers with complete information are equivalent.

**Theorem 8.19.**  $\mathfrak{S}_{\ell,v,e}^{hist} \approx \mathfrak{S}_{\ell,v,e}^{ml}$ .

*Proof sketch.* Our definition of TPTS only allows finite non-deterministic choices, i.e. we have a very restricted form of continuous-space MDP. We can thus adapt the argument of the corresponding proof for MDP [9, Lemma 10.102]: For each state (of possibly countably many), we construct an optimal memoryless (and deterministic) scheduler in the same way, replacing the summation by an integration for the continuous measures in the transition function. It remains to be shown that this scheduler is indeed measurable. For TPTS that are the semantics of SA, this follows from the way clock values are used in the guard sets so that optimal decisions are constant over intervals of clock values and expiration times (see e.g. the arguments in [34] or [119]).  $\square$

On the other hand, when restricting schedulers to see the expiration order only, history-dependent and memoryless schedulers are no longer equivalent.

**Theorem 8.20.**  $\mathfrak{S}_{\ell,v,o}^{hist} \succ \mathfrak{S}_{\ell,v,o}^{ml}$ .

*Proof.* Consider the SA  $M_2$  in Figure 8.4b. Let  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  be the (unknown) optimal scheduler in  $\mathfrak{S}_{\ell,v,o}^{ml}$  with respect to the max. probability of reaching  $\checkmark$ . Define  $\mathfrak{s}_{hist(l,v,o)}^{better} \in \mathfrak{S}_{\ell,v,o}^{hist}$  as: when in  $\ell_2$  and the last edge in the history is the left one (i.e.  $x$  is expired), go to  $\ell_3$ ; otherwise, behave like  $\mathfrak{s}_{ml(l,v,o)}^{opt}$ . This scheduler distinguishes  $\mathfrak{S}_{\ell,v,o}^{hist}$  and  $\mathfrak{S}_{\ell,v,o}^{ml}$ , by achieving a strictly higher maximal probability than  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  if and only if there are some combinations of clock

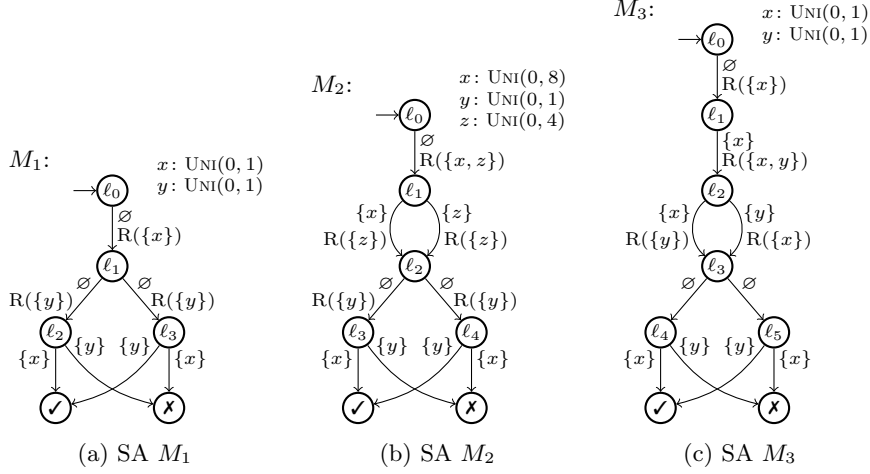


Figure 8.4: Distinguishing examples used in Theorems 8.18, 8.20, 8.21, 8.24, 8.25 and 8.29.

values (aspect  $v$ ) and expiration orders (aspect  $o$ ) in  $\ell_2$  that can be reached with positive probability via the left edge into  $\ell_2$ , for which  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  must nevertheless decide to go to  $\ell_4$ .

All possible clock valuations in  $\ell_2$  can be achieved via either the left or the right edge, but taking the left edge implies that  $x$  expires before  $z$  in  $\ell_2$ . It is thus sufficient to show that  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  must go to  $\ell_4$  in *some* cases where  $x$  expires before  $z$ . The general form of schedulers in  $\mathfrak{S}_{\ell,v,o}^{ml}$  in  $\ell_2$  is “go to  $\ell_3$  iff (a)  $x$  expires before  $z$  and  $v(x) \in S_1$  or (b)  $z$  expires before  $x$  and  $v(x) \in S_2$ ” where the  $S_i$  are measurable subsets of  $[0, 8]$ .  $S_2$  is in fact *irrelevant*: whatever  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  does when (b) is satisfied will be mimicked by  $\mathfrak{s}_{hist(l,v,o)}^{better}$  because  $z$  can only expire before  $x$  when coming via the right edge into  $\ell_2$ . Conditions (a) and (b) are independent.

With  $S_1 = [0, 8]$ , the maximal probability is  $\frac{77}{96} = 0.80208\bar{3}$ . Since this is the only scheduler in  $\mathfrak{S}_{\ell,v,o}^{ml}$  that is *relevant* for our proof, and never goes to  $\ell_4$  when  $x$  expires before  $z$ , it remains to be shown that the maximal probability under  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  is  $> \frac{77}{96}$ . With  $S_1 = [0, \frac{35}{12})$ , we have a maximal probability of  $\frac{7561}{9216} \approx 0.820421$ . Thus  $\mathfrak{s}_{ml(l,v,o)}^{opt}$  must sometimes go to  $\ell_4$  even when the left edge was taken, so  $\mathfrak{s}_{hist(l,v,o)}^{better}$  achieves a higher probability and thus distinguishes the classes.  $\square$

Knowing only the global elapsed time is less powerful than knowing the full history or the values of all clocks.

**Theorem 8.21.**  $\mathfrak{S}_{\ell,t,e}^{hist} \succ \mathfrak{S}_{\ell,t,e}^{ml}$  and  $\mathfrak{S}_{\ell,v,e}^{ml} \succ \mathfrak{S}_{\ell,t,e}^{ml}$ .

*Proof sketch.* Consider the SA  $M_3$  in Figure 8.4c. We have  $P_{\max}(\mathfrak{S}_{\ell,t,e}^{hist}) = 1$ : when in  $\ell_3$ , the scheduler sees from the history which of the two incoming edges

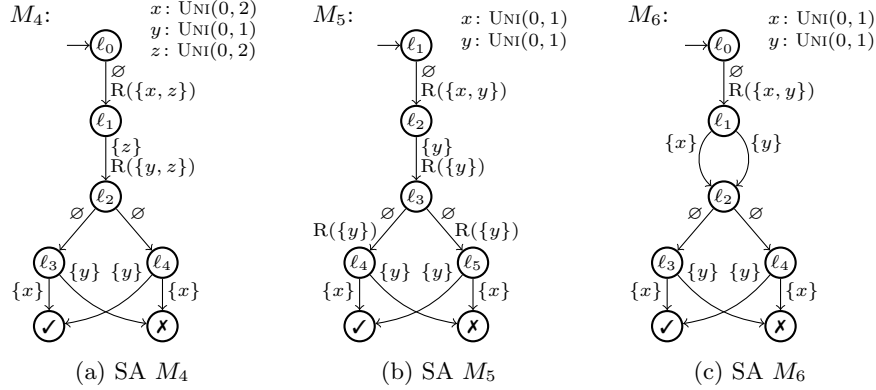


Figure 8.5: Distinguishing examples used in Theorems 8.22, 8.26, 8.28, and 8.30.

was used, and thus knows whether  $x$  or  $y$  is already expired. It can then make the optimal choice: go to  $\ell_4$  if  $x$  is already expired, or to  $\ell_5$  otherwise. We also have  $P_{\max}(\mathfrak{S}_{\ell,v,e}^{ml}) = 1$ : the scheduler sees that either  $v(x) = 0$  or  $v(y) = 0$ , which implies that the other clock is already expired, and the argument above applies. However,  $P_{\max}(\mathfrak{S}_{\ell,t,e}^{ml}) < 1$ : the distribution of elapsed time  $t$  on entering  $\ell_3$  is itself independent of which edge is taken. With probability  $\frac{1}{4}$ , exactly one of  $e(x)$  and  $e(y)$  is below  $t$  in  $\ell_3$ , which implies that that clock has just expired and thus the scheduler can decide optimally. Yet with probability  $\frac{3}{4}$ , the expiration times are not useful: they are both positive and drawn from the same distribution, but one unknown clock is expired. The wait for  $x$  in  $\ell_1$  ensures that comparing  $t$  with the expiration times in  $e$  does not reveal further information in this case.  $\square$

In the case of MDP, knowing the total elapsed time (i.e. steps) does not make a difference for unbounded reachability. Only for step-bounded properties is that extra knowledge necessary to achieve optimal probabilities. With SA, however, it makes a difference even in the unbounded case.

**Theorem 8.22.**  $\mathfrak{S}_{\ell,t,e}^{ml} \succ \mathfrak{S}_{\ell,e}^{ml}$ .

*Proof.* Consider SA  $M_4$  in Figure 8.5a. We have  $P_{\max}(\mathfrak{S}_{\ell,t,e}^{ml}) = 1$ : in  $\ell_2$ , the remaining time until  $y$  expires is  $e(y)$  and the remaining time until  $x$  expires is  $e(x) - t$  for the global time value  $t$  as  $\ell_2$  is entered. The scheduler can observe all of these quantities and thus optimally go to  $\ell_3$  if  $x$  will expire first, or to  $\ell_4$  otherwise. However,  $P_{\max}(\mathfrak{S}_{\ell,e}^{ml}) < 1$ :  $e(x)$  only contains the absolute expiration time of  $x$ , but without knowing  $t$  or the expiration time of  $z$  in  $\ell_1$ , and thus the current value  $v(x)$ , this scheduler cannot know with certainty which of the clocks expires first and is therefore unable to make an optimal choice in  $\ell_2$ .  $\square$

Finally, we need to compare the memoryless schedulers that see the clock expiration times with memoryless schedulers that see the expiration order. As

noted in before, these two views of the current state are incomparable unless we also see the clock values.

**Theorem 8.23.**  $\mathfrak{S}_{\ell,v,e}^{ml} \succ \mathfrak{S}_{\ell,v,o}^{ml}$ .

*Proof.*  $\mathfrak{S}_{\ell,v,e}^{ml} \not\approx \mathfrak{S}_{\ell,v,o}^{ml}$  follows from the same argument as in the proof of Theorem 8.18.  $\mathfrak{S}_{\ell,v,e}^{ml} \succ \mathfrak{S}_{\ell,v,o}^{ml}$  is because knowing the current clock values  $v$  and the expiration times  $e$  is equivalent to knowing the expiration order, since that is precisely the order of the differences  $e(c) - v(c)$  for all clocks  $c$ .  $\square$

**Theorem 8.24.**  $\mathfrak{S}_{\ell,t,e}^{ml} \not\approx \mathfrak{S}_{\ell,t,o}^{ml}$ .

*Proof.*  $\mathfrak{S}_{\ell,t,e}^{ml} \not\approx \mathfrak{S}_{\ell,t,o}^{ml}$  follows from the same argument as in the proof of Theorem 8.18. For  $\mathfrak{S}_{\ell,t,e}^{ml} \not\approx \mathfrak{S}_{\ell,t,o}^{ml}$ , consider the SA  $M_3$  of Figure 8.4c. We know from the proof of Theorem 8.21 that  $P_{\max}(\mathfrak{S}_{\ell,t,e}^{ml}) < 1$ . However, if the scheduler knows the order in which the clocks will expire, it knows which one has already expired (the first one in the order), and can thus make the optimal choice in  $\ell_3$  to achieve  $P_{\max}(\mathfrak{S}_{\ell,t,o}^{ml}) = 1$ .  $\square$

**Theorem 8.25.**  $\mathfrak{S}_{\ell,e}^{ml} \not\approx \mathfrak{S}_{\ell,o}^{ml}$ .

*Proof.* The argument of Theorem 8.24 applies by observing that, in  $M_3$  of Figure 8.4c, we also have  $P_{\max}(\mathfrak{S}_{\ell,e}^{ml}) < 1$  via the same argument as for  $\mathfrak{S}_{\ell,t,e}^{ml}$  in the proof of Theorem 8.21.  $\square$

Among the expiration-order schedulers, the hierarchy is as expected.

**Theorem 8.26.**  $\mathfrak{S}_{\ell,v,o}^{ml} \succ \mathfrak{S}_{\ell,t,o}^{ml} \succ \mathfrak{S}_{\ell,o}^{ml}$ .

*Proof sketch.* Consider  $M_5$  of Figure 8.5b. To maximise the probability, in  $\ell_3$  we should go to  $\ell_4$  whenever  $x$  is already expired or close to expiring, for which the amount of time spent in  $\ell_2$  is an indicator.  $\mathfrak{S}_{\ell,o}^{ml}$  only knows that  $x$  may have expired when the expiration order is “ $x$  before  $y$ ”, but definitely has not expired when it is “ $y$  before  $x$ ”. Schedulers in  $\mathfrak{S}_{\ell,t,o}^{ml}$  can do better: They also see the amount of time spent in  $\ell_2$ . Thus  $\mathfrak{S}_{\ell,t,o}^{ml} \succ \mathfrak{S}_{\ell,o}^{ml}$ . If we modify  $M_5$  by adding an initial delay on  $x$  from a new  $\ell_0$  to  $\ell_1$  as in  $M_3$ , then the same argument can be used to prove  $\mathfrak{S}_{\ell,v,o}^{ml} \succ \mathfrak{S}_{\ell,t,o}^{ml}$ : the extra delay makes knowing the elapsed time  $t$  useless with positive probability, but the exact time spent in  $\ell_2$  is visible to  $\mathfrak{S}_{\ell,v,o}^{ml}$  as  $v(x)$ .  $\square$

We have thus established the hierarchy of classic schedulers shown in Figure 8.3a, noting that some of the relationships follow from the theorems by transitivity.



### 8.3.2 The Non-Prophetic Hierarchy

Each non-prophetic scheduler class is clearly dominated by the classic and expiration-order scheduler classes that otherwise have the same information, for example  $\mathfrak{S}_{\ell,v,e}^{hist} \succ \mathfrak{S}_{\ell,v}^{hist}$ . This can be shown with very simple distinguishing SA. We show that the non-prophetic hierarchy follows the shape of the classic case, including the difference between global-time and pure memoryless schedulers, with the notable exception of memoryless schedulers being weaker than history-dependent ones.

To show the hierarchy of non-prophetic schedulers, we first re-use the argument of Theorem 8.16 to establish that here, too, the history-dependent schedulers are equivalent.

**Theorem 8.27.**  $\mathfrak{S}_{\ell,v}^{hist} \approx \mathfrak{S}_{\ell,t}^{hist} \approx \mathfrak{S}_{\ell}^{hist}$ .

*Proof.* This follows from the argument of Theorem 8.16.  $\square$

In comparing non-prophetic history-dependent with non-prophetic memoryless schedulers, we surprisingly obtain a different result than in the classic setting.

**Theorem 8.28.**  $\mathfrak{S}_{\ell,v}^{hist} \succ \mathfrak{S}_{\ell,v}^{ml}$ .

*Proof.* Consider the SA  $M_6$  in Figure 8.5c. It is similar to  $M_4$  of Figure 8.5a, and our arguments are thus similar to the proof of Theorem 8.22. On  $M_6$ , we have  $P_{\max}(\mathfrak{S}_{\ell,v}^{hist}) = 1$ : in  $\ell_2$ , the history reveals which of the two incoming edges was used, i.e. which clock is already expired, thus the scheduler can make the optimal choice. However, if neither the history nor  $e$  is available, we get  $P_{\max}(\mathfrak{S}_{\ell,v}^{ml}) = \frac{1}{2}$ : the only information that can be used in  $\ell_2$  are the values of the clocks, but  $v(x) = v(y)$ , so there is no basis for an informed choice.  $\square$

Other than that, the non-prophetic hierarchy is the same as in the classic case, including the difference between global-time and pure memoryless schedulers.

**Theorem 8.29.**  $\mathfrak{S}_{\ell,t}^{hist} \succ \mathfrak{S}_{\ell,t}^{ml}$  and  $\mathfrak{S}_{\ell,v}^{ml} \succ \mathfrak{S}_{\ell,t}^{ml}$ .

*Proof.* Consider the SA  $M_3$  in Figure 8.4c. We have  $P_{\max}(\mathfrak{S}_{\ell,t}^{hist}) = P_{\max}(\mathfrak{S}_{\ell,v}^{ml}) = 1$ , but  $P_{\max}(\mathfrak{S}_{\ell,t}^{ml}) = \frac{1}{2}$  by the same arguments as in the proof of Theorem 8.21.  $\square$

**Theorem 8.30.**  $\mathfrak{S}_{\ell,t}^{ml} \succ \mathfrak{S}_{\ell}^{ml}$ .

*Proof.* Consider the SA  $M_4$  in Figure 8.5a. The schedulers in  $\mathfrak{S}_{\ell}^{ml}$  have no information but the current location, so they cannot make an informed choice in  $\ell_2$ . This and the simple loop-free structure of  $M_4$  make it possible to analytically calculate the resulting probability:  $P_{\max}(\mathfrak{S}_{\ell}^{ml}) = \frac{17}{24} = 0.708\bar{3}$ . If information about the global elapsed time  $t$  in  $\ell_2$  is available, however, the value of  $x$  is revealed. This allows making a better choice, e.g. going to  $\ell_3$  when  $t \leq \frac{1}{2}$  and to  $\ell_4$  otherwise, resulting in  $P_{\max}(\mathfrak{S}_{\ell,t}^{ml}) \approx 0.771$  (statistically estimated with high confidence).  $\square$

We have thus established the hierarchy of non-prophetic schedulers shown in Figure 8.3b, where some relationships follow by transitivity.

## 8.4 Experiments

We built a prototype implementation of lightweight scheduler sampling for SA by extending the MODEST TOOLSET’s [88] MODES simulator, which already supports deterministic stochastic timed automata (STA [22]). Our goal is to experimentally find the minimal and maximal probabilities of unbounded reachability, that in turn confirm, or refute the theoretical implications of Section 8.3. Consequently, the experimental results show the feasibility of lightweight scheduler sampling when applied to SA with respect to the hierarchy of Figure 8.3

**Remark 8.31.** *This work was motivated by our desire to extend to SA the lightweight scheduler sampling approach that allows statistical model checking for non-deterministic models, which was first defined for MDP [52] and later extended to probabilistic timed automata (PTA) [49]. In contrast to model checking algorithms like value iteration for MDP, which only implicitly deal with schedulers, the lightweight approach has full control over the information that the scheduler it currently simulates gets: this is the data fed into the hash function to compute the seed for the pseudo-random number generator used to make the non-deterministic choices for each state (see line 3 of Algorithm 1 and line 10 of Algorithm 2 in [49]). To be effective, this approach needs the space of information available to the scheduler to be “small” and at least countable. Otherwise, it easily degenerates into performing simulation with the uniform scheduler. This is why the extension to PTA used a variant of the zone graph, a discrete abstraction of the PTA that preserves reachability probabilities. To extend the approach to SA, we need to find a reasonable class of schedulers that is “as small as possible”.*

**Experimental setup.** With some care, SA can be encoded into STA. Using the original algorithm for MDP of [52], our prototype works by providing to the schedulers a discretised view of the continuous components of the SA’s semantics, which, we recall, is a continuous-space MDP. The currently implemented discretisation is simple: for each real-valued quantity (the value  $v(c)$  of clock  $c$ , its expiration time  $e(c)$ , and the global elapsed time  $t$ ), it identifies all values that lie within the same interval  $[\frac{i}{n}, \frac{i+1}{n})$ , for integers  $i, n$ . We note that better static discretisations are almost certainly possible, e.g. a region construction for the clock values as in [119].

We have modelled  $M_1$  through  $M_6$  as STA in MODEST. For each scheduler class and model in the proof of a theorem, and discretisation factors  $n \in \{1, 2, 4\}$ , we sampled 10 000 schedulers and performed statistical model checking for each of them in the lightweight manner.

**Results.** In Figure 8.6 we report the min. and max. estimates,  $(\hat{p}_{\min}, \hat{p}_{\max})_{\mathcal{I}}$ , over all sampled schedulers. Where different discretisations lead to different estimates, we report the most extremal values. The subscript  $\mathcal{I}$  denotes the discretisation factors that achieved the reported estimates. The analysis for each sampled scheduler was performed with a number of simulation runs sufficient for the overall max./min. estimates to be within  $\pm 0.01$  of the true maxima/minima

$M_1$ :	$M_3$ :	$M_2$ :	$M_5$ :
Theorem 8.18:	Theorem 8.21:	Theorem 8.20:	Theorem 8.26:
$\mathfrak{S}_{\ell,v,e}^{hist} : (0.24, 0.76)_{2,4}$	$\mathfrak{S}_{\ell,t,e}^{hist} : (0.00, 1.00)_1$	$\mathfrak{S}_{\ell,o}^{hist} : (0.06, 0.94)_{1,2,4}$	$\mathfrak{S}_{\ell,t,o}^{ml} : (0.15, 0.86)_4$
$\mathfrak{S}_{\ell,v,o}^{hist} : (0.49, 0.51)_{1,2,4}$	$\mathfrak{S}_{\ell,v,e}^{ml} : (0.12, 0.86)_4$	$\mathfrak{S}_{\ell,v,o}^{ml} : (0.18, 0.83)_1$	$\mathfrak{S}_{\ell,o}^{ml} : (0.16, 0.84)_{1,2,4}$
Theorem 8.23:	$\mathfrak{S}_{\ell,t,e}^{ml} : (0.37, 0.65)_2$	$M_4$ :	$M_6$ :
$\mathfrak{S}_{\ell,v,e}^{ml} : (0.24, 0.76)_{2,4}$	Theorem 8.24:	Theorem 8.22:	Theorem 8.28:
$\mathfrak{S}_{\ell,v,o}^{ml} : (0.49, 0.51)_{1,2,4}$	$\mathfrak{S}_{\ell,t,e}^{ml} : (0.37, 0.65)_2$	$\mathfrak{S}_{\ell,t,e}^{ml} : (0.25, 0.79)_1$	$\mathfrak{S}_{\ell,v}^{hist} : (0.00, 1.00)_{1,2}$
Theorem 8.24:	$\mathfrak{S}_{\ell,t,o}^{ml} : (0.00, 1.00)_{1,2}$	$\mathfrak{S}_{\ell,e}^{ml} : (0.29, 0.71)_1$	$\mathfrak{S}_{\ell,v}^{ml} : (0.49, 0.51)_{1,2,4}$
$\mathfrak{S}_{\ell,t,e}^{ml} : (0.24, 0.76)_{2,4}$	Theorem 8.25:	Theorem 8.30:	
$\mathfrak{S}_{\ell,t,o}^{ml} : (0.49, 0.51)_{1,2,4}$	$\mathfrak{S}_{\ell,e}^{ml} : (0.34, 0.67)_4$	$\mathfrak{S}_{\ell,t}^{ml} : (0.22, 0.78)_{2,4}$	
Theorem 8.25:	$\mathfrak{S}_{\ell,o}^{ml} : (0.00, 1.00)_{1,2,4}$	$\mathfrak{S}_{\ell}^{ml} : (0.28, 0.72)_{1,2,4}$	
$\mathfrak{S}_{\ell,e}^{ml} : (0.24, 0.76)_{2,4}$	Theorem 8.29:		
$\mathfrak{S}_{\ell,o}^{ml} : (0.49, 0.51)_{1,2,4}$	$\mathfrak{S}_{\ell,t}^{hist} : (0.00, 1.00)_{1,2}$		
	$\mathfrak{S}_{\ell,v}^{ml} : (0.21, 0.78)_4$		
	$\mathfrak{S}_{\ell,t}^{ml} : (0.49, 0.51)_{1,2,4}$		

Figure 8.6: Results from the prototype of lightweight scheduler sampling for SA

of the *sampled* set of schedulers with probability  $\geq 0.95$  [52]. Note that  $\hat{p}_{\min}$  is an upper bound on the true minimum probability and  $\hat{p}_{\max}$  is a lower bound on the true maximum probability. We thus use these results to show the feasibility of the lightweight scheduler sampling technique applied to SA through the correspondence with the hierarchy of schedulers in the main body of the paper.

Increasing the discretisation factor or increasing the scheduler power generally increases the number of decisions the schedulers *can* make. This may also increase the number of *critical* decisions a scheduler *must* make to achieve the extremal probability. Hence, the sets of discretisation factors associated to specific experiments may be informally interpreted in the following way:

- $\{1, 2, 4\}$ : Fine discretisation is not important for optimality and optimal schedulers are not rare.
- $\{1, 2\}$ : Fine discretisation is not important for optimality, but increases rarity of optimal schedulers.
- $\{2, 4\}$ : Fine discretisation is important for optimality, optimal schedulers are not rare.
- $\{1\}$ : Optimal schedulers are very rare.
- $\{2\}$ : Fine discretisation is important for optimality, but increases rarity of schedulers.
- $\{4\}$ : Fine discretisation is important for optimality and optimal schedulers are not rare.

The results in Figure 8.6 respect and differentiate our hierarchy. In most cases, we found schedulers whose estimates were within the statistical error of calculated optima or of high confidence estimates achieved by alternative

statistical techniques. The exceptions involve  $M_3$  and  $M_4$ . We note that  $M_4$  makes use of an additional clock, increasing the dimensionality of the problem and potentially making near-optimal schedulers rarer. The best result for  $M_3$  and class  $\mathfrak{S}_{l,t,e}^{ml}$  was obtained using discretisation factor  $n = 2$ : a compromise between nearness to optimality and rarity. A greater compromise was necessary for  $M_4$ , and classes  $\mathfrak{S}_{l,t,e}^{ml}$  and  $\mathfrak{S}_{l,e}^{ml}$ , where we found near-optimal schedulers to be very rare and achieved best results using discretisation factor  $n = 1$ .

While we found no scheduler whose performance exceeded its defined theoretical limits, we note that our sampling approach is not exhaustive, and that when non-determinism is not spurious, there is a positive probability that we may miss some (near-)optimal schedulers and potentially miss rare counterexamples, if such exist. In the case of the present experiments, we also note that our static grid discretisation makes some schedulers impossible.

**Discussion.** The experiments show that lightweight scheduler sampling produces useful and informative results with SA. We believe the few sub-optimal values can be significantly improved by tailoring the discretisation to the particular instance. The continuous quantities do not all require the same granularity of discretisation and not all boundaries within a discretisation are useful. The present theoretical results allow us to develop better abstractions for SA, and thus to construct a refinement algorithm for efficient lightweight verification of SA that is applicable to realistically sized case studies. As is, they already demonstrate the importance of selecting a proper scheduler class for efficient verification, and that restricted classes are useful in planning scenarios.

## 8.5 Conclusions

Motivated by the possibility to apply statistical model checking to non-deterministic stochastic automata, we have defined a number of classes of schedulers and compared their expressiveness. We have shown that the various notions of information available to a scheduler class, such as history, clock order, expiration times or overall elapsed time, almost all make distinct contributions to the power of the class in SA. Our choice of notions was based on classic scheduler classes relevant for other stochastic models, previous literature on the character of non-determinism in and verification of SA, and the need to synthesise simple schedulers in planning. Our distinguishing examples clearly expose how to exploit each notion to improve the probability of reaching a goal. For verification of SA, we have demonstrated the feasibility of lightweight scheduler sampling, where the different notions may be used to finely control the power of the lightweight schedulers. To solve stochastic timed planning problems defined via SA, our analysis helps in the case-by-case selection of an appropriate scheduler class that achieves the desired trade-off between optimal probabilities and ease of implementation of the resulting plan.

The outcome of our investigation into the power of schedulers could be seen as a mostly negative result: there is no “easy” class that is equivalent to seeing the

entirety of a state with all clock values and expiration times, even in the simple setting of reachability probabilities. Even worse, in the non-prophetic setting, history matters, too. However, due to the undecidability of the underlying model checking problem, a strong, simple result was not to be expected. Yet we provide several important insights, for example that merely considering the expiration order is insufficient unless one restricts to regeneration on every edge as in [34]: our distinguishing SA for this case indeed relies on clock memory across locations. In general, our distinguishing SA clearly exhibit several interesting phenomena that can occur in SA and need to be taken into account in any SA verification approach. In particular, our insights are invaluable in building abstraction refinement techniques for schedulers to replace the crude discretisation of our prototype SMC tool, and they provide guidelines for control scenarios where an implementable class of schedulers to synthesise needs to be picked. We note that all the distinguishing SA we used in our proofs follow a simple common pattern and do not make use of any potentially “exotic” features of our definition of SA: they are simple acyclic SA with absolutely continuous probability measures, edges with either empty or singleton guard sets only, and at most three clocks.

We conjecture that the arguments of this paper also extend to steady-state/frequency measures (in the counterexamples, we would add loops back from absorbing to initial states), and that our results for the classic schedulers also transfer to SA with delayable actions. The next step should be to use the results of this paper to build a proper refinement-based implementation of the lightweight SMC approach as mentioned above.



# CHAPTER 9

---

## Conclusions

---

Model-based testing is appealing due to its high degree of automation: given a model, the otherwise labour-intensive and error-prone manual derivation of test cases, their execution, and their evaluation can be performed in a fully automatic way. It is therefore not surprising that MBT rapidly gained popularity in the industry, by providing more *structured* and more *efficient* test procedures. The widespread application of MBT thus benefits both suppliers and consumers, by granting higher quality products and services at lower costs.

In this thesis, we set out to develop an applicable MBT framework, that treats probabilistic decisions of systems and algorithms as “first class citizens”. Probabilistic decisions and soft real-time constraints were a focal point of the developed theory. In particular, we were interested to reconcile these properties with *non-deterministic choices*, thus allowing underspecification and implementation freedom in a fully-fledged way. This represents the major advantage over related approaches, that either assumed closed or *fully* probabilistic models that allow no interaction. To conclude the thesis, we summarise the presented work, and discuss possible future research.

### 9.1 Summary

The main contribution of this thesis is the applicable MBT framework for non-deterministic systems that exhibit probabilistic behaviour. This includes discrete probability choices made by the system itself, and soft real-time constraints on its delays. We provided evidence for its theoretical correctness, as well as readily available methods to apply it in practice.

Non-determinism in our models is resolved *probabilistically* by means of schedulers, and their resulting trace distributions. A study on a novel notion of schedulers, as well as a hierarchy of existing ones, thus mark another contribution of this thesis.

## Model-Based Testing

Our endeavour to develop the presented framework started by recalling **ioco** theory from the literature [174, 169] in Chapter 3. This choice was motivated by the fact, that **ioco** relied on labelled transition systems. LTSs utilize non-determinism – a highly desirable feature, given that it allows underspecification, implementation freedom, as well as the capabilities to model interaction with the implementation via parallel composition. They also form the foundation of many other modelling formalisms with varying properties. It thus seemed natural to use the extensions of LTSs to extend **ioco** theory itself.

With our desire to focus on probabilistic systems, probabilistic automata [158] were an instinctive choice as underlying modelling formalism. They allow to both model non-deterministic choices, as well as discrete probabilities. Inspired by the seminal work of [40], we first set out to conservatively extend the conformance relation **ioco** itself. The authors work culminated in the connection of *observations* and *trace distributions* – a connection that relates the invisible content of a black-box implementation with examinations made by an external observer. Hence, we developed the conformance relation **pioco** based on *finite* schedulers that induce trace distributions in Chapter 4. This enabled us to compare two probabilistic input output transition systems on a purely formal level. In particular, we showed our testing relation to be a conservative extension of **ioco**. This guaranteed that desirable features carried over to our theory, e.g. **pioco** is coarser than trace (distribution) inclusion. The remaining theory of test cases, their executions and evaluations was developed thereafter. We proved the framework to be *sound* and, in theory, *complete*. To see the framework’s application in practice, we conducted smaller-sized case studies based on examples known from the literature: 1. Kuth’s and Yao’s dice programs [114] 2. the binary exponential back-off protocol [106], and 3. the FireWire root contention protocol [165]. We implemented mutant applications, all of which were successfully eliminated, while all correct implementations passed our tests.

The next step comprised the incorporation of soft real-time constraints and stochastic time delays. However, before taking on the challenge of *general probability distributions* over time, we first focussed on exponential distributions. Exponentials are attractive due to their memoryless property, which allowed us to avoid *clocks* to account for real-time. Markov automata [67] were a tailor-made model, given their usage of non-determinism, discrete probabilities and exponential delays. This allowed us to study real-time in the established theory for pIOTSs with minor extensions to the modelling formalism in Chapter 5. To our surprise, the results carried over to the real-time setting almost exclusively. The reason for that is the near-modular design of the developed MBT framework; Conformance in **pioco** is defined via trace distributions, and the resulting probabilities assigned to traces. Thus, constructing schedulers in a similar manner meant that no major changes were needed on the theoretical side. On the practical side, we required new methods to infer whether observed *time* could be attributed to specified parameters of exponential distributions. In addition to Pearson’s  $\chi^2$  test, we utilized confidence intervals for Markovian parameters.



Carrying out multiple hypotheses tests on the same sample data meant that we have to apply  $\alpha$ -correction, to prevent an inflation of a type I error.

Lastly, we studied general continuous probability distributions as soft real-time requirements. Stochastic automata [50] enabled a clean treatment of this by utilizing clocks whose expiration time is based on such distributions. A careful construction of schedulers resulted in the presented framework for stochastic automata of Chapter 7. While theoretical aspects like the conformance relation, test cases, verdicts and the framework's correctness carried over, we once again needed to adjust practical methods. To that end, we used non-parametric inference methods to decide whether a sample of time stamps was drawn from a theoretical probability distribution. To define the framework as general as possible, we used the Kolmogorov-Smirnov test, and pointed out, that more efficient statistical tests exist for specific distributions.

## Trace Distribution Semantics

Driven by the desire to model *waiting* of a tester, we studied a novel notion of schedulers for Markov automata [67]. In particular, schedulers were equipped with the capability to *wait* before choosing the next action by scheduling distributions over time. This resulted in the *stoic* trace distributions semantics of Chapter 6. Trace distribution semantics of non-waiting schedulers were shown to be incomparable to those induced by stoic schedulers – a surprising result, given that the former are a subset of the latter. The comparison is spoiled by the maximal progress assumption, meaning that time is not allowed to pass in states that allow internal progress. However, stoic trace distribution semantics were shown to be coarser than existing notions of bisimulation relations. This is a natural result, given that trace-based semantics take on a linear view of a system's development, while bisimulation focuses on branching probabilities.

Non-determinism is resolved probabilistically by means of schedulers in all chapters. We took on the fundamental challenge to study schedulers in their own right in Chapter 8. Investigating schedulers for stochastic automata means our results carry over to probabilistic- and Markov automata, given their evident subset relation. We studied schedulers inspired by the literature of the classic complete information-view [30] and the non-prophetic view [91]. We used unbounded reachability problems and altered the information available to schedulers in any given location to establish a hierarchy of scheduler classes. The outcome of our investigation could be seen as a mostly negative result, as there was no *simple* class that yields optimal results. However, despite the fact that in most settings, memoryless schedulers suffice for reachability problems, we found that this is not the case for non-prophetic schedulers. Here, history based schedulers perform better. The study of scheduling power was inspired by their application in lightweight scheduler sampling. Our theoretical findings were consequently confirmed by a small scale experiment on LSS on our presented examples.

## 9.2 Discussion and Future Work

The presented MBT framework is provably correct: Every conforming implementation has an arbitrarily high chance to pass a test suite, and there exist test cases to reveal every non-conforming implementation. To apply the developed theory in practice, we studied several smaller sized case studies. It is thus a desirable goal to bring our theory into application on a larger scale.

To that end, the impeding factor is the non-existence of a proper *tool*. Our small examples were carried out by a tool-chain consisting of JTorX [15], MATLAB [86] and semi-automatically performed tasks. This included the evaluation of JTorX's log files, where an off-the-shelf shell script was used to count trace frequencies. On the same note, the probabilistic verdict relies on a comparison of those trace frequencies to their expectations. Many of the presented examples used relatively short sequences, so expected probabilities were mostly calculated by hand. A *push-button* MBT tool for such systems, would thus require to fill these two shortcomings. We do not expect this to necessitate an entirely new tool, as existing MBT tools already implement **ioco** theory. Rather, 1. existing tools could be extended to allow probabilistic specifications. Parametrized expected probabilities can then be calculated, and 2. the tool could be provided with parameters like test length or sample size for multiple test executions. This would aid in the sampling process necessary to apply our theory.

A second bottleneck lies in the requirement to gather *sufficiently large samples*. Collecting, say,  $10^5$  traces is an unreasonable amount when working in a realistic and industrial setting, particularly if soft real-time constraints and time delay are considered. While the execution of many passing test cases increases confidence in the functional correctness of the system, we see a possible solution of this dilemma in more *efficient* statistical hypothesis tests. That is, statistical tests with a higher *power* for equal sample size. To that end, we investigated sequential probability ratio tests (SPRT) – statistical tests that do not work on fixed sample sizes, but fixed errors and variable sample size (Appendix A.2). With every additional observation, a statistical test is performed to see whether a probabilistic verdict can be given with the current information. This approach, however, is spoiled by the fact that we still need to find the *best fitting* scheduler, maximising the probability of a sample to occur. Hence, the advantage of SPRTs is lost. The field of statistical hypothesis testing is a rich field, and more suitable tests than Pearson's  $\chi^2$  may exist. A different solution to reduce sample size requires more efficient tests than the Kolmogorov-Smirnov test. This is possible for parametrized distributions, but highly dependent on the application domain.

Lastly, finding the best-fitting scheduler may be substantially reduced in complexity, if smaller scheduler classes are considered.

# Appendices



# APPENDIX A

---

## Mathematical Background

---

We provide the background for the mathematical concepts used within the thesis. The covered material includes the basics of probability theory, and a short introduction to statistical hypothesis testing. Many of these may be familiar to the reader, and are appended for the sake of self-containment of the thesis. For a thorough introduction to probability theory, we refer to [46, 123], and for a better discussion on statistical hypothesis testing see [156].

### A.1 Probability Theory

We shortly recall basic concepts from probability theory used in the main body of the thesis. This is a brief overview for references, and the reader is referred to textbooks covering probability theory for a thorough and detailed introduction, e.g. [46, 123].

**Definition A.1** (Probability Distribution). *A probability sub-distribution over a countable set  $S$  is a function  $\mu : S \rightarrow [0, 1]$ , such that*

$$\sum_{s \in S} \mu(s) \leq 1.$$

*We use  $\text{SubDistr}(S)$  to denote the set of all sub-distributions over  $S$ . We write  $|\mu| = \sum_{s \in S} \mu(s)$  for the probability mass of  $\mu$ . If  $\mu$  is a probability sub-distribution with  $|\mu| = 1$ , then we call  $\mu$  a probability distribution and denote all such distributions over  $S$  by  $\text{Distr}(S)$ .*

**Definition A.2** (Support). *The support of a sub-distribution  $\mu \in \text{SubDistr}(S)$  is denoted*

$$\text{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$$

*Given a real number  $x$  and a sub-distribution  $\mu$  we denote  $x \cdot \mu$  the sub-distribution such that  $(x \cdot \mu)(s) = x \cdot \mu(s)$  for each  $s \in \text{supp}(\mu)$  if  $x \cdot |\mu| \leq 1$ .*

**Definition A.3** (Dirac Distribution). *If the support of a distribution is a singleton  $\{s\}$ , we call it a Dirac distribution, and denote it by  $\bar{s}$ , or Dirac( $s$ ).*

**Definition A.4** (Field/ $\sigma$ -Field). *Let  $\Omega$  be a non-empty set and  $\mathcal{F} \subseteq 2^\Omega$  a set of subsets of  $\Omega$ . Then  $\mathcal{F}$  is called a field, if it satisfies*

1.  $\Omega \in \mathcal{F}$ ,
2.  $A \in \mathcal{F} \implies A^c \in \mathcal{F}$ , and
3.  $A_1, A_2, \dots, A_n \in \mathcal{F} \implies \bigcup_{i=1}^n A_i \in \mathcal{F}$ .

$\mathcal{F}$  is called a  $\sigma$ -field, if instead of item 3 we have

$$A_1, A_2, \dots \in \mathcal{F} \implies \bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$$

**Definition A.5** (Borel  $\sigma$ -field). *The Borel  $\sigma$ -field  $\mathfrak{B}(\mathbb{R})$  is the smallest  $\sigma$ -field generated by the field of finite disjoint unions of right-semiclosed intervals in  $\mathbb{R}$ .*

**Definition A.6** (Measurable Space). *Let  $S$  be a set and  $\mathcal{F}_S$  a  $\sigma$ -field of  $S$ . Then we call  $(S, \mathcal{F}_S)$  a measurable space.*

**Definition A.7** ( $\sigma$ -additivity). *Let  $\mathcal{F}$  be a  $\sigma$ -algebra, and  $A_1, A_2, A_3 \dots$  pairwise disjoint sets in  $\mathcal{F}$ . A function  $\mu : \mathcal{F} \rightarrow \mathbb{R} \cup \{\infty\}$  is called  $\sigma$ -additive, if*

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i).$$

**Definition A.8** (Probability Measure). *Let  $(S, \mathcal{F}_S)$  be a measurable space. A  $\sigma$ -additive function  $\mu : \mathcal{F}_S \rightarrow [0, 1]$  is called a probability sub-measure, if  $\mu(S) \leq 1$ . We denote all sub-measures of  $S$  by  $\text{SubMeas}(S)$ . If additionally  $\mu(S) = 1$ , then we call  $\mu$  a probability measure and denote all measures of  $S$  by  $\text{Meas}(S)$ .*

**Definition A.9** (Product Measure). *Given probability measures  $\mu_1$  and  $\mu_2$ , we denote by  $\mu_1 \otimes \mu_2$  the product measure, which is the unique probability measure such that  $(\mu_1 \otimes \mu_2)(B_1 \times B_2) = \mu_1(B_1) \cdot \mu_2(B_2)$  for all  $B_1 \in \mathcal{F}_1$  and  $B_2 \in \mathcal{F}_2$ . For a collection of measures  $(\mu_i)_{i \in I}$ , we analogously denote the product measure by  $\bigotimes_{i \in I} \mu_i$ . We lift the same notation to a collection of sets of probability measures  $(M_i)_{i \in I}$  by*

$$\bigotimes_{i \in I} M_i \stackrel{\text{def}}{=} \left\{ \bigotimes_{i \in I} \mu_i \mid \mu_i \in M_i \text{ for all } i \in I \right\}.$$

**Definition A.10** (Probability Space). *A probability space is a triple  $(\Omega, \mathcal{F}, P)$ , such that*

1.  $\Omega$  is a set, called the sample set,
2.  $\mathcal{F} \subseteq 2^\Omega$  is a  $\sigma$ -field over  $\Omega$ . The elements of  $\mathcal{F}$  are called events, and
3.  $P : \mathcal{F} \rightarrow [0, 1]$  is a probability measure.

**Definition A.11** (Measurable function). Let  $(\Omega, \mathcal{F}_\Omega)$  and  $(\Omega', \mathcal{F}_{\Omega'})$  be measurable spaces. A function  $f : \Omega \rightarrow \Omega'$  is called measurable with respect to  $\mathcal{F}_\Omega$  and  $\mathcal{F}_{\Omega'}$ , if  $f^{-1}(B) \in \mathcal{F}_\Omega$  for all  $B \in \mathcal{F}_{\Omega'}$ .

**Definition A.12** (Borel-measurable function). Let  $(\Omega, \mathcal{F}_\Omega)$  be a measurable space. A measurable function  $f : \Omega \rightarrow \mathbb{R}$  with respect to  $\mathcal{F}_\Omega$  and  $\mathfrak{B}(\mathbb{R})$  is called Borel-measurable. We write  $f : (\Omega, \mathcal{F}_\Omega) \rightarrow (\mathbb{R}, \mathfrak{B}(\mathbb{R}))$  to denote Borel-measurable functions.

**Definition A.13** (Random variable). Let  $(\Omega, \mathcal{F}_\Omega, P)$  be a probability space. A Borel-measurable function  $X : (\Omega, \mathcal{F}_\Omega) \rightarrow (\mathbb{R}, \mathfrak{B}(\mathbb{R}))$  is called a random variable.

**Definition A.14** (Density functions). Let  $X$  be a real-valued random variable on a probability space  $(\Omega, \mathcal{F}_\Omega, P)$ . A real-valued, Lebesgue-integrable function  $f$  is called probability density function (PDF), if for all  $a \in \mathbb{R}$

$$P(X \leq a) = \int_{-\infty}^a f(x)dx$$

The cumulative distribution function (CDF)  $F_X$  of a PDF is given by  $F_X(x) = P(X \leq x)$ .

**Definition A.15** (Expected value). If  $X$  is a random variable with PDF  $f$ , we define its expected value as

$$\mathbb{E}(X) \stackrel{\text{def}}{=} \int_{\mathbb{R}} x \cdot f(x)dx.$$

**Definition A.16** (Bernoulli Random Variable). A (discrete) random variable  $X$  is Bernoulli distributed with parameter  $p \in (0, 1)$ , denoted  $X \sim \text{BER}(p)$ , if it has the distribution

$$f(x; p) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0. \end{cases}$$

**Definition A.17** (Uniform Distribution). A random variable  $X$  is uniformly distributed on the interval  $[a, b]$ , denoted  $X \sim \text{UNI}[a, b]$ , if it has the PDF

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

**Definition A.18** (Exponential Random Variable). A random variable  $X$  is exponentially distributed with parameter  $\lambda$ , denoted  $X \sim \text{EXP}(\lambda)$ , if it has the PDF

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

**Definition A.19** ( $\chi^2$  Random Variable). A random variable is  $\chi^2$  distributed with  $n$  degrees of freedom, if it has the PDF

$$f_n(x) = \begin{cases} \frac{x^{\frac{n}{2}-1} e^{-\frac{x}{2}}}{2^{\frac{n}{2}} \Gamma(\frac{n}{2})} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}, \text{ where } \Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt.$$

**Proposition A.20.** Let  $X \sim \text{EXP}(\lambda)$  and let  $x_1, \dots, x_n$  be realizations of  $X$ . For  $\alpha \in (0, 1)$ , the  $1 - \alpha$  confidence interval of  $\lambda$  based on its realisations is then given by

$$\left[ \frac{\chi_{\alpha/2, 2n}^2}{2 \sum_{i=1}^n x_i}, \frac{\chi_{1-\alpha/2, 2n}^2}{2 \sum_{i=1}^n x_i} \right]$$

*Proof.* We first prove: If  $X$  follows an exponential distribution with parameter  $\lambda$ , then  $Y = 2\lambda X$  follows an exponential distribution with parameter  $\frac{1}{2}$ . The density function of  $X$  is  $f(x | \lambda) = \lambda e^{-\lambda x}$  if  $x > 0$  and 0 otherwise. It is easy to see that the density function of  $Y$  is then given as  $g(y) = \frac{1}{2} e^{-\frac{y}{2}}$ . Note that the exponential distribution with parameter  $\frac{1}{2}$  is equivalent to  $\chi_2^2$  (Definition A.19).

For  $X_i \sim \text{EXP}(\lambda)$  with  $i = 1, \dots, n$  we define

$$h(X_1, \dots, X_n, \lambda) \stackrel{\text{def}}{=} 2\lambda \sum_{i=1}^n X_i = \sum_{i=1}^n Y_i.$$

Each  $Y_i$  follows a  $\chi_2^2$  distribution, and they are independent. Therefore  $h$  follows a  $\chi_{2n}^2$  distribution. Let  $\chi_{\alpha/2, 2n}^2$  and  $\chi_{1-\alpha/2, 2n}^2$  be the  $\alpha/2$  and the  $1 - \alpha/2$  quantiles of the  $\chi_{2n}^2$  distribution, then

$$P \left( \chi_{\alpha/2, 2n}^2 \leq 2\lambda \sum_{i=1}^n X_i \leq \chi_{1-\alpha/2, 2n}^2 \right) = 1 - \alpha.$$

Rearranging yields

$$P \left( \frac{\chi_{\alpha/2, 2n}^2}{2 \sum_{i=1}^n X_i} \leq \lambda \leq \frac{\chi_{1-\alpha/2, 2n}^2}{2 \sum_{i=1}^n X_i} \right) = 1 - \alpha.$$

□

## A.2 Statistical Hypothesis Testing

Statistical hypothesis testing is concerned with the observation, and realization of random variables. The intention of a statistical test is to find enough evidence to formally *reject* a conjecture, commonly called the *null hypothesis*. Intuitive examples of null hypothesis in practice are given in the conjecture that people of a certain nationality have an average body height of 1.68m, or that the average grade of a school class is 75%. The decision of rejection or acceptance is based on concrete realizations of the random variables summarized as the *sample*. Should the null hypothesis be rejected, we conclude that it is false. A common pitfall lies in the reverse: Accepting a hypothesis does not imply its correctness, but rather the lack of evidence to prove it wrong.

Mathematically, we utilize a mapping from a sample to a binary verdict: accept or reject. Different domains and application areas require a wide selection of concrete mappings, called the *test statistic*. Test statistics vary with the



representation of the underlying sample data. Sample data can be categorized into *quantitative* or *qualitative* data. We refer to Figure A.1 for an illustration. Quantitative data comprises 1. nominal data, like the preference of political parties within a population, 2. ordinal data, like school grades, or 3. binary data, like yes and no, or pass and fail. On the other hand, qualitative data encompasses 1. discrete data, like the number of damaged parts in a shipment, or 2. continuous data, like the body height of a person.

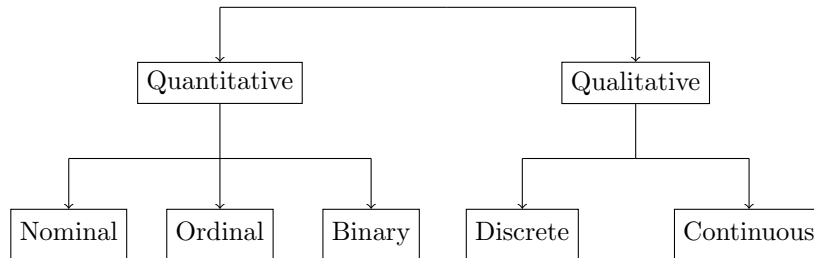


Figure A.1: Qualitative and quantitative classification of sample data.

In this section we recall two statistical testing methods alongside two test statistics, which are utilized within this thesis. We refer the avid reader to graduate textbooks for a more detailed approach on the subject, see e.g., [156]. For a recent survey on hypothesis testing in statistical model checking we refer to [151].

A common template of statistical hypothesis tests encompasses the quantities:

1. A formal statement of the null hypothesis  $H_0$ , such as: the mean of a random variable is equal to  $\mu$ ,
2. a mathematical statement of the alternative hypothesis  $H_A$ , such as: the mean of the random variable is not equal to  $\mu$ ,
3. the level of significance  $\alpha \in (0, 1)$ , a.k.a. probability to perform an *error of first kind* or *type I error*,
4. the probability to perform an *error of second kind*  $\beta \in (0, 1)$  or *type II error*, and
5. the sample size  $N$ .

### A.2.1 Statistical errors.

The goal of hypothesis testing is to exhibit control over the probability of wrong decisions. Since observations are based on random variables, there is no guarantee that the decision made is correct. Therefore, statistical tests always encompass two errors; the errors of first or type I error, and error of second kind or type II error. The first describes the probability to reject a true null hypothesis, while the latter entails the probability to accept a wrong null hypothesis.

These properties are quantified in the parameters  $\alpha$  and  $\beta$  respectively. Figure A.2 summarizes the probabilities with which each decision is made. The value  $1 - \beta$  is often referred to as the *power*  $p$  of a statistical test. It comprises the ability to correctly reject the null hypothesis. The value  $\beta$  and the power  $p$  are occasionally swapped in the literature.

	Hypothesis $H_0$ true	Alternative $H_A$ true
$H_0$ accepted	$1 - \alpha$ Correct Decision	$\beta$ Type II Error
$H_0$ rejected	$\alpha$ Type I Error	$1 - \beta$ Correct Decision

Figure A.2: Quantification of possible statistical errors. The statistical *error of first kind* is denoted by  $\alpha$ , while the statistical *error of second kind* is denoted by  $\beta$ . The value  $1 - \beta$  is frequently called the test power  $p$ .

### A.2.2 Two Types of Hypotheses Tests

It is inherently impossible to exhibit control over all three parameters  $\alpha$ ,  $\beta$  and  $N$  simultaneously. Defining values for both  $\alpha$  and  $\beta$  requires a certain sample size  $N$ . On the contrary, having a sample of size  $N$ , one can only exhibit control over  $\alpha$ , and  $\beta$  adjusts to that. Consequently, there are two fundamentally different approaches of statistical testing: 1. Sequential probability ratio testing, where both errors  $\alpha$  and  $\beta$  are fixed, and 2. Fixed sample size testing, where the error of first kind  $\alpha$  and sample size  $N$  are fixed. We introduce both shortly in the following and refer to Figure A.3 for a schematic illustration.

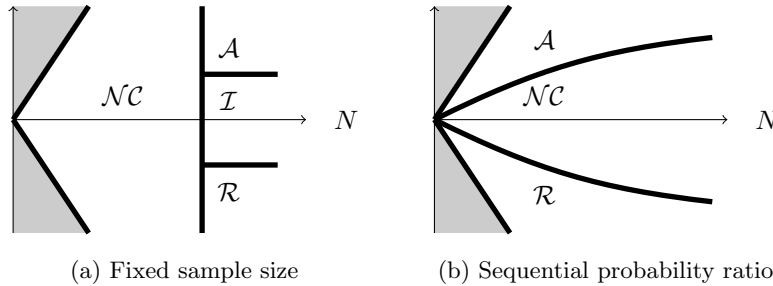


Figure A.3: Schematic visualisation of test decisions and acceptance regions for fixed sample size testing versus sequential ratio testing after [151].  $\mathcal{A}$  marks acceptance,  $\mathcal{R}$  marks rejection,  $\mathcal{I}$  marks inconclusiveness, and  $\mathcal{NC}$  marks the non-critical region. The unlabelled ordinate represents values of the test observation. Fixed sample size tests draw a conclusion as soon as they hit the boundary in  $N$ , while sequential ratio tests stay in a non-critical region, as long as they do not hit either the  $\mathcal{A}$  boundary, or the  $\mathcal{R}$  boundary.

**Sequential probability ratio testing** The sequential probability ratio test (SPRT), or sequential analysis is commonly attributed to Abraham Wald et. al [189]. Its key feature is the specification of  $\alpha$  and  $\beta$ , which leaves  $N$  variable. SPRT investigates a sample  $x_i$  of a random variable  $X$  with distribution  $f(x_i; \theta)$ , where  $\theta$  is unknown. The null-hypothesis is  $H_0 : \theta = \theta_0$ , whereas the alternative hypothesis is given as  $H_1 : \theta = \theta_1$ . The test exhibits its control over both  $\alpha$  and  $\beta$ , i.e. the error of first kind is not supposed to be greater than  $\alpha$ , and the error of second kind is not supposed to be greater than  $\beta$ . This requires the sample size  $N$  to be variable.

The intuition is as follows: for a given sample  $x_1, \dots, x_N$  the test calculates the cumulative sum of the log-likelihood ratio.

$$S_N \stackrel{\text{def}}{=} S_{N-1} + \frac{\log(f(x_i; \theta_1))}{\log(f(x_i; \theta_2))}$$

If  $S_N$  is above a certain threshold  $A$ , we accept  $H_1$ . However, if  $S_N$  is below a certain threshold  $B$ , we accept  $H_0$ . The test is inconclusive if neither of these cases hold, and another observation  $x_{N+1}$  is added. The decisive feature lies in the choice of  $A$  and  $B$  to guarantee the maximality of errors as  $\alpha$  and  $\beta$ . This is given for

$$A = \frac{1 - \beta}{\alpha} \text{ and } B = \frac{\beta}{1 - \alpha}.$$

Controlling both errors simultaneously makes the test flexible and efficient. However, its applicability is often hindered by the fact that new observations are not available. E.g. if statistical analysis is performed on a given data set, there might be no way to acquire additional samples, or if a certain population is exhaustively studied, additional observations might not be available at all.

**Fixed sample size testing** A far more commonly applied hypothesis test is given in the fixed sample size test. Here, the eponymous sample size  $N$  is fixed and a level of significance  $\alpha$  is chosen. The test establishes critical regions based on the null-hypothesis, the test statistic and  $\alpha$ . Then, a *test observation* is calculated based on the sample. The precise calculations depend on the test statistic, e.g.  $\chi^2$ ,  $t$ -value, etc. Acceptance of the null-hypothesis is determined depending whether or not the test observation lies in the critical region. Naturally, an error of second kind cannot be avoided, and statistical tests aim at being optimal with respect to a minimal  $\beta$ .

The general course of actions is as follows:

1. The hypothesis  $H_0$ , and its alternative  $H_1$  are phrased. A level of significance  $\alpha \in (0, 1)$  is set and possibly other parameters are chosen.
2. A sample is drawn.
3. The test statistic is chosen and critical regions are determined.
4. The test observation is calculated.
5. A conclusion is made:  $H_1$  is accepted if the test observation lies in the critical region, and  $H_0$  is accepted if not.

Fixed sample size tests can be *parametric* or *non-parametric*. The null hypothesis of the first entails a test for a parameter like a distribution mean or its variance. Non-parametric tests use different hypotheses; the null-hypothesis tests whether two distributions are equal *independent from individual parameters*.

**Application in this thesis.** While the SPRT seems appealing in the context of a probabilistic MBT framework, since new observations are readily available upon additional test executions, we opt for fixed sample sizes instead. The models considered in this thesis contain non-deterministic choices whose resolution depends on schedulers – They are thus initially not fully probabilistic. Thus, analysing whether a collected sample of traces can be attributed to a certain model, requires finding a *fitting scheduler* first. This makes SPRT not directly applicable, as every new observation requires new calculations with respect to schedulers. In addition, note that SPRT are parametric tests, whereas the used hypotheses tests in the thesis are non-parametric. Hence, the hypothesis tests used in this thesis are of fixed sample size.

We introduce the non-parametric fixed sample size tests used in this thesis in the following. We point out, that the intent of the hypothesis tests in both cases is to infer whether a sample belongs to a given probability distribution. Therefore, we apply non-parametric two-sided statistical tests.

### A.2.3 Pearson’s $\chi^2$ Test

Pearson’s  $\chi^2$  test is a non-parametric test statistic applicable for nominal data [2] dating back to Karl Pearson [145]. It is one of many  $\chi^2$  tests whose results are evaluated with respect to the  $\chi^2$  distribution (Definition A.19). Its purpose in the context of this thesis is to test the null-hypothesis that observed frequencies of a sample are consistent with a theoretical distribution. In particular, it used in Chapters 4, 5 and 7, to assess whether observations made on a black-box trace machine are consistent with a formal specification model. We point out, that Pearson’s  $\chi^2$  test has other applications, e.g. to test homogeneity of two samples, or a test of independence of two variables.

To apply the test, the underlying data has to be distinguishable in  $n$  different groups such that the probability of all categories in the theoretical distribution adds up to 1, e.g. numbers on a six sided die, or the preference of a political party within a given population. The test statistic  $\chi^2$  is calculated as the normalized accumulated squared deviation from a sample to its expectations, i.e.

$$\chi^2 \stackrel{\text{def}}{=} \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i},$$

where  $O_i$  is the number of observations of type  $i$ ,  $E_i$  is the expected number of type  $i$ , and  $n$  is the total number of categories. The empirical test observation  $\chi^2$  is then compared to critical values  $\chi_{crit}^2 \stackrel{\text{def}}{=} \chi_{\alpha, d}^2$  to determine acceptance of the null hypothesis. Here  $\alpha$  is the level of significance, and  $d$  are the degrees of freedom (d.f.). The degrees of freedom refer to the amount of categories that are

independent of each other. A six sided die, for instance, has 5 degrees of freedom, as the sixth value is immediately clear, if one knows the all the others. The critical values represent the  $1 - \alpha$ -percentiles of the  $\chi^2$  distribution of  $d$  degrees of freedom. That is, the probability mass in the percentile  $(-\infty, \chi_{\alpha, d}^2)$  contains  $1 - \alpha$  of the total probability mass of the probability density function. These values can be calculated according to Definition A.19. Values for commonly used assignments of  $\alpha$  and  $d$  are documented in lookup tables, e.g. [2]. For the convenience of the reader, we included one in Table A.2.

**Example A.21.** *Suppose we want to infer whether a given sample suggests a fair die for  $\alpha = 0.05$ . The straightforward categorisation is then given in the numbers 1 to 6 where each number occurs with probability  $\frac{1}{6}$ . Table A.1 depicts*







							Total
Observation	14	18	16	20	20	12	100
Expectation	$16.\bar{6}$	$16.\bar{6}$	$16.\bar{6}$	$16.\bar{6}$	$16.\bar{6}$	$16.\bar{6}$	100

Table A.1: Example observation and expectations of a six sided die.

a potential sample of size  $m$ . We calculate the empirical  $\chi^2$  score as

$$\begin{aligned} \chi^2 &= \frac{(14 - 16.\bar{6})^2}{16.\bar{6}} + \frac{(18 - 16.\bar{6})^2}{16.\bar{6}} + \frac{(16 - 16.\bar{6})^2}{16.\bar{6}} \\ &+ \frac{(20 - 16.\bar{6})^2}{16.\bar{6}} + \frac{(20 - 16.\bar{6})^2}{16.\bar{6}} + \frac{(12 - 16.\bar{6})^2}{16.\bar{6}} \\ &= 3.2. \end{aligned}$$

The experiment has 5 degrees of freedom, as the 6-th value is immediately given, if one knows the remaining 5. The critical value for  $d = 5$  and  $\alpha = 0.05$  is given as  $\chi_{0.05, 5}^2 = 11.07$ , cf. Table A.2. Since the empirical value does not exceed this threshold, we accept the hypothesis, that the sample was drawn from a fair die.

#### A.2.4 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test (KS-test) is a non-parametric hypothesis test [77] named after Andrey Kolmogorov and Nikolai Smirnov. Its purpose is to decide whether a sample comes from a population with a specific distribution (1-sample KS-test), or to test whether two samples were drawn from the same population (2-sample KS-test). In this thesis, we utilize the former. In particular, it is used to test whether recorded time stamps belong to a specified continuous distribution over real time in Chapters 5 and 7.

Suppose we gathered a sample  $x_1, \dots, x_n$  of  $n$  independently and identically distributed random variables  $X_i$ , where the  $x_i$  are assumed to be ordered increasingly. The KS-test compares the step function induced by the values  $x_i$

d.f. $m$	Probability $\alpha$							
	0.99	0.975	0.95	0.9	0.1	0.05	0.025	0.01
1	-	0.001	0.004	0.016	2.706	3.841	5.024	6.635
2	0.020	0.051	0.103	0.211	4.605	4.991	7.378	9.210
3	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345
4	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277
5	0.554	0.831	1.145	1.610	9.236	11.071	12.833	15.086
6	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812
7	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475
8	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090
9	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666
10	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209
11	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725
12	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217
13	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688
14	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141
15	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.578
16	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000
17	6.408	7.564	8.672	10.085	24.769	27.587	30.191	33.409
18	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805
19	7.633	8.907	10.117	11.651	27.204	30.144	32.852	36.191
20	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566
21	8.897	10.283	11.591	13.240	29.615	32.671	35.479	38.932
22	9.542	10.982	12.338	14.042	30.813	33.924	36.781	40.289
23	10.196	11.689	13.091	14.848	32.007	35.172	38.076	41.638
24	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980
25	11.524	13.120	14.611	16.473	34.382	37.652	40.646	44.314
26	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642
27	12.879	14.573	16.151	18.114	36.741	40.113	43.194	46.963
28	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278
29	14.257	16.047	17.708	19.768	39.087	42.557	45.722	49.588
30	14.954	16.791	18.493	20.599	40.256	43.773	46.979	50.892
40	22.164	24.433	26.509	29.051	51.805	55.758	59.342	63.691
50	29.707	32.357	34.764	37.689	63.167	67.505	71.420	76.154
60	37.485	40.482	43.188	46.459	74.397	79.082	83.298	88.379
70	45.442	48.758	51.739	55.329	85.527	90.531	95.023	100.425
80	53.540	57.153	60.391	64.278	96.578	101.879	106.629	112.329
90	61.754	65.647	69.126	73.291	107.565	113.145	118.136	124.116
100	70.065	74.222	77.929	82.358	118.498	124.342	129.561	135.807

Table A.2: Percentiles  $\chi_{\alpha,m}^2$  of the  $\chi^2$  distribution [2].

to the supposed underlying CDF (Definition A.14)  $F$ . The step function, or empirical distribution function (EDF) for  $n$  values, denoted  $F_n$ , is given as

$$F_n(x) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } x < x_1 \\ \frac{n_k}{n}, & \text{if } x_k \leq x < x_{k+1}, \\ 1, & \text{if } x > x_n \end{cases}$$

where  $n_k$  is the multiplicity of element  $x_k$ . The distance of  $F$  to  $F_n$  is called the  $n$ -th Kolmogorov-Smirnov statistic, i.e.

$$K_n = \sup_x |F_0(x) - F_n(x)|.$$

The Glivenko-Cantelli theorem [77] guarantees that  $K_n \xrightarrow{n \rightarrow \infty} 0$ , if the sample was indeed drawn from the distribution  $F$ . The Kolmogorov-Smirnov statistic is then compared to the critical values  $K_{\alpha,n}$ . A null-hypothesis is rejected iff  $K_n > K_{\alpha,n}$ . The critical values  $K_{\alpha,n}$  depend on the level of significance  $\alpha$ , and are the  $1 - \alpha$ -percentiles of the  $n$ -th Kolmogorov distribution. To the best of our knowledge there is no explicit method to calculate the distribution of  $K_n$ , but [191] provide a procedure to calculate  $P(K_n < x)$  with 13-15 digits accuracy for  $n$  ranging from 2 to at least 16.000. Its limiting distribution is given as

$$\lim_{n \rightarrow \infty} P(\sqrt{n}K_n \leq x) = 1 - 2 \sum_{i=1}^{\infty} (-1)^{i-1} e^{-2i^2 x^2}.$$

However, for all practical intents and purposes, values for  $n \leq 40$  exist in lookup tables, e.g. [139], and there is an approximation formula for  $n > 40$  in

$$K_{\alpha,n} \approx \frac{\sqrt{-0.5 \ln(\alpha/2)}}{\sqrt{n}}.$$

For the convenience of the reader, we included a reference table in Table A.3.

**Example A.22.** Assume given are  $t_1 = 0.26, t_2 = 0.33, t_3 = 0.55, t_4 = 0.77, t_5 = 1.18, t_6 = 1.41, t_7 = 1.46, t_8 = 1.97$  as sample data. We want to infer whether the the sample is consistent with a uniform distribution  $\text{UNI}[0,2]$ . The step function of the  $t_i$  for these values is given as

$$F_8(x) = \begin{cases} 0 & t < t_0 \\ \frac{k}{8} & t_k < x \leq t_{k+1}, k = 1, \dots, 7 \\ 1 & t \geq t_8 \end{cases}$$

$K_8 = 0.145$  is the distance between the EDF of the  $t_i$  and  $\text{UNI}[0,2]$ . The critical value of the Kolmogorov distribution for  $n = 8$  and  $\alpha = 0.05$  is  $K_{\text{crit}} = 0.454$ , cf. Table A.3. With  $K_8 < K_{\text{crit}}$ , the empiric value is below the given threshold. Hence, we accept the null-hypothesis that the data is consistent with  $\text{UNI}[0,2]$

n	Probability $\alpha$				n	Probability $\alpha$			
	0.1	0.05	0.02	0.01		0.1	0.05	0.02	0.01
1	0.950	0.975	0.990	0.995	21	0.259	0.287	0.321	0.344
2	0.776	0.842	0.900	0.929	22	0.253	0.281	0.314	0.337
3	0.636	0.708	0.785	0.829	23	0.247	0.275	0.307	0.330
4	0.566	0.624	0.689	0.734	24	0.242	0.270	0.301	0.323
5	0.509	0.563	0.627	0.669	25	0.238	0.264	0.295	0.317
6	0.468	0.520	0.577	0.617	26	0.233	0.260	0.290	0.310
7	0.436	0.483	0.538	0.576	27	0.229	0.254	0.284	0.305
8	0.410	0.454	0.507	0.542	28	0.225	0.250	0.279	0.300
9	0.387	0.430	0.480	0.513	29	0.221	0.246	0.274	0.295
10	0.369	0.410	0.457	0.489	30	0.218	0.242	0.270	0.290
11	0.352	0.391	0.437	0.468	31	0.214	0.238	0.266	0.285
12	0.340	0.375	0.419	0.449	32	0.211	0.234	0.262	0.281
13	0.330	0.361	0.403	0.432	33	0.208	0.231	0.258	0.277
14	0.314	0.349	0.390	0.418	34	0.205	0.227	0.254	0.273
15	0.304	0.338	0.377	0.404	35	0.202	0.224	0.260	0.270
16	0.295	0.327	0.366	0.392	36	0.199	0.221	0.247	0.265
17	0.286	0.318	0.355	0.381	37	0.196	0.218	0.244	0.262
18	0.279	0.310	0.346	0.371	38	0.194	0.215	0.240	0.258
19	0.271	0.301	0.337	0.361	39	0.192	0.213	0.238	0.255
20	0.265	0.294	0.329	0.352	40	0.189	0.210	0.235	0.252

Table A.3: Percentiles  $K_{\alpha,n}$  of the  $n$ -th Kolmogorov statistic [139].

### A.2.5 Accumulation of Type I Errors

A level of significance  $\alpha \in (0, 1)$  limits type 1 error by  $\alpha$ . Performing several statistical experiments on the same sample inflates this probability: if one experiment is performed at  $\alpha = 0.05$ , there is a 5% probability to incorrectly reject a true hypothesis. Performing 100 experiments, we expect to see a type 1 error 5 times. If all experiments are independent, the chance is thus 99.4%. This probability is known as the *family wise error rate* (FWER) [172]. The FWER becomes relevant in Chapters 5 and 7, where one  $\chi^2$  test, as well as multiple hypothesis tests for each delayed transition are performed.

There are generally two approaches to control this inflation: *single step* and *sequential* adjustments. The first evenly distributes the probability to perform a Type I error, while the latter adjusts it sequentially for every additional hypothesis. Both aim at limiting the *global* type I error in the statistical testing process by adjusting the *local* level of significance. Note that this correction comes at the cost of globally increasing  $\beta$ , thus decreasing the statistical power  $p$  of a hypothesis test. We point out that there is no *best* correction, and usage may vary with the number of hypotheses and other factors.



**Bonferroni correction.** The most prevalent example for single step adjustment is Bonferroni correction [172]. The method is straightforward: the corrected  $\alpha_{local}$  is evenly distributed for all  $m$  hypotheses, i.e.

$$\alpha_{local} = \frac{\alpha_{global}}{m}$$

**Šidák correction.** Another single step adjustment is given in the Šidák correction [172]. It is shown to be slightly more powerful than Bonferroni correction [1]. If the total number of hypothesis tests to be performed is  $m$ , then

$$\alpha_{local} = 1 - (1 - \alpha_{global})^{1/m}.$$

**Holm-Bonferroni correction.** A sequential procedure is given in the Holm-Bonferroni correction [172]. It is shown to be more powerful than Bonferroni correction [3]. Rather than distributing  $\alpha$  evenly, it is adjusted for every additional hypothesis test performed. That is, for hypotheses  $H_1, \dots, H_m$ , let  $p_1, \dots, p_m$  be their corresponding power. The hypotheses are then ordered  $H_{(1)}, \dots, H_{(m)}$  according to their increasingly ordered powers  $p_{(1)}, \dots, p_{(m)}$ . Then hypotheses  $H_{(1)}, \dots, H_{(k-1)}$  are rejected, and  $H_{(k)}, \dots, H_{(m)}$  are not, where  $k$  is the smallest index such that

$$p_{(k)} > \frac{\alpha_{global}}{m + 1 - k}.$$

All three corrections ensure the FWER to be smaller or equal to  $\alpha_{global}$ . For more correction methods we refer the reader to [172].



# APPENDIX B

---

## Publications by the Author

---

- Marcus Gerhold and Mariëlle Stoelinga. ioco theory for probabilistic automata. In *Proceedings of the 10th Workshop on Model Based Testing, MBT*, pages 23–40, 2015,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE*, pages 251–268, 2016,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of stochastic systems with ioco theory. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, A-TEST*, pages 45–51, 2016,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems with stochastic time. In *Proceedings of the 11th International Conference on Tests and Proofs, TAP*, pages 77–97, 2017,
- Pedro R. D’Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. A hierarchy of scheduler classes for stochastic automata. In *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FOSSACS*, pages 384–402, 2018,
- Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. *Formal Aspects of Computing*, 30(1):77–106, 2018,
- Marcus Gerhold, Arnd Hartmanns, and Mariëlle Stoelinga. Model-based testing for general stochastic time. In *Proceedings of the 10th International Symposium on NASA Formal Methods, NFM*, pages 203–219, 2018.



---

## Bibliography

---

- [1] Hervé Abdi. Bonferroni and šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics*, 3:103–107, 2007.
- [2] Alan Agresti. *An introduction to categorical data analysis*, volume 135. Wiley New York, 1996.
- [3] Mikel Aickin and Helen Gensler. Adjusting for multiple testing when reporting research results: the bonferroni vs holm methods. *American journal of public health*, 86(5):726–728, 1996.
- [4] Jamal N. Al-Karaki and Ahmed E. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communication*, 11(6):6–28, 2004.
- [5] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for probabilistic real-time systems (extended abstract). In *ICALP*, volume 510 of *LNCS*, pages 115–126. Springer, 1991.
- [6] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.
- [7] Todd R. Andel and Alec Yasinsac. On the credibility of MANET simulations. *IEEE Computer*, 39(7):48–54, 2006.
- [8] Alberto Avritzer, Laura Carnevali, Hamed Ghasemieh, Lucia Happe, Boudewijn R. Haverkort, Anne Kozirolek, Daniel S. Menasché, Anne Remke, Sahra Sedigh Sarvestani, and Enrico Vicario. Survivability evaluation of gas, water and electricity infrastructures. *Electronic Notes in Theoretical Computer Science*, 310:5–25, 2015.
- [9] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- [10] Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for markov chains. *Inf. Comput.*, 200(2):149–214, 2005.

- 
- [11] Paolo Ballarini, Nathalie Bertrand, András Horváth, Marco Paolieri, and Enrico Vicario. Transient analysis of networks of stochastic timed automata using stochastic state classes. In *QEST*, volume 8054 of *LNCS*, pages 355–371. Springer, 2013.
- [12] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. The Coq proof assistant reference manual: Version 6.1. Technical report, Inria, 1997.
- [13] Mark A. Beaumont, Wenyang Zhang, and David J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.
- [14] Joachim Behar, Aoife Roebuck, Joao S. Domingos, Elnaz Gederi, and Gari D. Clifford. A review of current sleep screening applications for smartphones. *Physiological measurement*, 34(7):R29, 2013.
- [15] Axel Belinfante. *JTorX: exploring model-based testing*. PhD thesis, University of Twente, Enschede, Netherlands, 2014.
- [16] Matthias Beyer and Winfried Dulz. Scenario-based statistical testing of quality of service requirements. In *Revised Selected Papers from the International Workshop on Scenarios: Models, Transformations and Tools*, pages 152–173, 2003.
- [17] Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krcál, Gilles Nies, and Marvin Stenger. Battery-aware scheduling in low orbit: The GomX-3 case. In *FM*, volume 9995 of *LNCS*, pages 559–576. Springer, 2016.
- [18] Nikolaž Bjørner and Frank S. de Boer, editors. *Proceedings of the 20th International Symposium on Formal Methods, FM*, volume 9109 of *LNCS*. Springer, 2015.
- [19] Stefan Blom and Marieke Huisman. The VerCors tool for verification of concurrent programs. In *International Symposium on Formal Methods*, pages 127–131. Springer, 2014.
- [20] Barry W. Boehm. *Software Engineering Economics*, pages 99–150. Springer Berlin Heidelberg, 2001.
- [21] Henrik Bohnenkamp and Axel Belinfante. Timed testing with TorX. In *Formal Methods Europe*, volume 3582 of *LNCS*, pages 173–188. Springer, 2005.
- [22] Henrik C. Bohnenkamp, Pedro R. D’Argenio, Holger Hermanns, and Joost-Pieter Katoen. MoDeST: A compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Software Eng.*, 32(10):812–830, 2006.

- [23] Frank Böhr. Model-based statistical testing of embedded systems. In *IEEE 4th Intl. Conf. on Software Testing, Verification and Validation*, pages 18–25, 2011.
- [24] Frank Böhr. *Model-Based Statistical Testing of Embedded Real-Time Software with Continuous and Discrete Signals in a Concurrent Environment: The Usage Net Approach*. PhD thesis, University of Kaiserslautern, 2012.
- [25] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and logic*. Cambridge university press, 2002.
- [26] Steven Borowiec. AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol. *The Guardian*, 15, 2016.
- [27] Robert H. Bourdeau and Betty H. C. Cheng. A formal semantics for object model diagrams. *IEEE Trans. Software Eng.*, 21(10):799–821, 1995.
- [28] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability and performance analysis of extended AADL models. *Comput. J.*, 54(5):754–775, 2011.
- [29] Laura Brandán Briones and Ed Brinksma. A test generation framework for quiescent real-time systems. In *4th International Workshop, FATES*, pages 64–78, 2004.
- [30] Mario Bravetti and Pedro R. D’Argenio. Tutte le algebre insieme: Concepts, discussions and relations of stochastic process algebras with general distributions. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 44–88. Springer, 2004.
- [31] Mario Bravetti and Roberto Gorrieri. The theory of interactive generalized semi-Markov processes. *Theor. Comput. Sci.*, 282(1):5–32, 2002.
- [32] Tomás Brázdil, Holger Hermanns, Jan Krcál, Jan Kretínský, and Vojtech Reháč. Verification of open interactive Markov chains. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 474–485, 2012.
- [33] Tomás Brázdil, Jan Krcál, Jan Kretínský, and Vojtech Reháč. Fixed-delay events in generalized semi-Markov processes revisited. In *CONCUR*, volume 6901 of *LNCS*, pages 140–155. Springer, 2011.
- [34] Jeremy Bryans, Howard Bowman, and John Derrick. Model checking stochastic automata. *ACM Trans. Comput. Log.*, 4(4):452–492, 2003.
- [35] Peter Buchholz, Jan Kriege, and Dimitri Scheftelowitsch. Model checking stochastic automata for dependability and performance measures. In *DSN*, pages 503–514. IEEE Computer Society, 2014.

- 
- [36] Yuliya Butkova, Hassan Hatefi, Holger Hermanns, and Jan Krcál. Optimal continuous time Markov decisions. In *ATVA*, volume 9364 of *LNCS*, pages 166–182. Springer, 2015.
- [37] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [38] Stefano Cattani. *Trace-Based Process Algebras for Real-Time Probabilistic Systems*. PhD thesis, University of Birmingham, 2005.
- [39] Ling Cheung, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Switched PIOA: parallel composition via distributed scheduling. *Theor. Comput. Sci.*, 365(1-2):83–108, 2006.
- [40] Ling Cheung, Mariëlle Stoelinga, and Frits W. Vaandrager. A testing scenario for probabilistic processes. *J. ACM*, 54(6), 2007.
- [41] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 287–302. Springer, 2009.
- [42] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE transactions on software engineering*, (3):178–187, 1978.
- [43] Duncan Clarke, Thierry Jéron, Vlad Rusu, and Elena Zinovieva. STG: A symbolic test generation tool. In *8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 470–475, 2002.
- [44] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, December 1996.
- [45] Rance Cleaveland, Zeynep Dayar, Scott A. Smolka, and Shoji Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93 – 148, 1999.
- [46] Donald L. Cohn. *Measure Theory*. Birkhäuser.
- [47] William J. Conover. A Kolmogorov goodness-of-fit test for discontinuous distributions. *Journal of the American Statistical Association*, 67(339):591–596, 1972.
- [48] Pedro R. D’Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. A hierarchy of scheduler classes for stochastic automata. In *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FOSSACS*, pages 384–402, 2018.



- [49] Pedro R. D’Argenio, Arnd Hartmanns, Axel Legay, and Sean Sedwards. Statistical approximation of optimal schedulers for probabilistic timed automata. In *iFM*, volume 9681 of *LNCS*, pages 99–114. Springer, 2016.
- [50] Pedro R. D’Argenio and Joost-Pieter Katoen. A theory of stochastic systems part I: stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.
- [51] Pedro R. D’Argenio, Matias David Lee, and Raúl E. Monti. Input/output stochastic automata. In *FORMATS*, volume 9884 of *LNCS*, pages 53–68. Springer, 2016.
- [52] Pedro R. D’Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of Markov decision processes. *STTT*, 17(4):469–484, 2015.
- [53] Mohammad Torabi Dashti and David A. Basin. Tests and refutation. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis, ATVA*, pages 119–138, 2017.
- [54] Luca de Alfaro. The verification of probabilistic systems under memoryless partial-information policies is hard. Technical report, DTIC Document, 1999.
- [55] Rocco De Nicola. Extensional equivalences for transition systems. *Acta Inf.*, 24(2):211–237, 1987.
- [56] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
- [57] René G. de Vries, Axel Belinfante, and Jan Feenstra. Automated testing in practice: The highway tolling system. In *Proceedings of the 14th International Conference on Testing Communicating Systems*, pages 219–234, 2002.
- [58] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification, CAV*, pages 592–600. Springer, 2017.
- [59] Yuxin Deng and Matthew Hennessy. On the semantics of Markov automata. *Information and Computation*, 222:139–168, 2013.
- [60] Yuxin Deng, Matthew Hennessy, Rob J. van Glabbeek, and Carroll Morgan. Characterising testing preorders for finite probabilistic processes. *CoRR*, 2008.
- [61] Yuxin Deng, Rob J. van Glabbeek, Matthew Hennessy, and Carroll Morgan. Testing finitary probabilistic processes. In *Proceedings of the 20th International Conference on Concurrency Theory, CONCUR*, pages 274–288, 2009.

- 
- [62] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: A systematic review. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering, ASE, WEASEL Tech*, pages 31–36. ACM, 2007.
- [63] Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. An improved conformance testing method. In *Proceedings of the 25th International Conference on Formal Techniques for Networked and Distributed Systems*, pages 204–218, 2005.
- [64] Marie Duflot, Marta Kwiatkowska, Gethin Norman, and David Parker. A formal analysis of bluetooth device discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.
- [65] Winfried Dulz and Fenhua Zhen. MaTeLo - statistical usage testing by annotated sequence diagrams, markov chains and TTCN-3. In *3rd International Conference on Quality Software, QSIC*, pages 336–342, 2003.
- [66] Christian Eisentraut, Jens Chr. Godskesen, Holger Hermanns, Lei Song, and Lijun Zhang. Probabilistic bisimulation for realistic schedulers. In *Proceedings of the 20th International Symposium on Formal Methods, FM*, pages 248–264, 2015.
- [67] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351. IEEE Computer Society, 2010.
- [68] Juhan P. Ernits, Andres Kull, Kullo Raiend, and Jüri Vain. Generating tests from EFSM models using guided model checking and iterated search refinement. In *Revised Selected Papers of the First Combined International Workshops on Formal Approaches to Software Testing and Runtime Verification, FATES and RV*, pages 85–99, 2006.
- [69] Loe M. G. Feijs, Nicolae Goga, and Sjouke Mauw. Probabilities in the TorX test derivation algorithm. In *SAM, 2nd Workshop on SDL and MSC*, pages 173–188, 2000.
- [70] Yuan Feng and Lijun Zhang. When equivalence and bisimulation join forces in probabilistic automata. In *19th International Symposium on Formal Methods, FM*, pages 247–262, 2014.
- [71] Marcus Gerhold, Arnd Hartmanns, and Mariëlle Stoelinga. Model-based testing for general stochastic time. In *Proceedings of the 10th International Symposium on NASA Formal Methods, NFM*, pages 203–219, 2018.

- [72] Marcus Gerhold and Mariëlle Stoelinga. ioco theory for probabilistic automata. In *Proceedings of the 10th Workshop on Model Based Testing, MBT*, pages 23–40, 2015.
- [73] Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE*, pages 251–268, 2016.
- [74] Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of stochastic systems with ioco theory. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation, A-TEST*, pages 45–51, 2016.
- [75] Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems with stochastic time. In *Proceedings of the 11th International Conference on Tests and Proofs, TAP*, pages 77–97, 2017.
- [76] Marcus Gerhold and Mariëlle Stoelinga. Model-based testing of probabilistic systems. *Formal Aspects of Computing*, 30(1):77–106, 2018.
- [77] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric Statistical Inference*. Springer, 2011.
- [78] Elizabeth Gibney. Google AI algorithm masters ancient game of Go. *Nature News*, 529(7587):445, 2016.
- [79] Sergio Giro and Pedro R. D’Argenio. Quantitative model checking revisited: Neither decidable nor approximable. In *FORMATS*, volume 4763 of *LNCS*, pages 179–194. Springer, 2007.
- [80] Noah J. Goodall. Can you program ethics into a self-driving car? *IEEE Spectrum*, 53(6):28–58, 2016.
- [81] David Goodman and Raymond Keene. Man versus machine: Kasparov versus deep blue. *ICGA Journal*, 20(3):186–187, 1997.
- [82] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *FOSE*, pages 167–181. ACM, 2014.
- [83] Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. Analysis of timed and long-run objectives for markov automata. *LMCS*, 10(3), 2014.
- [84] Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. Modelling and analysis of markov reward automata. In *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis, ATVA*, pages 168–184, 2014.

- 
- [85] Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. Modelling and analysis of markov reward automata. In *Proc. of the 12th Int. Symposium on Automated Technology for Verification and Analysis, ATVA*, pages 168–184, 2014.
- [86] MATLAB Users Guide. The Mathworks. *Inc., Natick, MA*, 5:333, 1998.
- [87] Havva Gulay Gurbuz and Bedir Tekinerdogan. Model-based testing for software safety: a systematic mapping study. *Software Quality Journal*, Sep 2017.
- [88] Ernst Moritz Hahn, Arnd Hartmanns, and Holger Hermanns. Reachability and reward checking for stochastic timed automata. In *AVoCS*, volume 70 of *Electronic Communications of the EASST*, 2014.
- [89] Peter G. Harrison and Ben Strulo. SPADES – a process algebra for discrete event simulation. *J. Log. Comput.*, 10(1):3–42, 2000.
- [90] Arnd Hartmanns. *On the analysis of stochastic timed systems*. PhD thesis, Saarland University, 2015.
- [91] Arnd Hartmanns, Holger Hermanns, and Jan Krcál. Schedulers are no prophets. In *Semantics, Logics, and Calculi*, volume 9560 of *LNCS*, pages 214–235. Springer, 2016.
- [92] Arnd Hartmanns, Sean Sedwards, and Pedro R. D’Argenio. Efficient simulation-based verification of probabilistic timed automata. In *Winter Simulation Conference, WSC*, pages 1419–1430, 2017.
- [93] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.
- [94] Holger Hermanns, Julia Krämer, Jan Krcál, and Mariëlle Stoelinga. The value of attack-defence diagrams. In *POST*, volume 9635 of *LNCS*, pages 163–185. Springer, 2016.
- [95] Holger Hermanns, Jan Krcál, and Jan Kretínský. Probabilistic bisimulation: Naturally on distributions. In *Proceedings of the 25th International Conference on Concurrency Theory, CONCUR*, pages 249–265, 2014.
- [96] Robert M. Hierons and Mercedes G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software*, pages 1804–1818, 2009.
- [97] Robert M. Hierons, Mercedes G. Merayo, and Manuel Núñez. Testing from a stochastic timed system with a fault model. *J. Log. Algebr. Program.*, 78(2):98–115, 2009.
- [98] Robert M. Hierons and Manuel Núñez. Testing probabilistic distributed systems. volume 6117 of *LNCS*, pages 63–77. Springer, 2010.

- [99] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*. John Wiley & Sons, 2013.
- [100] Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing for nondeterministic systems. *Formal Asp. Comput.*, 29(2):335–364, 2017.
- [101] Felix Hübner, Wen-ling Huang, and Jan Peleska. Experimental evaluation of a novel equivalence class partition testing strategy. *Software & Systems Modeling*, 2017.
- [102] Antti Huima. Implementing Conformiq Qtronic. In *Testing of Software and Communicating Systems*, pages 1–12. Springer Berlin Heidelberg, 2007.
- [103] Iksoon Hwang and Ana R. Cavalli. Testing a probabilistic FSM using interval estimation. *Computer Networks*, pages 1108–1125, 2010.
- [104] Shantanu Ingle and Madhuri Phute. Tesla autopilot: semi autonomous driving, an uptick for future autonomy. *International Research Journal of Engineering and Technology*, 3(9), 2016.
- [105] Claude Jard and Thierry Jéron. TGV: theory, principles and algorithms. *STTT*, 7(4):297–315, 2005.
- [106] Bertrand Jeannot, Pedro R. D’Argenio, and Kim G. Larsen. Rapture: A tool for verifying markov decision processes. *Tools Day*, 2:149, 2002.
- [107] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678, 2011.
- [108] Macarthur Job. Air disaster, vol. 2. *Canberra: Aerospace Publications*, 1996.
- [109] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: high-performance language-independent model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 692–707. Springer, 2015.
- [110] Kristian Karl. Graphwalker. [www.graphwalker.org](http://www.graphwalker.org). Accessed: 2018-09-18.
- [111] Joost-Pieter Katoen. The probabilistic model checking landscape. In *LICS*, pages 31–45. ACM, 2016.
- [112] John C. Kelly, Joseph S. Sherif, and Jonathan Hops. An analysis of defect densities found during software inspections. *Journal of Systems and Software*, 17(2):111 – 117, 1992.

- 
- [113] Henning Kerstan and Barbara König. Coalgebraic trace semantics for continuous probabilistic transition systems. *Logical Methods in Computer Science*, 9(4), 2013.
- [114] Donald E. Knuth and Andrew C. Yao. The complexity of nonuniform random number generation. *Algorithms and complexity: new directions and recent results*, pages 357–428, 1976.
- [115] Daniel Krawczyk. *Reasoning: The Neuroscience of how We Think*. Academic Press, 2017.
- [116] Moez Krichen and Stavros Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.
- [117] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. MANET simulation studies: the incredibles. *Mobile Computing and Communications Review*, 9(4):50–61, 2005.
- [118] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification, CAV*, pages 585–591, 2011.
- [119] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *CONCUR*, volume 1877 of *LNCS*, pages 123–137. Springer, 2000.
- [120] Kim G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using UPPAAL. volume 3395 of *LNCS*, pages 79–94. Springer, 2005.
- [121] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. UPPAAL TRON user manual. *CISS, BRICS, Aalborg University, Aalborg, Denmark*, 2009.
- [122] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28, 1991.
- [123] Harold J. Larson. *Introduction to probability theory and statistical inference*, volume 12. Wiley New York, 1969.
- [124] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
- [125] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Estimating rewards & rare events in nondeterministic systems. In *AVoCS*, volume 72 of *Electronic Communications of the EASST*, 2015.

- [126] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Scalable verification of Markov decision processes. In *SEFM*, volume 8938 of *LNCS*, pages 350–362. Springer, 2015.
- [127] Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE computer*, 26(7):18–41, 1993.
- [128] Markus Lohrey, Pedro R. D’Argenio, and Holger Hermanns. Axiomatising divergence. In *International Colloquium on Automata, Languages, and Programming*, pages 585–596. Springer, 2002.
- [129] Robyn R. Lutz. Analyzing software requirements errors in safety-critical, embedded systems. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 126–133. IEEE, 1993.
- [130] Robyn R. Lutz. Targeting safety-related errors during software requirements analysis. *Journal of Systems and Software*, 34(3):223–230, 1996.
- [131] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. 1988.
- [132] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1994.
- [133] Mercedes G. Merayo, Iksoon Hwang, Manuel Núñez, and Ana Cavalli. A statistical approach to test stochastic and probabilistic systems. In *Formal Methods and Software Engineering*, volume 5885 of *LNCS*, pages 186–205. Springer, 2009.
- [134] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1982.
- [135] Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.
- [136] Wojciech Mostowski, Erik Poll, Julien Schmaltz, Jan Tretmans, and Ronny Wichers Schreur. Model-based testing of electronic passports. In *Proceedings of the 14th International Workshop on Formal Methods for Industrial Critical Systems, FMICS*, pages 207–209, 2009.
- [137] Jiawang Nie, James Demmel, and Ming Gu. Global minimization of rational functions and the nearest GCDs. *Journal of Global Optimization*, 40(4):697–718, 2008.
- [138] Manuel Núñez and Ismael Rodríguez. Towards testing stochastic timed systems. In *FORTE*, pages 335–350, 2003.
- [139] Patrick O’Connor and Andre Kleyner. *Practical reliability engineering*. John Wiley & Sons, 2012.

- 
- [140] National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. *Strategic Planning*, 2002.
- [141] Chee-Mun Ong. *Dynamic simulation of electric machinery: using MATLAB/SIMULINK*, volume 5. Prentice hall PTR Upper Saddle River, NJ, 1998.
- [142] Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. *CoRR*, arXiv:1403.0504, 2014.
- [143] Sofia Costa Paiva, Adenilso Simao, Mahsa Varshosaz, and Mohammad Reza Mousavi. Complete ioco test cases: a case study. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*, pages 38–44. ACM, 2016.
- [144] Lawrence C. Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.
- [145] Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.
- [146] Alexandre Petrenko, Arnaud Dury, S. Ramesh, and Swarup Mohalik. A method and tool for test optimization for automotive controllers. In *IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops ICSTW*, pages 198–207. IEEE, 2013.
- [147] Alexandre Petrenko, Nina Yevtushenko, and Gregor von Bochmann. Fault models for testing in context. In *International Conference on Formal Description Techniques IX*, pages 163–178, 1996.
- [148] Avi Pfeffer. Practical probabilistic programming. In *Inductive Logic Programming*, volume 6489 of *LNCS*, pages 2–3. Springer Berlin Heidelberg, 2011.
- [149] Alexander Pretschner. Model-based testing. In *27th International Conference on Software Engineering, ICSE*, pages 722–723, 2005.
- [150] Stacy J. Prowell. JUMBL: A tool for model-based statistical testing. In *Proceedings of the 36th Annual International Conference on System Sciences*, pages 9–pp. IEEE, 2003.
- [151] Daniël Reijnsbergen, Pieter-Tjerk de Boer, Werner R. W. Scheinhardt, and Boudewijn R. Haverkort. On hypothesis testing for statistical model checking. *STTT*, 17(4):377–395, 2015.
- [152] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015.



- [153] Enno Ruijters and Mariëlle Stoelinga. Better railway engineering through statistical model checking. In *ISoLA*, volume 9952 of *LNCS*, pages 151–165. Springer, 2016.
- [154] Martin Russell and Roger Moore. Explicit modelling of state occupancy in hidden Markov models for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, volume 10, pages 5–8, 1985.
- [155] Jonathan Schaeffer and Aske Plaat. Kasparov versus Deep Blue: The rematch. *ICGA Journal*, 20(2):95–101, 1997.
- [156] Mark J. Schervish. *Theory of statistics*. Springer Science & Business Media, 2012.
- [157] Ina Schieferdecker. Model-based testing. *IEEE Software*, 29(1):14–18, 2012.
- [158] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Cambridge, MA, USA, 1995.
- [159] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *Computer Aided Verification*, pages 202–215. Springer Berlin Heidelberg, 2004.
- [160] Muhammad Shafique and Yvan Labiche. A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, 17(1):59–76, Feb 2015.
- [161] Bluetooth SIG. Bluetooth specification, version 1.2. [www.bluetooth.com](http://www.bluetooth.com), 2003. Accessed: 2018-09-18.
- [162] Lei Song, Lijun Zhang, and Jens Chr. Godskesen. Late weak bisimulation for Markov automata. *CoRR*, abs/1202.4116, 2012.
- [163] Graham Steel. Formal analysis of PIN block attacks. *Theoretical Computer Science*, 367(1-2):257–270, 2006.
- [164] Mariëlle Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-Time and Parametric Systems*. PhD thesis, University of Nijmegen, The Netherlands, 2002.
- [165] Mariëlle Stoelinga and Frits Vaandrager. Root contention in IEEE 1394. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 53–74. Springer, 1999.
- [166] Willem Gerrit Johan Stokkink, Mark Timmer, and Mariëlle Stoelinga. Divergent quiescent transition systems. In *Proceedings of the 7th International Conference on Tests and Proofs, TAP*, pages 214–231, 2013.

- 
- [167] Ben Strulo. *Process algebra for discrete event simulation*. PhD thesis, Imperial College of Science, Technology and Medicine. University of London, October 1993.
- [168] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.
- [169] Mark Timmer, Ed Brinksma, and Mariëlle Stoelinga. Model-based testing. In *Software and Systems Safety - Specification and Verification*, pages 1–32. 2011.
- [170] Mark Timmer, Joost-Pieter Katoen, Jaco van de Pol, and Mariëlle Stoelinga. Efficient modelling and generation of Markov automata. In *23rd International Conference on Concurrency Theory, CONCUR*, volume 7454 of *LNCS*, pages 364–379. Springer, 2012.
- [171] Mark Timmer, Jaco van de Pol, and Mariëlle Stoelinga. Confluence reduction for Markov automata. In *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2013.
- [172] Larry E. Toothaker. *Multiple comparison procedures*. Number 89. Sage, 1993.
- [173] Jan Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29(1):49–79, 1996.
- [174] Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, pages 1–38, 2008.
- [175] Jan Tretmans. *On the Existence of Practical Testers*, pages 87–106. Springer International Publishing, 2017.
- [176] Jan Tretmans and Ed Brinksma. Côte de resyste – automated model based testing. 2002.
- [177] Jan Tretmans, Klaas Wijbrans, and Michel R. V. Chaudron. Software engineering with formal methods: The development of a storm surge barrier control system revisiting seven myths of formal methods. *Formal Methods in System Design*, 19(2):195–215, 2001.
- [178] Hans Triebel. *Higher Analysis*, volume 93. Barth, 1992.
- [179] Hasan Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311 – 325, 1992.
- [180] Mark Utting. How to design extended finite state machine test models in java. *Model-Based Testing for Embedded Systems*, pages 147–169, 2012.

- [181] Mark Utting, Bruno Legeard, Fabrice Bouquet, Elizabeta Fourneter, Fabien Peureux, and Alexandre Vernotte. Recent advances in model-based testing. In *Advances in Computers*, volume 101, pages 53–120. Elsevier, 2016.
- [182] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.
- [183] Petra van den Bos, Ramon Janssen, and Joshua Moerman. n-complete test suites for IOCO. In *Proceedings of the 29th International Conference Testing on Software and Systems, IFIP*, pages 91–107, 2017.
- [184] Rob J. van Glabbeek. The linear time-branching time spectrum. In *Proceedings of Theories of Concurrency, CONCUR*, pages 278–297, 1990.
- [185] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative, and stratified models of probabilistic processes. volume 121, pages 59–80. Elsevier, 1995.
- [186] Axel van Lamsweerde. Formal specification: a roadmap. In *Proceedings of the 22nd International Conference on the Future of Software Engineering, ICSE*, pages 147–159, 2000.
- [187] Michiel van Osch. Hybrid input-output conformance and test generation. In *Revised Selected Papers from the First Combined International Workshops on Formal Approaches to Software Testing and Runtime Verification, FATES and RV*, pages 70–84, 2006.
- [188] 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, An Outcome of the FORTEST Network, Revised Selected Papers*, pages 39–76, 2008.
- [189] Abraham Wald. Sequential tests of statistical hypotheses. *Ann. Math. Statist.*, 16(2):117–186, 06 1945.
- [190] Gwendolyn H. Walton, Jesse H. Poore, and Carmen J. Trammell. Statistical testing of software based on a usage model. *Softw., Pract. Exper.*, 25(1):97–108, 1995.
- [191] Jingbo Wang, Wai Wan Tsang, and George Marsaglia. Evaluating kolmogorov’s distribution. *Journal of Statistical Software*, 8(18), 2003.
- [192] Gou Watanabe and Norihiro Ishikawa. Da Vinci surgical system. *Kyobu geka. The Japanese journal of thoracic surgery*, 67(8):686–689, 2014.
- [193] Bruce Weber. Swift and slashing, computer topples Kasparov. *New York Times*, 12, 1997.

- 
- [194] James A. Whittaker. What is software testing? And why is it so hard? *IEEE software*, 17(1):70–79, 2000.
- [195] Verena Wolf, Christel Baier, and Mila E. Majster-Cederbaum. Trace machines for observing continuous-time Markov chains. *ENTCS*, 153(2):259–277, 2006.
- [196] Verena Wolf, Christel Baier, and Mila E. Majster-Cederbaum. Trace semantics for stochastic systems with nondeterminism. *Electr. Notes Theor. Comput. Sci.*, 164(3):187–204, 2006.
- [197] Nicolás Wolovick. *Continuous probability and nondeterminism in labeled transition systems*. PhD thesis, Universidad Nacional de Córdoba, Córdoba, Argentina, 2012.
- [198] Nicolás Wolovick and Sven Johr. A characterization of meaningful schedulers for continuous-time Markov decision processes. In *FORMATS*, volume 4202 of *LNCS*, pages 352–367. Springer, 2006.

---

## Samenvatting

---

Kansen spelen een belangrijke rol in veel computerapplicaties. Een groot scala aan algoritmen, protocollen en berekenmethoden gebruiken randomisatie om hun doelen te bereiken. Een cruciale vraag is dan of zulke probabilistische systemen werken zoals bedoeld. Om dit te onderzoeken worden zulke systemen getest door een groot aantal goed ontworpen tests, die het geobserveerde gedrag vergelijken met een specificatie. Doorgaans worden deze tests gemaakt met de hand, waarbij menselijke fouten niet onmogelijk zijn. Een andere aanpak is het automatisch genereren van deze tests. Model-gebaseerd testen is een innovatieve testmethode vanuit de formele methoden, die de taak van tests maken poogt te automatiseren. Het heeft tractie gevonden in zowel academia als de industrie, doordat het sneller en rigoreuzer kan testen. Echter, klassiek model-gebaseerde testmethoden zijn niet afdoende wanneer de systemen van stochastische aard zijn.

Dit proefschrift introduceert een rigoreus model-gebaseerd testraamwerk, dat het mogelijk maakt dergelijke systemen automatisch te testen. De gepresenteerde methoden kunnen functionele correctheid beoordelen, discrete probabilistische keuzes, en harde en zachte tijdsbeperkingen. Allereerst wordt het vakgebied van model-gebaseerd testen beschreven en gerelateerd werk besproken. Het raamwerk is daarna duidelijk en stapsgewijs opgesteld. Vervolgens wordt een model-gebaseerd testraamwerk geïntantieerd om het doel aan te geven van de theoretische competenten, zoals een conformiteitsrelatie of testgevallen. Dit raamwerk wordt dan conservatief uitgebreid door discrete probabilistische keuzes aan de specificatietaal toe te voegen. Ten slotte wordt dit probalistische raamwerk uitgebreid met harde en zachte tijdsbeperkingen. Uitspraken over klassiek functionele correctheid zijn dus uitgebreid met statistische goodness of fit methoden. Het bewijs dat dit raamwerk correct is worden gegeven alvorens de mogelijkheden ervan aan te stippen door middel van kleine casussen bekend van de literatuur.

Het raamwerk verenigt niet-deterministische en probabilistische keuzes op een volwaardige manier middels het gebruik van schedulers. Dit maakt schedulers op zich zelf al interessant voor een eigen studie. Dit wordt gedaan in het tweede deel van dit proefschrift: We introduceren een gelijkwaardigheid-srelatie gebaseerd op schedulers voor Markov-automaten en vergelijken haar onderscheidende expressiviteit wat betreft spoordistributies en bisimulatie-relaties. Als laatst onderzoeken we de kracht van verschillende schedulerklassen voor stochastische automaten. We vergelijken de bereikbaarheidskansen van verschillende schedulers door de informatie die voor hen beschikbaar is te veranderen om een hiërarchie van schedulerklassen te vestigen.

## Titles in the IPA Dissertation Series since 2015

**G. Alpár.** *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World.* Faculty of Science, Mathematics and Computer Science, RU. 2015-01

**A.J. van der Ploeg.** *Efficient Abstractions for Visualization and Interaction.* Faculty of Science, UvA. 2015-02

**R.J.M. Theunissen.** *Supervisory Control in Health Care Systems.* Faculty of Mechanical Engineering, TU/e. 2015-03

**T.V. Bui.** *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness.* Faculty of Mathematics and Computer Science, TU/e. 2015-04

**A. Guzzi.** *Supporting Developers' Teamwork from within the IDE.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

**T. Espinha.** *Web Service Growing Pains: Understanding Services and Their Clients.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

**S. Dietzel.** *Resilient In-network Aggregation for Vehicular Networks.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

**E. Costante.** *Privacy throughout the Data Cycle.* Faculty of Mathematics and Computer Science, TU/e. 2015-08

**S. Cranen.** *Getting the point — Obtaining and understanding fix-points in model checking.* Faculty of

Mathematics and Computer Science, TU/e. 2015-09

**R. Verdult.** *The (in)security of proprietary cryptography.* Faculty of Science, Mathematics and Computer Science, RU. 2015-10

**J.E.J. de Ruiter.** *Lessons learned in the analysis of the EMV and TLS security protocols.* Faculty of Science, Mathematics and Computer Science, RU. 2015-11

**Y. Dajsuren.** *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems.* Faculty of Mathematics and Computer Science, TU/e. 2015-12

**J. Bransen.** *On the Incremental Evaluation of Higher-Order Attribute Grammars.* Faculty of Science, UU. 2015-13

**S. Picek.** *Applications of Evolutionary Computation to Cryptology.* Faculty of Science, Mathematics and Computer Science, RU. 2015-14

**C. Chen.** *Automated Fault Localization for Service-Oriented Software Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

**S. te Brinke.** *Developing Energy-Aware Software.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

**R.W.J. Kersten.** *Software Analysis Methods for Resource-Sensitive Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2015-17

**J.C. Rot.** *Enhanced coinduction.* Faculty of Mathematics and Natural Sciences, UL. 2015-18

- M. Stolikj.** *Building Blocks for the Internet of Things.* Faculty of Mathematics and Computer Science, TU/e. 2015-19
- D. Gebler.** *Robust SOS Specifications of Probabilistic Processes.* Faculty of Sciences, Department of Computer Science, VUA. 2015-20
- M. Zaharieva-Stojanovski.** *Closer to Reliable Software: Verifying functional behaviour of concurrent programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21
- R.J. Krebbers.** *The C standard formalized in Coq.* Faculty of Science, Mathematics and Computer Science, RU. 2015-22
- R. van Vliet.** *DNA Expressions – A Formal Notation for DNA.* Faculty of Mathematics and Natural Sciences, UL. 2015-23
- S.-S.T.Q. Jongmans.** *Automata-Theoretic Protocol Programming.* Faculty of Mathematics and Natural Sciences, UL. 2016-01
- S.J.C. Joosten.** *Verification of Interconnects.* Faculty of Mathematics and Computer Science, TU/e. 2016-02
- M.W. Gazda.** *Fixpoint Logic, Games, and Relations of Consequence.* Faculty of Mathematics and Computer Science, TU/e. 2016-03
- S. Keshishzadeh.** *Formal Analysis and Verification of Embedded Systems for Healthcare.* Faculty of Mathematics and Computer Science, TU/e. 2016-04
- P.M. Heck.** *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05
- Y. Luo.** *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance.* Faculty of Mathematics and Computer Science, TU/e. 2016-06
- B. Ege.** *Physical Security Analysis of Embedded Devices.* Faculty of Science, Mathematics and Computer Science, RU. 2016-07
- A.I. van Goethem.** *Algorithms for Curved Schematization.* Faculty of Mathematics and Computer Science, TU/e. 2016-08
- T. van Dijk.** *Sylvan: Multi-core Decision Diagrams.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09
- I. David.** *Run-time resource management for component-based systems.* Faculty of Mathematics and Computer Science, TU/e. 2016-10
- A.C. van Hulst.** *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs.* Faculty of Mechanical Engineering, TU/e. 2016-11
- A. Zawedde.** *Modeling the Dynamics of Requirements Process Improvement.* Faculty of Mathematics and Computer Science, TU/e. 2016-12
- F.M.J. van den Broek.** *Mobile Communication Security.* Faculty of Science, Mathematics and Computer Science, RU. 2016-13
- J.N. van Rijn.** *Massively Collaborative Machine Learning.* Faculty of Mathematics and Natural Sciences, UL. 2016-14
- M.J. Steindorfer.** *Efficient Immutable Collections.* Faculty of Science, UvA. 2017-01

- W. Ahmad.** *Green Computing: Efficient Energy Management of Multi-processor Streaming Applications via Model Checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02
- D. Guck.** *Reliable Systems – Fault tree analysis via Markov reward automata.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03
- H.L. Salunkhe.** *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors.* Faculty of Mathematics and Computer Science, TU/e. 2017-04
- A. Krasnova.** *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT).* Faculty of Science, Mathematics and Computer Science, RU. 2017-05
- A.D. Mehrabi.** *Data Structures for Analyzing Geometric Data.* Faculty of Mathematics and Computer Science, TU/e. 2017-06
- D. Landman.** *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities.* Faculty of Science, UvA. 2017-07
- W. Lueks.** *Security and Privacy via Cryptography – Having your cake and eating it too.* Faculty of Science, Mathematics and Computer Science, RU. 2017-08
- A.M. Şutii.** *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod.* Faculty of Mathematics and Computer Science, TU/e. 2017-09
- U. Tikhonova.** *Engineering the Dynamic Semantics of Domain Specific Languages.* Faculty of Mathematics and Computer Science, TU/e. 2017-10
- Q.W. Bouts.** *Geographic Graph Construction and Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2017-11
- A. Amighi.** *Specification and Verification of Synchronisation Classes in Java: A Practical Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01
- S. Darabi.** *Verification of Program Parallelization.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02
- J.R. Salamanca Tellez.** *Coequations and Eilenberg-type Correspondences.* Faculty of Science, Mathematics and Computer Science, RU. 2018-03
- P. Fiterău-Broştean.** *Active Model Learning for the Analysis of Network Protocols.* Faculty of Science, Mathematics and Computer Science, RU. 2018-04
- D. Zhang.** *From Concurrent State Machines to Reliable Multi-threaded Java Code.* Faculty of Mathematics and Computer Science, TU/e. 2018-05
- H. Basold.** *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic.* Faculty of Science, Mathematics and Computer Science, RU. 2018-06
- A. Lele.** *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems.* Faculty of Mathematics and Computer Science, TU/e. 2018-07
- N. Bezirgiannis.** *Abstract Behavioral Specification: unifying modeling and programming.* Faculty of



Mathematics and Natural Sciences,  
UL. 2018-08

**M.P. Konzack.** *Trajectory Analysis: Bridging Algorithms and Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2018-09

**E.J.J. Ruijters.** *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

**F. Yang.** *A Theory of Executability: with a Focus on the Expressivity of Process Calculi.* Faculty of Mathematics and Computer Science, TU/e. 2018-11

**L. Swartjes.** *Model-based design of baggage handling systems.* Faculty of Mechanical Engineering, TU/e. 2018-12

**T.A.E. Ophelders.** *Continuous Similarity Measures for Curves and Surfaces.* Faculty of Mathematics and Computer Science, TU/e. 2018-13

**M. Talebi.** *Scalable Performance Analysis of Wireless Sensor Network.* Faculty of Mathematics and Computer Science, TU/e. 2018-14

**R. Kumar.** *Truth or Dare: Quantitative security analysis using attack trees.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

**M.M. Beller.** *An Empirical Evaluation of Feedback-Driven Software Development.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

**M. Mehr.** *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems.* Faculty of Mathematics and Computer Science, TU/e. 2018-17

**M. Alizadeh.** *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations.* Faculty of Mathematics and Computer Science, TU/e. 2018-18

**P.A. Inostroza Valdera.** *Structuring Languages as Object-Oriented Libraries.* Faculty of Science, UvA. 2018-19

**M. Gerhold.** *Choice and Chance - Model-Based Testing of Stochastic Behaviour.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20