# Trace-based Process Algebras for Real-Time Probabilistic Systems

Stefano Cattani

A thesis submitted to
The University of Birmingham
for the degree of Doctor of Philosophy

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
United Kingdom

June 2005

# Abstract

We study the problem of defining trace-based models for systems with real-time and stochastic behaviour. Trace semantics is a simple and intuitive model that describes the behaviour of a system in terms of the possible actions it can perform, and it can be used for the specification of safety properties. We present two process algebras, one featuring real-time and another stochastic behaviour, and investigate their trace semantics.

In the case of real-time systems, we propose a real-time extension to the process algebra CSP. Inspired by timed automata, a successful formalism for the specification and verification of real-time systems, we handle real time by means of clocks, i.e. real-valued variables that increase at the same rate as time. We give a discrete trace semantics to our language and define the resulting refinement and equivalence relations. One advantage of our proposal is that it is possible to automatically verify refinement relations between processes. We demonstrate how this can be achieved and also describe the syntactic restrictions needed and the limitations of this approach.

We take a different approach in the case of stochastic systems: in this case we focus on the operational model, and investigate ways to define a trace semantics. We study the interaction between non-deterministic and probabilistic behaviour in systems with continuous state spaces, arbitrary probability distributions and uncountable branching. We identify the class of schedulers that ensures measurability properties on executions, and show that such measurability properties are preserved by parallel composition. Under these restrictions, we define trace semantics and show that a more general notion of bisimulation is needed when dealing with continuous state spaces in order to preserve linear-time semantics. Finally, we show how this model can be used to give semantics to a stochastic process algebra.

# Acknowledgements

In writing this thesis, I have been greatly assisted in terms of encouragement, time, and guidance by a number of individuals. Firstly, my supervisor, Marta Kwiatkowska, for her timely assistance during our weekly meetings to discuss my progress, and for her constant evaluation of the various drafts of my work. Secondly, I would like to thank Roberto Segala, without whom I would not be in the position of writing this thesis and who has provided me with lots of helpful ideas and technical support. Then, I would like to acknowledge Mark Dermot Ryan for his friendship and support throughout my PhD. I would also like to thank Gethin Norman for his technical support, Prakash Panangaden for our helpful discussions, and Achim Jung for his advice.

Finally, an equally important contribution has come from my family and from my friends here in Birmingham: Richard Coltman, John Woodward, Volker Sorge, Tebogo Seipone, Dzena and Jon Rowe, Hannah Harris, Gan Ohama, Amin Farjudian.

This research was funded by the EPSRC project "Verification of Quality of Service Properties in Timed Systems" (GR/N22960).

# Table of Contents


**1 Introduction** **1**

1.1 Aims and Contributions . . . 3

1.1.1 Real-time Systems . . . 4

1.1.2 Stochastic Systems . . . 4

1.2 Structure of the Thesis . . . 5

**2 Preliminaries** **7**

2.1 Labelled Transition Systems . . . 7

2.1.1 Equivalences for Labelled Transition Systems . . . 8

2.1.2 Discussion . . . 10

2.2 Process Algebras and CSP . . . 10

2.2.1 CSP . . . 11

2.3 Probability and Measure Theory Background . . . 12

2.3.1 Basic Definitions . . . 12

2.3.2 Regular Conditional Probabilities . . . 14

2.3.3 A $\sigma$-algebra on Probability Measures . . . 15

**I Real-Time Systems** **17**

**3 Real-Time Systems** **19**

3.1 Timed Automata . . . 19

3.1.1 Fundamental Results about Timed Automata . . . 20

3.1.2 Extensions to Timed Automata . . . 22

3.1.3 Model Checking Timed Automata . . . 23

3.2 Timed Process Algebras . . . 23

3.3 Our Aims and Contribution . . . 24

**4 A Real-Time Process Algebra** **27**

4.1 Introduction . . . 27

4.2 Preliminaries: Timed Automata . . . 27

4.2.1 Timed Automata . . . 28

4.2.2 Semantics of Timed Automata . . . 29

4.2.3 Region Automata . . . 30
4.2.4 Timed Bisimulation . . . 32
4.3 Clocked CSP . . . 33
4.3.1 Aims . . . 33
4.3.2 The Language . . . 33
4.3.3 Design Choices . . . 34
4.3.4 Internal and External Clocks . . . 35
4.4 Operational Semantics . . . 36
4.4.1 Clocked CSP and Timed Automata . . . 38
4.4.2 Equivalence Relations via the Operational Model . . . 39
4.4.3 Normal Form . . . 39

**5 Denotational Semantics 41**
5.1 Introduction . . . 41
5.2 Trace Semantics . . . 42
5.2.1 Region Traces . . . 42
5.2.2 Rules for Region Traces . . . 43
5.3 Failure Semantics . . . 48
5.3.1 Region Failures . . . 49
5.3.2 Rules for Region Failures . . . 50
5.4 Analysing the Semantic Models . . . 52
5.4.1 Comparison with the Operational Semantics . . . 52
5.5 Model Checking Clocked CSP . . . 57
5.5.1 The Main Idea . . . 58
5.5.2 A Model Checking Example . . . 58
5.5.3 Considerations . . . 62
5.6 Discussion . . . 62

**6 Conclusions 63**
6.1 Future Work . . . 63

**II Stochastic Systems 65**

**7 Stochastic Systems 67**
7.1 Discrete Probabilities . . . 67
7.2 Continuous Systems . . . 70
7.3 Our Aims and Contribution . . . 72

**8 Stochastic Transition Systems 75**
8.1 Introduction . . . 75
8.2 The Model . . . 76
8.2.1 Stochastic Transition Systems . . . 76

8.2.2 Combined Transitions . . . 77
8.2.3 Executions and Traces . . . 78
8.2.4 A $\sigma$-algebra on Executions . . . 78
8.2.5 Schedulers . . . 80
8.3 Probabilistic Executions . . . 81
8.3.1 Measurable Schedulers and Probabilistic Executions . . . 81
8.3.2 A Probability Measure on Executions . . . 83
8.3.3 Weak Transitions . . . 84
8.4 Parallel Composition and Measurability . . . 86
8.4.1 Parallel Composition . . . 86
8.4.2 Compositionality Results . . . 86
8.5 Discussion . . . 90

**9 Equivalences for Stochastic Transition Systems 93**
9.1 Introduction . . . 93
9.2 Trace Semantics . . . 94
9.3 Local Bisimulation . . . 95
9.3.1 Local Bisimulation and Trace Semantics . . . 96
9.4 Bisimulation . . . 97
9.4.1 Strong Bisimulation . . . 97
9.4.2 Weak Bisimulation . . . 98
9.4.3 Discussion: a Logical Characterisation of Bisimulation? . . . 101
9.5 Trace Equivalence and Bisimulation . . . 102
9.5.1 Fringes . . . 103
9.5.2 Execution Correspondence Structure . . . 103
9.5.3 The Main Result . . . 104
9.5.4 Summary . . . 115
9.6 Discussion . . . 116

**10 A Stochastic Process Algebra 117**
10.1 Introduction . . . 117
10.2 The Language . . . 118
10.2.1 Syntax . . . 118
10.2.2 What We Model . . . 119
10.3 Operational Semantics . . . 120
10.3.1 SOS Rules . . . 121
10.4 Equivalences for SCSP . . . 125
10.4.1 Trace Semantics . . . 125
10.4.2 Bisimulation Equivalences . . . 128
10.5 Discussion and Summary . . . 129

**11 Conclusions** **131**
11.1 Future Work . . . . . 132
11.1.1 Stochastic Transition Systems . . . . . 132
11.1.2 Stochastic Process Algebra . . . . . 132

**Bibliography** **133**

**Index** **147**

# List of Figures


2.1 Two systems that are trace equivalent but not failure equivalent. . . . . 9
2.2 Two systems that are failure equivalent but not bisimilar. . . . . 10

4.1 A timed automaton. . . . . 29
4.2 A region graph. . . . . 31
4.3 A timed automaton (top) $\mathcal{A}$ and the corresponding region automaton $R(\mathcal{A})$ (bottom). . . . . 32
4.4 Explanation of the $\mathsf{free}(\bullet)$ operator. . . . . 38
4.5 Timed automaton induced by $(x \leq 2) \triangleright \{|x|\}(a, x > 1) \rightarrow P$. Clock $x$ is bound by the initial reset. . . . . 39

5.1 The region automata corresponding to some operators. . . . . 46
5.2 Two region automata, $R_1$ with internal clock $x_1$ and $R_2$ with internal clock $x_2$, and their parallel composition $R_1 \underset{A}{||} R_2$. . . . . 55
5.3 The timed automaton corresponding to the bounded invariance property. . 59
5.4 The region automaton corresponding to the bounded invariance property. . 59
5.5 A possible implementation process $P$ and its parallel composition with the test automaton $T$ corresponding to the bounded invariance property. . . . . 60
5.6 The region automaton corresponding to the process $P \underset{\{a\}}{||} T$ of Figure **??**. . 61

8.1 A simple stochastic transition system. . . . . 82
8.2 A simple weak transition. . . . . 85

9.1 Trace semantics is not preserved by parallel composition. . . . . 94
9.2 $S_1$ and $S_2$ are not trace equivalent but locally bisimilar. . . . . 96
9.3 Two systems that cannot be distinguished without negation. . . . . 102
9.4 Two systems that cannot be distinguished without infinite conjunction. . . 102
9.5 An execution correspondence structure. . . . . 104
9.6 The relationship between the equivalences for stochastic transition systems 116

10.1 A graphical representation of a SCSP term. . . . . 120
10.2 Relationship between the different equivalence relations for SCSP. . . . . 129

# List of Tables


2.1 Operational semantics for CSP. . . . . 12

4.1 Transition relation of the lts associated to a timed automaton. . . . . 29
4.2 Syntax of Clocked CSP . . . . . 33
4.3 Rules for internal clocks. . . . . 35
4.4 Rules for invariants . . . . . 36
4.5 Rules for clock resets . . . . . 37
4.6 Rules for transitions . . . . . 37
4.7 Normal form for CCSP processes. . . . . 40
4.8 Rewrite rules to obtain CCSP terms that are equivalent up to the operational semantics, i.e., they induce the same timed automaton. . . . . 40

10.1 Syntax of Stochastic CSP . . . . . 118
10.2 Operational semantics rules for SCSP for probabilistic terms. . . . . 121
10.3 Operational semantics rules for SCSP's standard operators. . . . . 122
10.4 Operational semantics rules for the parallel operator. . . . . 123

# Chapter 1

# Introduction

Formal analysis of concurrent systems has been a major topic of research for more than twenty years. When analysing distributed and/or concurrent systems, their correctness is much harder to verify since the processes involved can interact in many different ways. All this creates the need for research into formal methods for the specification and analysis of concurrent systems.

Over the years, the requirement for verification has been constantly reinforced by the spread of computers to virtually every part of our life, thus making the correctness of these devices a fundamental topic. At the same time, new issues are introduced as different types of behaviour are being analysed. In particular, communication systems and ubiquitous computing have prompted more research in the areas of real-time and probabilistic systems, which are the subject of this thesis. The need for such systems (and their analysis) can be easily justified. Consider, for example, the transmission of video or audio streams across a network: several factors influence the quality of the stream, depending on the delivery of packets occurring in a timely fashion. So, in order to verify the functionality of the protocol in a certain network, one might want to verify properties like *each packet arrives to destination within n seconds with a probability of at least* 90%. Applications also include safety-critical systems, like air traffic control systems, for which the notion of time and potential failures are critical. Therefore, it is necessary to define a framework for the analysis of real-time systems that are able to consider performance related issues as well as their reliability.

The first challenge is to define formal models that are capable of describing the systems we wish to model. These are necessary in order to give a system an unambiguous semantics, on the basis of which we can define equivalence relations and verification techniques. Candidate models must also be able to represent the behaviour of systems, in particular the non-deterministic behaviour that arises from concurrency.

Graph based representations such as labelled transition systems [Plo81] can be used for the modelling of concurrent systems: the nodes of the graph represent the possible states the system can be in, and the transitions represent its communications. Non-determinism is modelled by enabling several transitions from each state. While such models are intuitive and often easier to study, they are unsuitable for describing large systems, because the size of the graphs would make the process too error prone.

Process algebras are one successful approach to the compositional modelling of systems: the first proposals date back more than twenty years, to the introduction of CCS [Mil89] and CSP [Hoa85]. Process algebras offer the opportunity to describe systems at a *high level*, and with a high degree of *compositionality*: systems are defined in terms of their smaller components by means of several operators, like the choice or parallel operators.

Process algebras are designed to describe the behaviour of distributed and concurrent processes, where processes are defined in terms of their communication with other components. In this way, it is easier to describe processes, since one can define smaller and simpler components, without having to deal with huge labelled transition graphs.

Once a formalism for describing systems is derived, it is necessary to define the semantics of the terms of the language and relations among them. In this case, CCS and CSP take two different approaches: CCS is given semantics in terms of labelled transitions systems and the basic process equivalence notion is that of bisimulation, that considers equivalent processes by taking into account their branching behaviour. On the other hand, the semantics of CSP terms is denotational, and each process is mapped to a suitable domain denoting its behaviour. Different semantic domains are possible for CSP, in particular the traces and the failures domains. CSP is designed around the notion of process refinement, allowing for hierarchical verification of systems: a process refines a specification if all the possible behaviours (e.g. traces) of the former are allowed by the latter.

It is desirable that relations defined for a model are compositional with respect to the operators of the language: in this way, it is possible to replace a component by an equivalent one, without changing the semantics of the whole system, and the analysis of a large system can be reduced to the analysis of its smaller components. This again stresses the need for a compositional approach.

Direct verification can also be applied to systems: in this case a system is checked against a specification, usually given in terms of a formula of a variant of temporal logic. Extensive research in the area of model checking has been done to address this problem, leading to several automatic tools. Direct verification is strongly connected with the definition of equivalence relations defined for a model, as we want equivalent systems to respect the same properties. This, of course, poses the problem of what needs to be preserved, thus leading to possibly different relations.

So far we have have given a brief and partial description of the framework that one would like to set up for the modelling and verification of concurrent systems. Such an approach has been extended to consider systems with more general behaviour. In particular, we are interested in real-time systems and stochastic systems. Timed systems extend traditional systems by incorporating the notion of time, either by explicitly annotating actions with the delay associated with them, or by using auxiliary constructs like clocks (e.g., timed automata [AD91]). Similarly, stochastic systems augment traditional systems with probability distributions, either as distributions regulating the passage of time (e.g., [Hil96, Bra02, D'A99]), or as discrete choice between processes (e.g., [LS91b, Seg95b]), or both. Chapter 3 and Chapter 7 will give a review of existing extensions, both from the point of view of transition systems and process algebras.

In this thesis we focus on trace semantics. Given a system, a trace is a sequence of actions that the system can perform during one of its possible runs. The trace model is one of the possible classical CSP semantic domains and, in many ways, it is one of the simplest semantic models. By describing a process in terms of its traces, only its linear-time behaviour is considered, and all the branching information is ignored. Intuitively, trace semantics describes the behaviour that one can observe in a system without interacting with it.

Trace semantics is limited as it can only be used to verify or preserve safety properties, that is, properties of the type "nothing bad can happen". Consider a process $P$ and the corresponding set of traces $\mathcal{T}(P)$; every process equivalent to $P$ must have the same set of traces. In this way, only the behaviour described by $P$ is allowed. The trace approach also suits the hierarchical approach by considering the problem of language inclusion. In this

case, process $P$ denotes a specification and all the possible allowed behaviours; in order for an implementation process to refine $P$, its set of traces must be included in $\mathcal{T}(P)$. Note, in particular, that a process can refine $P$ by not doing anything: this extreme case highlights how trace semantics can only enforce safety properties by disallowing bad behaviour, without forcing implementations to do anything. As a consequence, trace semantics is not powerful enough to enforce liveness properties or to guarantee the willingness of a process to communicate with the environment. Trace semantics can be automatically verified; in particular, in the context of CSP, refinement relations can be verified with the model checking tool FDR2 [For93].

The trace semantic model can be extended to reason about timed processes and stochastic processes. In the case of timed processes, traces also record the absolute instant at which actions are performed. Like in the untimed case, timed trace semantics can be used to preserve timed safety properties. For instance, in Chapter 5, we show how to enforce the preservation of the following timed safety property: if an action $a$ has occurred, then some other action $b$ cannot occur for $n$ time units after the occurrence of $a$.

In the case of stochastic processes, the notion of traces is replaced by that of distributions over the set of traces. Also, in the probabilistic case, trace semantics can be use to verify and preserve probabilistic safety properties. For example, the following property can be preserved by trace distribution refinement: if an event $a$ has occurred, then the probability that some other event $b$ occurs within $n$ time units is not greater than $p$.

However, several difficulties are introduced when extending the trace model: in particular, while working with infinite state systems (induced by the continuous representation of time), decidability of equivalence and refinement relations is lost. Moreover, trace semantics is not compositional when probabilistic behaviour is introduced.

We also discuss bisimulation relations: these are widely considered the standard and finest acceptable equivalence relations for concurrent processes. Bisimulation relates states of systems if they enable the same transitions, leading to equivalent states. Intuitively, bisimulation is more refined than trace semantics because two processes are bisimilar if, at each step of the computation, they are willing to offer the same set of actions to the environment. This allows for the preservation and specification of liveness properties. Bisimulation has been widely studied, and also extended to real-time and stochastic systems, and it enjoys stronger decidability properties than trace semantics.

When discussing real-time systems, we also consider failure semantics; this is finer than trace equivalence but not bisimulation equivalence. Failure semantics augments trace semantics with branching information, by annotating each trace with the set of actions that can be refused after its execution. In Chapter 2 we give a more detailed description of all these relations.

## 1.1 Aims and Contributions

The unifying theme of this thesis is the definition of process algebras for real-time and stochastic systems with equivalence and refinement relations based on trace semantics. We will consider the two extensions separately, firstly analysing a real-time process algebra, then extending it with stochastic behaviour.

Notwithstanding this common theme, we will take two alternative approaches: the different state of the art and the difficulties that are intrinsic to the problem in either area have led us to state different objectives and, consequently, we will tackle the problems in different ways.

### 1.1.1 Real-time Systems

We aim to define a process algebra for real-time systems inspired by CSP that models timed automata and is equipped with relations based on trace semantics. In this case, we focus on decidability issues: the objective is to obtain refinement and equivalence relations that can be automatically verified.

Timed extensions to CSP have already been proposed, most notably Timed CSP [RR88]. However, they have never been able to match the success of CSP due to the lack of verification techniques and the limitations of the traditional approach to verification based on refinement.

We take an approach similar to that of CSP and give the language a denotational semantics. Since we model timed automata, we avoid the undecidability issues arising from using the standard timed trace semantics and use their successful decidable quotienting techniques in order to define a discrete semantics that still takes into account the passing of time. This is done by recording the relationship between the values of clocks at the time actions are performed. We define a discrete denotational semantics which is equivalent to the operational semantics in terms of timed automata, and we can adapt existing techniques to the verification of refinement relations.

The main contribution of this part is a proposal for a new semantic domain for a real-time CSP-based process algebra. This approach allows for the automatic verification of refinement relations, both in the trace and failure models. However, this comes at a price, and we highlight the restrictions that are necessary to achieve this and the limitations of our approach.

### 1.1.2 Stochastic Systems

The aim for this part is the definition of a process algebra for real-time systems extended with arbitrary probability distributions and equipped with relations based on trace semantics.

In this case, the problem is more complex than in the case of real-time systems and even formulating a correct definition of trace semantics requires a better understanding of the model. For this reason, our focus is an analysis of the problems arising from the interaction of non-deterministic and probabilistic behaviour in systems with continuous state spaces and we concentrate on the definition of a transition-based model that will be used to give semantics to a stochastic process algebra.

We use this model to study the restrictions that are necessary in order to achieve a mathematically sound definition of trace distributions and, in general, of those properties that involve the analysis of a system over several steps of computation. We also observe that, in this setting, a more general definition of bisimulation relations is needed if we want them to be strong enough to preserve trace equivalence.

Finally, we show how a simple CSP-based stochastic process algebra can be assigned semantics in this model, thus justifying the work described above, without necessarily aiming to give a complete approach to the modelling of stochastic systems.

We believe that this part constitutes progress in the understanding of systems with continuous state spaces and, in particular, of the interaction between non-determinism and probabilistic behaviour.

**Published work.** The work on the process algebra for real-time systems has been presented at AVoCS'03 [CK03], and an extended journal version has been accepted for publication [CK05]; both papers are co-authored with Marta Kwiatkowska. A further paper

concerned with the interaction of non-determinism and probabilities [CSKN05] has been presented at FOSSACS'05; this paper is co-authored with Roberto Segala, who formulated the initial proposal for the model of stochastic transition systems and identified the need for a more general definition of bisimulation in the continuous case, Marta Kwiatkowska and Gethin Norman, who provided technical support.

## 1.2 Structure of the Thesis

This thesis is divided into two main parts, each of which can be read independently. Chapter 2 serves as an introduction to the notions that will recur throughout the thesis, and also gives a short summary of the notions of measure theory that are needed for Part II. After this chapter, the thesis is structured as follows:

- **Part I: Real-Time Systems.**
  - **Chapter 3** gives a review of existing approaches for the modelling of real-time systems.
  - **Chapter 4.** We present the real-time process algebra Clocked CSP, as an extension to standard CSP with clocks; the language is given an operational semantics in terms of timed automata. We also discuss the restrictions that are needed in order to obtain a compositional semantics.
  - **Chapter 5.** Clocked CSP is given a CSP-style denotational semantics, both in terms of traces and failures. The semantic domain is modelled on region automata, a discretisation of the infinite state space induced by timed automata. We show that the operational and denotational semantics are congruent, thus allowing for the automatic verification of refinement relations and chains of refinement.
  - **Chapter 6.** We make some concluding remarks and highlight the next steps to improve on our approach.
- **Part II: Stochastic Systems.**
  - **Chapter 7** gives a review of existing approaches to probabilistic modelling.
  - **Chapter 8.** We introduce the model of stochastic transition systems and study the restrictions needed in how non-determinism is resolved in order to obtain tractable executions.
  - **Chapter 9.** Using the results of Chapter 8, we define trace semantics for stochastic transition systems. We also define a more general notion of bisimulation and show that it is sound with respect to trace equivalence.
  - **Chapter 10.** A simple process algebra for stochastic systems is presented and given semantics in terms of stochastic transition systems. Trace and bisimulation equivalence relations are also defined.
  - **Chapter 11.** In this final chapter we summarise the contribution of our work concerning stochastic systems, and discuss possible directions of future work.

# Chapter 2

# Preliminaries

This chapter introduces the notions needed in the thesis which have been presented in the literature. Section 2.1 introduces labelled transition systems and their basic equivalence notions, Section 2.2 introduces the basic concepts of process algebras, focusing in particular on CSP, and Section 2.3 summarises the notions of measure and probability theory that will needed in the second part of this thesis.

## 2.1 Labelled Transition Systems

Graph-based representations are probably the most successful formalism for the modelling of concurrent systems. Labelled transition systems [Plo81, Kel76] are one such model: a labelled transition system is a collection of nodes and edges connecting them. Nodes represent the possible states that the system can be in, and the edges represent the possible transitions between states. Edges are also labelled by the actions that a system performs during the transition.

**Definition 2.1 (Labelled Transition System).** A labelled transition system (lts) is a tuple:

$$L = (Q, \overline{q}, \Sigma, \rightarrow)$$

where $Q$ is the set of states, $\overline{q}$ is the initial state, $\Sigma$ is the alphabet of possible actions and $\rightarrow \subseteq Q \times (\Sigma \cup \{\tau\}) \times Q$ is the transition relation.

We write $p \xrightarrow{a} q$ whenever $(p, a, q) \in \rightarrow$. The alphabet $\Sigma$ does not include the special internal action $\tau$; we say that all actions in $\Sigma$ are visible.

Systems can be composed by means of parallel composition. Two main approaches on how the parallel operator is defined can be found in the literature: in the process algebra CCS [Mil89], each action has a compatible action that it can synchronise with, and the result of the synchronisation is a $\tau$ action, denoting an internal computation. The process algebra CSP [Hoa85] defines a multi-way parallel operator, whereby processes synchronise on common actions (or actions in an interface alphabet); after the synchronisation, the corresponding action is still visible and it can be used for further synchronisation.

**Definition 2.2 (Parallel Composition).** Given two labelled transition systems $L_1 = (Q_1, \overline{q}_1, \Sigma, \rightarrow_1)$ and $L_2 = (Q_2, \overline{q}_2, \Sigma, \rightarrow_2)$, the parallel composition with respect to the interface alphabet $A \subseteq \Sigma$ is the new labelled transition system:

$$L_1 \underset{A}{||} L_2 = (Q_1 \times Q_2, (\overline{q}_1, \overline{q}_2), \Sigma, \rightarrow)$$

where $\rightarrow$ is defined as follows:

- if $a \in A$, then $(p_1, p_2) \xrightarrow{a} (q_1, q_2)$ if $p_1 \xrightarrow{a} q_1$ and $p_2 \xrightarrow{a} q_2$;
- if $a \notin A$, then $(p_1, p_2) \xrightarrow{a} (q_1, p_2)$ if $p_1 \xrightarrow{a} q_1$ or $(p_1, p_2) \xrightarrow{a} (p_1, q_2)$ if $p_2 \xrightarrow{a} q_2$ .

When analysing the behaviour of a system over a run, the relevant information lies in the sequence of states and actions that the system has stepped trough during its evolution. This corresponds to the notion of execution.

An execution for an lts $L$ is an alternating sequence of states and actions $q_0 a_1 q_1 a_2 \ldots$, either infinite or, if finite, ending with a state, such that, for all $i$, $(q_i, a_{i+1}, q_{i+1}) \in \rightarrow$. Denote the length of a finite execution $\alpha = q_0 a_1 q_1 \ldots a_n q_n$ by $|\alpha| = n$, and, if $\alpha$ is infinite, by $|\alpha| = \infty$. Given a finite execution $\alpha = q_0 a_1 \ldots a_n q_n$, let $s$ be the ordered sequence of actions appearing in $\alpha$; then we extend the $\rightarrow$ notation and write $q_0 \xrightarrow{s} q_n$. It is possible to abstract from internal computations, that is, $\tau$ actions as follows: given a finite execution $\alpha = q_0 a_1 \ldots a_n q_n$, let $t$ denote the ordered sequence of visible actions in $\alpha$; then we write $q_0 \stackrel{t}{\Rightarrow} q_n$. A particular case is that of *weak transitions*: a weak transition is a sequence of transitions whose visible trace is either a single action $a \in \Sigma$ or no action at all; a weak transition is denoted by $q \stackrel{a}{\Rightarrow} q'$, with $a \in \Sigma \cup \{\tau\}$.

This leads to the notion of traces: a trace is sequence of actions in $\Sigma$ and carries the information about the visible part of an execution. We denote a trace by the list of actions: $t = \langle a_1 a_2 \ldots \rangle$. Given a finite trace $t_1 = \langle a_1 \ldots a_n \rangle$ and a trace $t_2 = \langle b_1 b_2 \ldots \rangle$, we denote their concatenation by $t_1 \hat{} t_2 = \langle a_1 \ldots a_n b_1 b_2 \ldots \rangle$. Given an action $a \in \Sigma$ and a trace $t$, we often write $a \hat{} t$ instead of the more precise $\langle a \rangle \hat{} t$.

### 2.1.1 Equivalences for Labelled Transition Systems

**Bisimulation.** Bisimulation relations were introduced for the process algebra CCS. They have arguably become the most successful and widely used notion of equivalence for concurrent systems. A bisimulation is an equivalence relation on the set of states of a system that equates two states if every transition of one state can be matched by a transition with the same label of the other, and if such two transitions lead to equivalent states.

**Definition 2.3 (Strong Bisimulation).** Two labelled transition systems $L_1$ and $L_2$ are bisimilar, denoted by $L_1 \sim L_2$, if there exists an equivalence relation $\sim$ on $Q_1 \cup Q_2$ such that:

- $\overline{q}_1 \sim \overline{q}_2$ and
- whenever $q_1 \sim q_2$ and $q_1 \xrightarrow{a} q_1'$, then there exists a transition $q_2 \xrightarrow{a} q_2'$ such that $q_1' \sim q_2'$.

According to the definition above, bisimulation does not abstract from internal computation and each $\tau$ action of a system must be matched by a $\tau$ action of the other. For this reason, *weak* bisimulation $\approx$ was defined: in this case, two states $q_1$ and $q_2$ are weakly bisimilar if every transition $q_1 \xrightarrow{a} q_1'$ is matched by a weak transition of $q_2$, that is a transition $q_2 \stackrel{a}{\Rightarrow} q_2'$, such that $q_1' \approx q_2'$. It is clear that strong bisimulation implies weak bisimulation, since normal transitions are special cases of weak transitions.

**Trace semantics.** A less refined way to identify two systems as equivalent is if they present the same external behaviour, that is, if they have the same set of traces. Define a function *trace*, that, given an execution $\alpha$, returns the corresponding trace. Then the

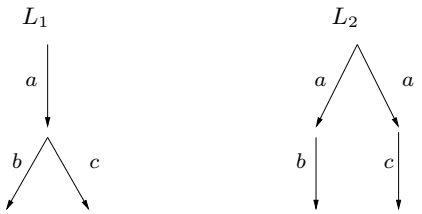



Figure 2.1: Two systems that are trace equivalent but not failure equivalent.

(finite) trace semantics $\mathcal{T}(L)$ of a labelled transition system $L$ is defined as the set of traces $trace(\alpha)$, for all finite executions $\alpha$, or, equivalently:

$$\mathcal{T}(L) = \{t \in \Sigma^* | \exists q.\overline{q} \stackrel{t}{\Longrightarrow} q\}.$$

It is possible to give a similar trace semantics based on infinite traces. Trace semantics abstracts from the branching structure of a system and therefore is often not fine enough. However, it benefits from simplicity and it can be used for the specification of safety properties: a system $L_1$ (the specification) describes all the safe, allowed behaviours. The implementation system respects such safety properties if its behaviour in terms of traces is contained in the behaviour of the specification. This corresponds to the concept of *refinement* and is defined as follows: a system $L_2$ refines a specification $L_1$ if the traces of $L_2$ are a subset of the traces of $L_1$:

$$L_1 \sqsubseteq_{\mathcal{T}} L_2 \text{ iff } \mathcal{T}(L_1) \supseteq \mathcal{T}(L_2)$$

If two systems are equivalent under the trace semantics (that is, they have the same sets of traces), they respect the same set of safety properties.

Finally, trace equivalence is a weaker notion of equivalence than bisimulation, as stated by the following;

**Theorem 2.4.** *Given two labelled transition systems $L_1$ and $L_2$, if $L_1 \approx L_2$ then $\mathcal{T}(L_1) = \mathcal{T}(L_2)$.*

**Failure Semantics.** Trace semantics ignores the branching structure of a system and therefore is often viewed as inadequate. Consider the systems of Figure 2.1: even if they have the same trace semantics, they have a different behaviour after executing the first $a$ action: $L_1$ can still synchronise on both $b$ and $c$, while the choice has already been made for $L_2$, that can only synchronise on one action (note that the two systems are not bisimilar). For this reason, the notion of failures was introduced [BHR84]. Informally, a failure is a trace enriched with the information of the actions that are rejected after the execution of such trace. For instance, $L_1$ of Figure 2.1 accepts both $b$ or $c$ after an $a$, while $L_2$ can always refuse one of them, no matter which $a$ transition was executed. It follows that the two systems are not failure equivalent.

Formally, a state $q$ refuses an action $a$ if there is no state $q'$ such that $(q, a, q') \in \rightarrow$. We say that $q$ is stable if there is no $\tau$-labelled transition leaving $q$. We define the refusal set of a stable state $q$ as follows: given a set of actions $F$, $q \,\mathsf{ref}\, F$ if $q$ is stable and $q$ refuses all actions in $F$. We can define the stable failures of an lts $L$ as follows:

$$\mathcal{F}(L) = \{(t, F) \mid \exists q.\overline{q} \stackrel{t}{\Longrightarrow} q \text{ and } q \,\mathsf{ref}\, F\}.$$

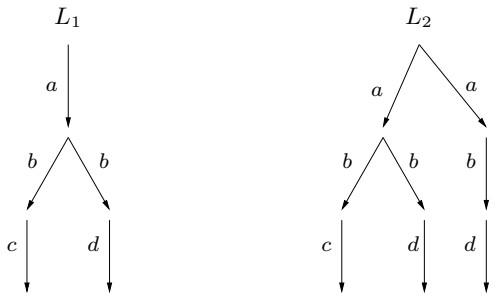



Figure 2.2: Two systems that are failure equivalent but not bisimilar.

As for trace semantics, failure equivalence is defined as equality of sets of failures, and failure refinement $\sqsubseteq_{\mathcal{F}}$ as inverse set inclusion of $\mathcal{F}(L)$.

It is easy to check that failure equivalence implies trace equivalence and that bisimulation implies failure equivalence. However, bisimulation is strictly more distinguishing than failure equivalence, as shown by the two systems of Figure 2.2, that are failure equivalent but not bisimilar. We have the following results:

**Theorem 2.5.** *Given two labelled transition systems $L_1$ and $L_2$, if $L_1 \approx L_2$ then $\mathcal{F}(L_1) = \mathcal{F}(L_2)$.*

**Theorem 2.6.** *Given two labelled transition systems $L_1$ and $L_2$, if $\mathcal{F}(L_1) = \mathcal{F}(L_2)$ then $\mathcal{T}(L_1) = \mathcal{T}(L_2)$.*

### 2.1.2 Discussion

In this section we have presented only the equivalence relations that we consider in this thesis. More can be defined, e.g. the failure/divergences model [BR84, Ros98], testing equivalences [DH84], or ready simulation [LS91b, BIM95]. When choosing which equivalence relation to adopt for a given model, several factors have to be taken into account. Firstly, we need to decide what properties are to be preserved by the equivalence relation: while trace semantics can only preserve safety properties, bisimulation is more powerful, but at the same time it might be too fine for some purposes. Secondly, we would like the equivalence relations be *compositional* with respect to the constructs of the formalism: for example, if we have two equivalent systems, we would like their parallel composition with a third system to be equivalent. This allows for compositional reasoning, and components can be substituted by equivalent ones. The notions we have defined above, for instance, are compositional with respect to parallel composition. However, we will show that this is not always the case if we extend the models.

## 2.2 Process Algebras and CSP

Graph-based models for the specification of systems are very simple and intuitive, but they are limited and not suitable for high-level specification of large systems. Process algebras are an alternative method for the specification of concurrent systems, and provide a language to describe systems at a higher level, by means of constructs that allow one to compose systems from their smaller components. Process algebras are often mapped onto labelled transition systems; therefore the two approaches are not exclusive and they often complement each other and use the same notions.

Among the first proposals we find two process algebras, CCS [Mil89] and CSP [BHR84, Hoa85]. CCS is given an operational semantics in terms of labelled transition systems and equivalences between processes founded on the notion of bisimulation. CSP, on the other hand, was first given a denotational semantics in terms of traces, failures and failure/divergences; only later it was given a congruent operational semantics. In this thesis, we take a hybrid approach: while our process algebras are inspired by CSP and we give them trace based semantics, we rely on operational models to guarantee the decidability of the trace-based equivalences/refinements (Part I), or define the trace semantics of the language (Part II). In this section we briefly describe CSP, its operators, and its semantic model.

### 2.2.1 CSP

CSP is a process algebra introduced by Tony Hoare [Hoa85]. It describes concurrent systems in terms of their sequential components, characterised by the sequences of actions that they can perform. CSP processes over the action alphabet $\Sigma$ (not including the special silent action $\tau$) are generated by the following syntax:

$$\begin{aligned} P \quad ::= \quad & STOP \mid SKIP \mid a \rightarrow P \mid P \sqcap P \mid P \,\Box\, P \mid \\ & P \underset{A}{||} P \mid P \setminus A \mid f[P] \mid X \mid \mathsf{rec}\, X.P \mid P \,;\, P \end{aligned}$$

where $a \in \Sigma$, $A \subseteq \Sigma$ and $f : \Sigma \rightarrow \Sigma$ is a bijective renaming function. The operators above represent, respectively: deadlock, successful termination, action prefix, internal choice, external choice, interface parallel, hiding, renaming, process name, recursion and sequential composition. We use only one type of parallel operator, as the others, e.g. interleaving, can be expressed in terms of the interface parallel. Table 2.1 gives the operational semantics rules mapping CSP terms onto labelled transition systems, whose state space consists of the CSP terms (plus the special state $\Omega$, denoting the process that has successfully terminated), and whose alphabet includes the special action $\checkmark$ that denotes successful termination. Note, in particular, that CSP has two choice operators. One, internal choice, denotes pure non-determinism as the environment has no control on how the choice is resolved. The other, external choice, lets the environment decide which action is to be executed; however, either processes can perform internal actions before the choice is resolved.

The semantics of a CSP term $P$ is given by the set of sequences of actions that it can perform (*traces*), the set of pairs containing a trace and the actions that can be refused after it (*failures*) or the set of failures and the set of traces causing infinite executions of internal actions (*divergences*). While all these notions are defined denotationally and rely on concepts from metric spaces and domain theory, they have been shown congruent to the operational model.

Different relations are built upon these semantic models; for each of them, equivalences between processes are defined as set equalities. CSP also introduces the idea of *refinement*: a process $P_1$ is refined by another process $P_2$ ($P_1 \sqsubseteq P_2$) if every behaviour of $P_2$ is a possible behaviour of $P_1$, that is, if it is "less non-deterministic". This idea is formally defined as inverse set inclusion of traces, failures or divergences. For a detailed description, see introductory books to CSP, e.g. [Ros98] or [Sch99]. FDR2 [For93] is an efficient commercial tool for the automatic verification of these refinement relations for CSP processes.

$$\frac{}{SKIP \xrightarrow{\checkmark} \Omega} \qquad \frac{}{(a \to P) \xrightarrow{a} P}$$

$$\frac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P'}\ a \neq \tau \qquad \frac{P \xrightarrow{\tau} P'}{P \,\Box\, Q \xrightarrow{\tau} P' \,\Box\, Q}$$

$$\frac{Q \xrightarrow{a} Q'}{P \,\Box\, Q \xrightarrow{a} Q'}\ a \neq \tau \qquad \frac{Q \xrightarrow{\tau} Q'}{P \,\Box\, Q \xrightarrow{\tau} P \,\Box\, Q'}$$

$$\frac{}{P \sqcap Q \xrightarrow{\tau} P} \qquad \frac{}{P \sqcap Q \xrightarrow{\tau} Q}$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \underset{A}{||} Q \xrightarrow{a} P' \underset{A}{||} Q'}\ a \in A$$

$$\frac{P \xrightarrow{a} P'}{P \underset{A}{||} Q \xrightarrow{a} P' \underset{A}{||} Q}\ a \notin A \qquad \frac{Q \xrightarrow{a} Q'}{P \underset{A}{||} Q \xrightarrow{a} P \underset{A}{||} Q'}\ a \notin A$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P'}\ a \in A \qquad \frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{a} P'}\ a \notin A$$

$$\frac{P \xrightarrow{a} P'}{f[P] \xrightarrow{f(a)} f[P']} \qquad \frac{P \xrightarrow{a} P'}{\mathsf{rec}\, X.P \xrightarrow{a} P'[\mathsf{rec}\, X.P/X]}$$

$$\frac{P \xrightarrow{a} P'}{P; Q \xrightarrow{a} P'; Q}\ a \neq \checkmark \qquad \frac{P \xrightarrow{\checkmark} P'}{P; Q \xrightarrow{\tau} Q}$$

Table 2.1: Operational semantics for CSP.

## 2.3 Probability and Measure Theory Background

In this section we review the basic definitions and results of measure theory that are necessary in the second part of this thesis. A basic knowledge of topology and metric spaces is assumed. Most results can be found in standard textbooks, e.g. [Ash72, Bil95, Dud89]; [Pan97] serves as good introduction to measure theory and its importance in concurrency theory.

### 2.3.1 Basic Definitions

**$\sigma$-algebras.** Given a set $X$, an *algebra* over $X$ is a family $F_X$ of subsets of $X$ that includes $X$ and is closed under complementation and finite union; $\mathcal{F}_X$ is a *$\sigma$-algebra* over $X$ if we additionally require closure under countable union. A *measurable space* is a pair $(X, \mathcal{F}_X)$, where $\mathcal{F}_X$ is a $\sigma$-algebra over $X$. The elements of $\mathcal{F}_X$ are called *measurable sets*. We abuse the notation and refer to $X$ as a measurable space whenever the corresponding

$\sigma$-algebra is clear from the context. The $\sigma$-algebra (resp., algebra) generated by a family $G$ of subsets of $X$ is the *smallest* $\sigma$-algebra (resp, algebra) including $G$. Even if more than one $\sigma$-algebra can be defined over one set, we always identify the $\sigma$-algebra corresponding to one set by the means of subscripts whenever this leads to no ambiguities.

Given a topological space $(X, T)$*, a particular $\sigma$-algebra that we are interested in is the *Borel* $\sigma$-algebra, the $\sigma$-algebra generated by the open sets of the topology, and we denote it by $\mathcal{B}(X)$. This is the $\sigma$-algebra that is used on the set of the real numbers.

**Theorem 2.7.** *Let $G$ be a family of subsets of $X$.*

1. *Define $F_1(G)$ to be the family containing $\emptyset$, $X$, and the complements of all the sets in $G$.*
2. *Define $F_2(G)$ to be the family containing all finite intersections of elements in $F_1(G)$.*
3. *Define $F_3(G)$ to be the family containing all finite unions of disjoint elements in $F_2(G)$.*

*Then $F_3(G)$ is to the algebra generated by $G$.*

**Measures.** Given a measurable space $(X, \mathcal{F}_X)$, a *measure* over $(X, \mathcal{F}_X)$ is a function $\mu : \mathcal{F}_X \to \mathbb{R}^{\geq 0}$ such that, for every countable family of pairwise disjoint measurable sets $\{A_i\}_{i\in I}$, $\mu(\cup_{i\in I} A_i) = \sum_{i\in I} \mu(A_i)$. The triple $(X, \mathcal{F}_X, \mu)$ is called a *measure space*. A *probability* (resp., *sub-probability*) measure $\mu$ over $(X, \mathcal{F}_X)$ is a measure such that $\mu(X) = 1$ (resp., $\mu(X) \leq 1$). A measurable set whose complement has probability 0 is called a *support* for a measure $\mu$. If $\mu$ is a (sub-)probability measure, $(X, \mathcal{F}_X, \mu)$ is called a *(sub-)probability space*. We denote the set of probability (resp., sub-probability) measures over $(X, \mathcal{F}_X)$ by $\mathcal{D}(X, \mathcal{F}_X)$ (resp., $sub\mathcal{D}(X, \mathcal{F}_X)$). We sometimes use the term probability distribution to denote a probability measure. A particular and recurring family of probability measures is the Dirac measures: given a measurable space $(X, \mathcal{F}_X)$ and $x \in X$, the Dirac measure for $x$ is defined as $dirac(x)(A) = 1$ if $x \in A$, $dirac(x)(A) = 0$ otherwise.

**Measurable functions.** A function $f : (X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$ is *measurable* if the pre-image of every measurable set $B \in \mathcal{F}_Y$ is measurable, that is, if $f^{-1}(B) = \{x \in X \mid f(x) \in B\} \in \mathcal{F}_X$. Given a measurable space $(X, \mathcal{F}_X)$, the indicator function for a measurable set $A \in \mathcal{F}_X$ is the measurable function $I_A(x) = 1$ if $x \in A$, and 0 otherwise. Let $(X, \mathcal{F}_X, \mu)$ be a probability space, $(Y, \mathcal{F}_Y)$ a measurable space and $f$ a measurable function from $X$ to $Y$. The induced probability measure for $f$ over $(Y, \mathcal{F}_Y)$ is given by $f(\mu)$ defined as $f(\mu)(B) = \mu(f^{-1}(B))$ for all $B \in \mathcal{F}_Y$.

**Theorem 2.8.** *Let $f$ be a function from $(X, \mathcal{F}_X)$ to $(Y, \mathcal{F}_Y)$, both measurable spaces and $G$ be a family of subsets of $Y$ generating $\mathcal{F}_Y$. Then $f$ is measurable iff, for each element $A \in G$, we have that $f^{-1}(A) \in \mathcal{F}_X$.*

*Given a set $X$, a family $T$ of subsets of $T$ is a topology for $X$ if

- $\emptyset \in T$ and $X \in T$,
- $T$ is closed under finite intersection,
- $T$ is closed under arbitrary union.

The pair $(X, T)$ is called a topological space.

**Product spaces and product measures.** The *product space* of two measurable spaces $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$ is the measurable space $(X \times Y, \mathcal{F}_X \otimes \mathcal{F}_Y)$, where $\mathcal{F}_X \otimes \mathcal{F}_Y$ is the $\sigma$-algebra generated by the *rectangles* $A \times B = \{(x, y) \mid x \in A, y \in B\}$, for all $A \in \mathcal{F}_X$ and $B \in \mathcal{F}_Y$; we alternatively denote $\mathcal{F}_X \otimes \mathcal{F}_Y$ by $\mathcal{F}_{X\times Y}$. The union of two measurable spaces is the measurable space $(X \cup Y, \mathcal{F}_{X\cup Y})$, where $\mathcal{F}_{X\cup Y}$ is the $\sigma$-algebra generated by the union of $\mathcal{F}_X$ and $\mathcal{F}_Y$.

Given two measure spaces $(X, \mathcal{F}_X, \mu_X)$ and $(Y, \mathcal{F}_Y, \mu_Y)$, they induce a measure $\mu$ on the product space $(X \times Y, \mathcal{F}_{X\times Y})$ defined on rectangles $A \times B$, $A \in \mathcal{F}_X$ and $B \in \mathcal{F}_X$, as $\mu(A\times B) = \mu_X(A)\times\mu_Y(B)$. This measure is uniquely extended to the $\sigma$-algebra generated by the rectangles and it is denoted by $\mu_X \otimes \mu_Y$. Given a measure on the product space, we call the measures induced by the projection functions $\pi_i$, $i = 1, 2$, the *marginal* measures $\mu_i$, $i = 1, 2$.

**Extensions of measures.** We often define a measure only on a subset of the $\sigma$-algebra of a measure set. We identify a few conditions that are sufficient to guarantee that the measure is uniquely extended to the whole $\sigma$-algebra.

**Theorem 2.9.** *Let $\mu_1$ and $\mu_2$ be two measures defined on a measurable space $(X, \mathcal{F}_X)$. If $\mu_1$ and $\mu_2$ agree on all the members of an algebra $F$ generating $\mathcal{F}_X$, then $\mu_1$ and $\mu_2$ coincide.*

We call a family $S$ of subsets of a set $X$ a *semi-ring* if $S$ includes $\emptyset$, is closed under finite intersection, and if whenever $A, B \in S$, then there exists a finite family $\{A_i\}_{i\in\{0\ldots n\}}$ of pairwise disjoint elements of $S$ such that $A \setminus B = \cup_{i=0}^{n} A_i$.

**Theorem 2.10.** *Let $\mu_1$ and $\mu_2$ be two measures defined on a measurable space $(X, \mathcal{F}_X)$. If $\mu_1$ and $\mu_2$ agree on all the members of a semi-ring generating $\mathcal{F}_X$, then $\mu_1$ and $\mu_2$ coincide.*

### 2.3.2 Regular Conditional Probabilities

As we will demonstrate later, our construction will require conditional probabilities. In the discrete case, we can define the probability of an event $A$ given $B$ as $P(A|B) = P(A \cap B)/P(B)$, which is defined only when $P(B) > 0$. Unfortunately, this cannot be done in general for the continuous case, as it is still meaningful to condition with respect to events of probability 0. It is therefore necessary to extend the concept of conditional probabilities.

**Definition 2.11.** Let $(X, \mathcal{F}_X, \mu)$ be a probability space, $(Y, \mathcal{F}_Y)$ a measurable space and $f : (X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$ a measurable function. A **regular conditional probability** for $\mu$ with respect to $f$ is a function $\nu : Y \times \mathcal{F}_X \to [0, 1]$ such that:

1. $\nu(y, \cdot)$ is a probability measure on $\mathcal{F}_X$, for each $y \in Y$;
2. $\nu(\cdot, A)$ is a measurable function on $(Y, \mathcal{F}_Y)$, for each $A \in \mathcal{F}_X$;
3. $\mu(A \cap f^{-1}(B)) = \int_B \nu(y, A)\, f(\mu)(dy)$.

Regular conditional probabilities do not exist for all probability spaces. It is necessary to impose restrictions on the kind of measurable spaces we consider. A *Polish space* is the topological space underlying a complete separable metric space. Given a Polish space $X$, $(X, \mathcal{F}_X)$ is a standard Borel space if $\mathcal{F}_X$ is the Borel $\sigma$-algebra generated by the topology. Finally, given a standard Borel space $(X, \mathcal{F}_X)$, $Y \subseteq X$ is an *analytic set* if it is the

continuous image of some Polish space. $(Y, \mathcal{F}_Y)$ is an *analytic space* if it is measurably isomorphic to an analytic set in a Polish space, that is, if there exists a measurable function whose inverse is also measurable. Note that singleton sets are measurable in Polish and analytic spaces.

Examples of analytic sets are the discrete spaces and any open or closed subset of the reals equipped with the Borel $\sigma$-algebra. Analytic sets are closed under union and Cartesian product. Thus, analytic sets are quite general; for instance, the semantic model of timed systems can be given by the product of a discrete set (the states in the symbolic representation of a system) and the possible values of time (the real numbers).

**Theorem 2.12.** *Let $f : (X, \mathcal{F}_X, \mu) \to (Y, \mathcal{F}_Y)$ be a measurable function and $(Y, \mathcal{F}_Y)$ an analytic space. Then there exists a regular conditional probability $\nu$ for $f$.*

**Theorem 2.13.** *Let $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$ be two measurable spaces and $f$ a measurable function from $X \times Y$ to the non-negative reals equipped with the corresponding Borel $\sigma$-algebra. Assume we have a function $\nu : Y \times \mathcal{F}_X \to \mathbb{R}^{\geq 0}$ such that $\nu(y, \cdot)$ is a measure on $(X, \mathcal{F}_X)$ for all $y \in Y$ and $\nu(\cdot, A)$ is measurable for all $A \in \mathcal{F}_X$. Then $\int_X f(x, y)\nu(y, dx)$ exists and is a measurable function of $Y$.*

**Theorem 2.14.** *Let $(X, \mathcal{F}_X, \mu)$ and $(Y, \mathcal{F}_Y, \nu)$ be measure spaces and $\{\mu_y\}_{y \in Y}$ be a family of measures on $(X, \mathcal{F}_X)$ such that $\mu(A) = \int_Y \mu_y(A)\nu(dy)$ for all $A \in \mathcal{F}_X$. Then $\int_Y \int_X f(x)\mu_y(dx)\nu(dy)$ is defined and equal to $\int_X f d\mu$ for all $[-\infty, \infty]$-valued functions $f$ such that $\int_X f d\mu$ is defined in $[-\infty, \infty]$.*

### 2.3.3 A $\sigma$-algebra on Probability Measures

In the following, we define probability distributions on sets of probabilistic transitions, that is, transitions whose target is a probability measures on states. We therefore need to define a $\sigma$-algebra on sets of probability measures; we use the standard construction, due to Giry [Gir81]. Let $(X, \mathcal{F}_X)$ be a measurable set and $\mathcal{D}(X, \mathcal{F}_X)$ the set of probability measures on $X$. We build a $\sigma$-algebra on the set of probability measures $\mathcal{D}(X, \mathcal{F}_X)$ as follows: for each $A \in \mathcal{F}_X$, define a function $p_A : \mathcal{D}(X, \mathcal{F}_X) \to [0, 1]$ by $p_A(\nu) = \nu(A)$. The $\sigma$-algebra on $\mathcal{D}(X, \mathcal{F}_X)$, denoted by $\mathcal{F}_{\mathcal{D}(X, \mathcal{F}_X)}$, is the *least* $\sigma$-algebra such that all the functions $p_A$ are measurable. The generators of the $\sigma$-algebra are the sets of probability measures $D_{A,I} = p_A^{-1}(I) = \{\mu \in \mathcal{D}(X, \mathcal{F}_X) \mid \mu(A) \in I\}$, for all $A \in \mathcal{F}_X$ and $I \in \mathcal{B}([0, 1])$.

# Part I

# Real-Time Systems

# Chapter 3

# Real-Time Systems

Traditional formalisms such as labelled transitions systems or process algebras enable the verification of properties of systems that depend on the order in which actions are executed, but any timing aspects are ignored. For example, one can check if the occurrence of an action $a$ precedes the occurrence of an action $b$, but no information on the amount of time that elapses between these two actions can be obtained. For this reason, such formalisms are not suitable for the verification of *real-time* systems, that is, systems for which the information about the passage of time is critical. We need more powerful techniques and tools that are capable of dealing with the notion of time formally.

Depending on the way that time itself is modelled, we can distinguish two main approaches to real-time verification: time can be represented discretely, that is, the system is observed only at discrete time points, or it could progress in a continuous way, thus behaving as a real-valued variable. The discrete approach is suitable for some applications and its relative simplicity leads to many decidable properties; it has therefore been proposed in the literature (e.g., [NS94, HR95]). However, the continuous representation is clearly preferable, since it describes real-time behaviour accurately. This is the approach that we will analyse.

Several proposals to real-time modelling and verification are present in the literature, both from the operational point of view, by extending labelled transition systems with constructs to deal with real-time (e.g. [AD94, LV91]), and from the process-algebraic point of view, by giving a timed variant of virtually every classical process algebras (e.g. Timed CSP [RR88], Timed ACP [BB91], and several timed extensions of CCS [MT90, YPD94]). As in the untimed case, the distinction between these two approaches is frequently blurred, with the operational models often used to define the semantics for process algebras.

In the following two sections we give an overview of the approaches to real-time modelling and verification that are relevant to our purposes. On the operational side, we concentrate on timed automata in Section 3.1, since this is arguably the most studied and successful model, especially for automatic verification, besides being the main model for our proposal of a process algebra. In Section 3.2 we give an overview of timed process algebras, in particular Timed CSP and others that relate to timed automata.

## 3.1 Timed Automata

Timed automata were introduced in the early nineties by Alur and Dill [AD91] (based on previous work by Dill [Dil90]), as an extension of traditional labelled systems with *clocks*, real-valued variables whose values progress at the same rate as real time.

Clocks control the execution of a system by the use of *clock constraints* (or *guards*) on

transitions. A clock constraint is a boolean expression obtained by comparing clocks $x$ against integer values. For instance, a possible guarded transition between two locations (states in timed automata terminology) $l_1$ and $l_2$ is $l_1 \xrightarrow{a,x\geq 1} l_2$: action $a$ can be executed only when the value of $x$ is greater or equal to 1. Clocks can be reset to 0 upon entering locations; even if different clocks can have different values, they always increase at the same rate as real time. Note that, in the original model [AD91], clock resets are specified on transitions rather than on locations, but it easy to show that the two models are equivalent.

The timed automata model has been enriched with the introduction of *invariants* [HNSY92]: invariants are particular clock constraints on the location that enforce the progress of the execution by forcing the system to perform an action when the invariant is not satisfied any more. Timed automata will be formally described in the next chapter in more detail.

Informally, a timed automaton evolves as follows: from each location, the automaton can either perform any one transition whose guard is currently true, or it can let time elapse, but only as long as the location invariant remains true, at which point it must execute an action.

A timed automaton is formally given semantics in terms of a labelled transition system, where each state records the location of the timed automaton and the values of all clocks (called clock valuations) and each transition is either an action transition corresponding to an edge of the timed automaton whose guard is satisfied, or a timed transition, labelled by a real value and denoting the passage of time. Since timed automata model real-time, the state space is infinite.

### 3.1.1 Fundamental Results about Timed Automata

Timed automata are a simple and elegant formalism for the modelling of real-time systems. However, their success derives from the fact that, even though they model infinite (dense and even continuous) state spaces, several of their properties are decidable and, for this reason, many model checking algorithms and tools for timed automata are available. A good introduction to timed automata and to the main results about them can be found in [Alu97].

Timed automata can also be studied from an automata-theoretic point of view, and indeed this is how they were first considered [AD91, AD96]: if we give timed automata accepting conditions on the locations, we can view them as a timed version of Büchi automata (or just of normal finite-state automata) and we can study the properties of the language $L(\mathcal{A})$ that a timed automaton $\mathcal{A}$ accepts. In the case of timed automata, timed languages are considered: timed languages are composed of timed traces (or words) that record both the actions performed and the absolute instant at which they were performed. A first fundamental result is that the emptiness of the language accepted by a timed automaton is decidable. The languages accepted by timed automata also enjoy several closure properties, like closure under finite union and intersection, but not under complementation. Unfortunately, other properties do not hold: the problem of language inclusion (see the discussion below) and that of universality (whether an automaton accepts all possible timed words) are not decidable [AD91].

Equivalence relations and preorders like timed bisimulation [Čer93], time abstract simulation [HHK95] and simulation [CVWY92, TAKB96] have been defined, by applying and adapting the traditional notions to the infinite state labelled transition systems corresponding to timed automata. Despite the fact that the transition systems have dense

state spaces, all of these relations have been shown to be decidable.

All of the positive results described above rely on the *region* construction: this is a method to obtain a finite quotient of the state space, by grouping clock valuations that are equivalent. Informally, clock valuations are considered equivalent if they behave in the same way with respect to possible clock constraints: two clock valuations are equivalent if they agree on the integral parts of the values of all clocks and on the ordering of their fractional parts. All algorithms for timed automata have the region construction as their starting point; since the size of the region graph grows very fast (exponentially in the size of constants against which clocks are compared and the number of clocks, independently [CY92]), more efficient techniques have been devised.

For the purpose of this thesis, we are interested in the problem of language inclusion for timed automata.

### Language Inclusion for Timed Automata

The problem of language inclusion can be stated as follows: given two timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$, is it possible to decide whether $L(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$? $A_1$ is the specification, denoting all the possible admissible behaviours, while $A_2$ represents an implementation, whose possible behaviours must be contained in the set of behaviours of $A_1$.

As mentioned above, this problem is undecidable for timed automata, both for infinite and finite words. Note, in particular, that standard techniques used to decide language inclusion fail since timed languages are not complementable, that is, given a timed automaton $A$, it is not always possible to find a timed automaton $A^c$ such that $L(A^c) = L(A)^c$, where the complementation is with respect to the universe of timed words. For these reasons, it is not possible to reduce the inclusion problem to the emptiness problem $L(\mathcal{A}_2) \cap L(\mathcal{A}_1)^c$; this is the standard procedure used for untimed languages.

Despite this negative result that dates back to the early works on timed automata [AD94], several attempts have been made to give timed automata semantics in terms of timed traces and to find conditions under which the language inclusion problem becomes decidable. This is still the subject of active research.

One possibility is to consider only deterministic specifications [AD94]: a timed automaton is deterministic if, for all possible locations, only one edge is enabled for a given action at a given time, that is, given an action $a$, the guards of all the $a$-labelled transitions are mutually exclusive. In this way, at each point in time, at most one transition for each action is enabled, thus achieving determinism. Of course, it is not possible in general to generate a language-equivalent deterministic timed automaton for all timed automata. Event-clock automata represent a determinisable class of timed automata [AFH99]: in this model, a different clock corresponds to each action and it is reset each time such action is performed. The automaton does not have full control of clocks but their values are determined by the actions performed. Event-clock automata are closed under complementation.

An interesting result is that closure under complementation is not necessary to obtain decidability: even though there are timed automata with only one clock that are not complementable, the language inclusion problem is decidable if we only allow specifications with one clock [OW04]. In the same paper, it is shown that the problem is decidable also if we only allow mote than one clock but only compare them to 0 in the specification automaton.

Yet another way to restrict timed automata to achieve decidability is by allowing only *open* specifications or only *closed* specifications [OW03b]: a timed automaton is open if it does not allow for equality in clock constraints, that is $<$ and not $\leq$, while it is closed if

it allows the opposite class of constraints. Furthermore, this result does not hold under strongly monotonic dense time, that is, a model of time whereby no two actions can occur at the same time.

**Digitisation.** A different and successful approach is that of *digitisation* [HMP92, Bošy99, AMP98, Oua02], that is, identifying the conditions under which the dense-time behaviour of a system can be reduced to its discrete-time fragment. In this way, the question of language inclusion under dense time semantics is reduced to the same question under discrete time semantics, thus achieving decidability. Informally, a timed trace is digitised by considering the integer values nearest to the time real values associated to each action in the trace; consequently, a timed automaton is closed under digitisation if its timed trace semantics contains the digitisation of all of its traces; similarly, a timed automaton is closed under inverse digitisation if its trace semantics can be induced by the set of its digitised traces. Finally, we can reduce to discrete-time language inclusion if we consider implementations closed under digitisation and specifications closed under inverse digitisation. Unfortunately, it has been shown that, while it is decidable whether an implementation is closed under digitisation, it is not decidable whether a specification is closed under inverse digitisation [OW03a].

All of the techniques discussed above identify subclasses of timed automata for which the language inclusion problem is decidable. However, they highlight a few drawbacks, especially for our purposes. Firstly, we lose generality because of the restrictions on the model; in a process algebra, systems are likely to be constructed by composing smaller ones via several operators, and therefore it would be desirable to place as little restriction as possible on what process can be handled. Then, we aim to have a CSP-inspired process algebra whose semantics is based on the idea of refinement; note that all of the approaches are asymmetric, as they put different restrictions on timed automata modelling specifications and implementations. This makes it impractical or impossible to have successive refinements; in particular, one of the traits of CSP is the fact that specifications are written in the same language as implementations, and, since they are at the same level, implementations can become specifications of more refined implementations, and so on. Finally, some of the approaches above are not decidable: for instance, digitisation is a decidable technique, but it is undecidable when it is possible to apply it. For these reasons, in the next chapters, we will try to define a new semantic model for our process algebra; as we will see, we will avoid the problems described above, albeit under certain restrictions.

### 3.1.2 Extensions to Timed Automata

Timed automata were the basis for several variants that have been introduced, which usually modify the way that clocks are reset or the nature of clocks constraints. It is known, for instance, that one can extend the possible clock constraints by allowing the comparison of the difference of two clocks against a constant, that is, constraints of the type $x - y \sim c$, without changing the properties of the resulting class of timed automata [AD94]. Such clock constraints are called *triangular*, as opposed to the regular *rectangular* constraints. On the other hand, if we allow constraints of the type $x + y \sim c$, checking emptiness becomes undecidable and the model is more expressive [AD94], but it becomes decidable for two clocks, undecidable for four clocks and open for three [BD00]. If we allow for silent transitions in timed automata, the model becomes more expressive, contrary to what happens for standard untimed automata [BDGP98]. Parametric timed automata

have also been considered [HRSV02]: in this model, clocks can be compared against parameters, rather than just integer values. A model checking algorithm is also proposed for this extended model. Updatable timed automata [BDFP00, BDFP04] allow clocks to be reset to any integer value or to a non-deterministic value greater than a constant. These changes alter the expressive power of the model, and, in particular, they lead to decidability for the emptiness problem only with rectangular constraints and deterministic updates. This contrasts with the equivalent result for standard timed automata, for which the type of constraint does not change the emptiness result.

### 3.1.3 Model Checking Timed Automata

Several model checking algorithms have been developed for timed automata; in particular, there are model checking techniques to verify properties expressed in the logic TCTL [ACD93], which is an extension of CTL [CE81] with a timed version of the until operator that specifies timed bounds.

The starting point for any algorithm is the region graph. Unfortunately, as we know, the size of the region graph grows exponentially, and it is therefore unsuitable for direct verification. For this reason, more efficient symbolic techniques have been developed.

Most model checking algorithms use symbolic techniques that avoid computing the whole region graph. One of the first approaches is that of [HNSY92] that collapses sets of regions by representing them as conjunctions of clock constraints, called *zones*, and identifying a predecessor relation to traverse the state space backwards. Alternatively, forward explorations have been proposed [DOTY96]. These works are the basis for feasible model checking of timed automata, and several improvements on them have been proposed, also using efficient data structures. A review of model checking for timed automata can be found in [Yov96].

Several verification tools have been developed, most notably Uppaal [LPY97, ABB+01], jointly developed by the universities of Aalborg and Uppsala, KRONOS [Yov97], developed at Verimag, and TIMES [AFM+02], developed by Uppsala University.

## 3.2 Timed Process Algebras

Most of the traditional process algebras have been extended with real-time constructs, for example, there are timed extensions of CCS [MT90, Yi90] and CSP [RR88]. Extending a process algebra with real time constructs presents design choices that are similar to those described above for operational models, like the choice between discrete and continuous time representation. An introduction to the issues concerning timed process algebras can be found in [NS92].

More recently, there have been proposals aimed at extending process algebras to model timed automata. Such process algebras, rather than annotate action transitions with time delays, introduce clocks as extra syntactic objects, and the evolution of a term depends on the values of the clocks. [YPD94] proposes an extension of CCS with constructs inspired by timed automata (without invariants), which also serve as the semantic model for the algebra; in this model, transitions are labelled by actions, guards and clock resets, and a parallel operator is considered too. This algebra serves as a descriptive language for timed automata. Another process algebra handling clocks is proposed in [LV96], where a language equivalent to timed automata with respect to trace equivalence is proposed. A comprehensive approach can be found in [DB96], where a process algebra modelling timed automata is proposed by adding separate operators for action, guards, clock resets and

location invariants; the regular part of this process algebra is axiomatised with respect to timed bisimulation [D'A97].

**Timed CSP.** The most successful timed extension to CSP is probably Timed CSP [RR88, DJR+92]. Timed CSP extends CSP by the addition of three constructs to deal with time: delay, denoting the idling of the system for a certain amount of time; timed prefixing, denoting the delay associated with each action transition; and timeout, denoting the timed choice between two processes. Timed CSP therefore handles time by specifying delays explicitly, unlike timed automata. Many denotational semantic models have been proposed for CSP [Ree88], while an equivalent operational model can be found in [Sch95]. Most works concentrated on the timed failures model. In this case a process is described in terms of its timed failures: pairs of the type $(t, F)$, where $t$ is a timed trace, that is, a sequence of actions together with the time at which they occurred, and $F$ is a timed refusal, that is, the set of actions that can be refused after the trace $t$, together with the intervals at which they can be refused. Since the semantic domain of Timed CSP is continuous, there are no general algorithms for the verification of the refinement relations, thus limiting the usefulness of this language, especially if compared to standard CSP.

Automatic verification can be achieved under some limitations, and there are two main approaches to this problem, *timewise refinement* [Sch97, Sch99] and *digitisation* [Oua01]. The idea behind timewise refinement is to ignore time in order to establish only the functional properties of a Timed CSP process. This is done by considering an untimed CSP process as specification and a Timed CSP process as implementation. It is possible to automatically verify whether the functional behaviour of the implementation refines the specification. This approach is limited as no timed properties can be verified; moreover, it is not possible to have chains of refinements in the same model, as the implementation cannot be used as a specification for further refinements.

More interesting and promising is the work on digitisation: this extends the approach described in the previous section to Timed CSP and identifies the conditions under which refinement under discrete-time semantics is equivalent to refinement in the continuous-time semantics.. In this context, it is possible to verify refinements relations with the model checker FDR2. However, some limitations still apply. In particular, it is in general undecidable to know whether digitisation techniques can be applied and chains of refinements are not possible.

Some work has been done to relate Timed CSP to timed automata: [Jac92] shows how to translate Timed CSP processes into timed automata in order to use techniques to verify properties of processes expressed in the logic TCTL. More recently, equivalence between Timed CSP (with a slightly different semantic model) and closed timed automata has been proved [OW02]; this paper also showed how to extend digitisation techniques to timed automata in order to use FDR2 to verify refinement of timed traces.

To our knowledge, no attempt to extend CSP to model timed automata directly has been made. This is the approach that we will take in this part of the thesis, trying to overcome the undecidability results for the language inclusion problem by exploiting the positive results about timed automata.

## 3.3 Our Aims and Contribution

We aim to extend CSP to model timed automata directly. Because of the undecidability results for language inclusion for timed automata due to the continuous representation of time, we take a different approach and employ the fundamental technique of timed

automata, that is, the region construction, to discretise the infinite state space caused by the representation of time. Thus, we propose a linear-time semantics based on traces that relies on the region automata model, and extend the equivalence and refinement relations in the style of CSP. We would also like the semantic model to be compositional with respect to the standard CSP constructs. As we will see, these aims are sometimes in conflict with each other and we will have to place restrictions on the model in order to achieve them.

The main contribution of this part of the thesis is the proposal of a new timed process algebra extending CSP, modelled on timed automata and with a semantic model based on regions. As a a descriptive language for timed automata, this process algebra largely adopts constructs present in other existing process algebras for timed automata [YPD94, LV96, D'A99], with the addition of those constructs that are particular to CSP.

We define new equivalence and refinement relations based on the region model for the terms of the algebra, by considering both traces and failures. This defines the theoretical foundation for a new approach to refinement in a timed setting. We also show that the relations we define are decidable for processes with a finite state representations. Although more research is needed to achieve a fully automatic procedure to verify timed properties through refinement, we highlight possible directions for future work.

# Chapter 4

# A Real-Time Process Algebra

## 4.1 Introduction

In this chapter, we introduce the real-time process algebra called Clocked CSP. It extends a variant of traditional CSP to deal with real-time by the addition of clocks and constructs that manipulate them in the style of timed automata. Clocked CSP is given a straightforward operational semantics in terms of timed automata. However, having in mind the objective of a compositional denotational semantics, we identify necessary restrictions on the language terms. Most of these restrictions are aimed at handling clocks: since the evolution of a term depends on the values of clocks, we need to control the way in which processes manipulate clocks so that side effects that would compromise compositionality are avoided. The main restrictions involve external choice, for which we give an alternative and simpler definition, and the parallel operator, which is restricted to apply only to components that do not share clocks. These two constructs cause problems since they are the only ones that allow components to perform actions independently from the other components; therefore, a component could change the values of clocks, thus modifying the behaviour of other components that reference these clocks. The problems arise due to clocks acting as shared variables for independent processes.

In this and the next chapter, we present the theoretical foundations of our timed extension of CSP, by giving it a formal syntax and operational semantics. We also show its relationship with timed automata and study its expressive power.

#### Chapter Outline

This chapter is structured in the following way. In Section 4.2 we give the definitions of timed automata needed to define Clocked CSP. In Section 4.3 we introduce the extended language, Clocked CSP, and in Section 4.4 we give it an operational semantics.

## 4.2 Preliminaries: Timed Automata

In this section, we formally introduce timed automata and we give the notions that we will use in the remainder of the chapter. Most of this section is standard in the literature (see [AD94]), and we highlight the changes we make. The importance of timed automata has already been discussed in Chapter 3.

### 4.2.1 Timed Automata

A timed automaton is a labelled transition system augmented with clocks. Clocks are real valued variables whose values control the executions of a timed automaton in two ways. As *invariants*, they control how long the system can idle in one state (or location, in the terminology proper of timed automata); as *guards* on transitions, they allow or disallow transitions depending on their current values.

Formally, let $\mathcal{C}$ be a set of clocks; we let clocks be ranged over by $x, y, \cdots$ and sets of clocks by $X, Y, \cdots$.

Given a set of clocks $\mathcal{C}$, the set $\mathcal{B}(\mathcal{C})$ of *clock constraints* is generated by the grammar:

$$\phi ::= x \prec c \mid \phi \wedge \phi \mid \neg\phi \tag{4.1}$$

for $x \in \mathcal{C}$, $\prec \in \{<, \leq\}$ and $c \in \mathbb{N}$. We denote the set of clocks that appear in a clock constraint $\phi$ as $clocks(\phi)$.

**Definition 4.1.** Given a set of clocks $\mathcal{C}$, a *valuation* $\nu$ is a function $\nu : \mathcal{C} \to \mathbb{R}^{\geq 0}$ that assigns a non-negative real value to each clock.

Given a valuation $\nu$ and a non-negative real value $d$, the valuation $\nu + d$ is defined as $(\nu + d)(x) = \nu(x) + d$ for all clocks $x \in \mathcal{C}$. Given a valuation $\nu$ and a set of clocks $X \subseteq \mathcal{C}$, $\nu[X]$ is a new valuation that agrees with $\nu$ on all clocks, except for those in $X$, whose value has been set to 0; formally, $\nu[X](x) = 0$ if $x \in X$, $\nu[X](x) = \nu(x)$ otherwise.

A clock valuation $\nu$ satisfies a constraint $\phi$ ($\nu \models \phi$) if $\phi$ evaluates to true when clocks are replaced by their valuation under $\nu$. We say that a constraint $\phi$ is *past closed* if, for all valuations $\nu$ and positive reals $d$, if $\nu + d \models \phi$ then $\nu \models \phi$, and denote the set of past closed constraints by $\mathcal{B}_c(\mathcal{C})$; informally, past closed constraints denote conditions that only place constraints on the upper limits of the values of clocks.

**Definition 4.2 (Timed Automata).** A timed automaton $\mathcal{A}$ is a tuple

$$\left(L, \overline{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow\right)$$

where:

- $L$ is the set of locations,
- $\overline{l}$ is the initial location,
- $\Sigma$ is the set of actions (or alphabet),
- $\mathcal{C}$ is the set of clocks,
- $\mathcal{I} : L \to \mathcal{B}_c(\mathcal{C})$ is the location invariant function,
- $\kappa : L \to 2^{\mathcal{C}}$ is the set of clock resets, and
- $\rightarrow \subseteq L \times (\Sigma \cup \{\tau\}) \times \mathcal{B}(\mathcal{C}) \times L$ is the set of edges (or transitions). We write $l \xrightarrow{a,\varphi} l'$ whenever $(l, a, \varphi, l') \in \rightarrow$; we call $\varphi$ the *guard* of this edge.

Informally, the execution of a timed automaton $\mathcal{A}$ proceeds as follows. Initially the control is in the initial location $l$, all clocks are set to zero and their value starts increasing at the same rate as real time. At each point, if the timed automaton is at location $l$, then the control can idle in this location if the invariant of $l$ is true under the current values of the clocks; if it is false then the system is forced to execute an action. While the control is at $l$, the system can non-deterministically choose to execute a transition $l \xrightarrow{a,\varphi} l'$ if the guard $\varphi$ is satisfied by the current values of the clocks and the control moves to $l'$. If both the invariant of a location is false and no guard is satisfied, then the system deadlocks.

| | |
|---|---|
| Action | $\dfrac{l \xrightarrow{a,\phi} l' \quad \nu \models \phi}{(free(l),\nu) \xrightarrow{a}_{lts} (l',\nu)}$ |
| Reset | $\dfrac{l \in L}{(l,\nu) \xrightarrow{\kappa(l)}_{lts} (free(l),\nu[\kappa(l)])}$ |
| Delay | $\dfrac{\forall d' \leq d \quad \nu + d' \models \mathcal{I}(l)}{(free(l),\nu) \xrightarrow{d}_{lts} (free(l),\nu + d)}$ |

Table 4.1: Transition relation of the lts associated to a timed automaton.

**Example 4.1.** Figure 4.1 shows a simple timed automaton with two clocks $x$ and $y$. Locations are represented by nodes and labelled by the clocks that are reset upon entering each location and by the location invariants. Edges represent transitions and are labelled by an action and by the corresponding guard.

### 4.2.2 Semantics of Timed Automata

A timed automaton is given semantics in terms of a labelled transition system. At each point of the computation one must know the location the system is in and the current value of clocks. The state space of the transition system is thus given by the cross product of locations and clock valuations. The semantics of a timed automaton is given by the lts $LTS_{\mathcal{A}} = (Q, \overline{q}, \Sigma \cup \mathbb{R}^{\geq 0} \cup 2^{\mathcal{C}}, \rightarrow_{lts})$, defined as follows:

- $Q$ is the set of states. A state is either a pair $(l,\nu)$ or $(free(l),\nu)$, where $l \in L$ and $\nu$ is a clock valuation. $(free(l),\nu)$ denotes the state in the transition system corresponding to location $l$, but after the clocks in $\kappa(l)$ have been reset. On the other hand, location $(l,\nu)$ denotes the set of states before the set of clocks $\kappa(l)$ has been reset.

- the initial state is $\overline{q} = (\overline{l}, \nu_0)$, where $\nu_0(x) = 0$ for all $x \in \mathcal{C}$.

- $\rightarrow_{lts} \subseteq Q \times ((\Sigma \cup \{\tau\}) \cup \mathbb{R}^{\geq 0} \cup 2^{\mathcal{C}}) \times Q$ is the set of transitions defined by the rules of Table 4.1.

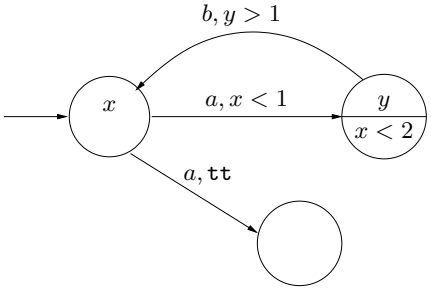



Figure 4.1: A timed automaton.

The definition we have given is different from the one usually in the literature since we consider clock resets as visible actions. Any subset of clocks $X \subseteq 2^{\mathcal{C}}$ can be reset, and this is visible in the lts. Upon entering a location $l$, the action of resetting the (possibly empty) set of clocks $\kappa(l)$ is instantaneously executed, leading to the new clock valuation $\nu[\kappa(l)]$. The reason for this will become clear later on when we give semantics to our language which will make clock resets visible; this makes no difference in our case because we do not use relations based on the labelled transition systems (e.g., timed bisimulation). It is worth pointing out that, even with this new semantics, the undecidability result for language inclusion still holds, as it is easy to observe that reset actions introduce no branching and therefore they do not add any nondeterminism; it suffices to ignore such actions and we obtain the same traces as with the usual semantics.

Note, in particular, that new locations are entered only after an action in $\Sigma$, and that after entering a new location a reset action is performed: therefore each action in $\Sigma$ is immediately followed by a (possibly empty) clock reset. We can easily define the equivalent transition system without reset action by collapsing each action transition $(l, \nu) \xrightarrow{a} (l', \nu)$ and the following reset transition $(l', \nu) \xrightarrow{\kappa(l')} (free(l'), \nu[\kappa(l')])$ into a single transition $(l, \nu) \xrightarrow{a} (free(l'), \nu[\kappa(l')])$. We define this new transition system as $LTS'_{\mathcal{A}}$, with alphabet $\Sigma \cup \mathcal{R}^{\geq 0}$. This corresponds to the labelled transition system semantics usually found in the literature [Plo81].

### 4.2.3 Region Automata

The transition system defined above has an infinite (continuous) set of states and actions. In order to model check timed automata, we discretise such state space into equivalence classes that relate clock valuations that agree on the integral part of clocks and on the ordering of their fractional part. Let $c_x$ be the greatest constant against which clock $x$ is compared, and, given a real number $c$, $\lfloor c \rfloor$ the integral part of $x$ and $fr(c)$ its fractional part. Given a set of clocks $\mathcal{C}$, two valuations $\nu$ and $\nu'$ are equivalent ($\nu \equiv_{\mathcal{C}} \nu'$) if all of the following conditions hold:

- for all $x \in \mathcal{C}$, $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or they both exceed $c_x$;
- for all $x, y \in \mathcal{C}$, with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fr(\nu(x)) \leq fr(\nu(y))$ iff $fr(\nu'(x)) \leq fr(\nu'(y))$;
- for all $x \in \mathcal{C}$ with $\nu(x) \leq c_x$, $fr(\nu(x)) = 0$ iff $fr(\nu'(x)) = 0$.

A *clock region* is an equivalence class induced by $\equiv_{\mathcal{C}}$, and the *region graph* can be thought of as the equivalence classes together with the transitions between the classes. Since all valuations in the same region agree on the integral parts of the clocks, it is clear that the same set of action transitions can be enabled from all states within a region. We denote the set of regions associated to a timed automaton $\mathcal{A}$ by $R_{\mathcal{A}}$, and we let $r, r_1, r_2 \ldots$ range over regions. If $\mathcal{A}$ is clear from the context, it will be elided and we will denote the set of regions as $R$. We often denote a region by the set of clock constraints that are met by the valuations in the region only. We denote by $r_0$ the starting region (all clocks set to 0) and by $r_{max}$ the region for which $x > c_x$ for all clocks $x$. We say that a region $r$ satisfies a condition on clocks $\phi \in \mathcal{B}(\mathcal{C})$ if the condition evaluates to true under all the valuations in $r$; in this case we write $r \models \phi$.

With passage of time, the automaton changes region. We define the *successor* region as the next region that the automaton will move to by letting time elapse. Formally, we define a function $succ : R \longrightarrow R$ such that $succ(r) = r'$ if for all $\nu \in r$ there exists $d \in \mathbb{R}^{\geq 0}$

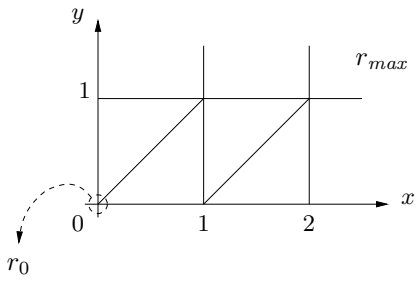



Figure 4.2: A region graph.

such that $\nu + d \in r'$ and for all $d' < d$ either $\nu + d' \in r$ or $\nu + d' \in r'$. Note that *succ* is undefined for $r_{max}$.

The action of moving to the next region involves an increment of the value of all clocks, but only some of them actually cause the change of a region. For example, if we consider two clocks $x$ and $y$, when going from the region $x = y = 0$ to the region $0 < x = y < 1$, both clocks change region. However when going from $(0 < x < 1) \wedge (y = 0)$ to $(0 < x < 1) \wedge (0 < y < 1) \wedge (y < x)$, it is only $y$ that changes region. We are interested in identifying the set of clocks that change their own region as it will be convenient in the following. We define $ch : R \longrightarrow 2^{\mathcal{C}}$ as the set of clocks that change their own region at the next *succ* action; $ch(r)$ is the smallest set $X$ of clocks such that, for all valuations $\nu \in r$ and $\eta \in succ(r)$, we have $\nu \equiv_{\mathcal{C} \setminus X} \eta$. We also define $\Delta$ as the set of actions describing the change of region due to the passage of time; the elements of $\Delta$ are $\delta_X$, where $X \in 2^{\mathcal{C}}$.

**Example 4.2 (Region Graph).** Figure 4.2 shows the region graph for two clocks $x$ and $y$, with $c_x = 2$ and $c_y = 1$. Each area, line or intersection point represent a different region.

We can now define the region automaton corresponding to a timed automaton $\mathcal{A}$.

**Definition 4.3 (Region Automata).** Given a timed automaton $\mathcal{A}$, the corresponding **region automaton** is a labelled transition system $\mathcal{R}(\mathcal{A}) = (Q_r, \Sigma_r, \overline{q}_r, \rightarrow_r)$ defined as follows:

- $Q_r = \{(l, r) \mid l \in L \cup free(L),\ r \in R_{\mathcal{A}}\}$
- $\Sigma_r = \Sigma \cup \Delta \cup 2^{\mathcal{C}}$, with $\Delta = \{\delta_X | X \subseteq \mathcal{C}\}$
- $\overline{q}_r = (\overline{l}, r_0)$
- $\rightarrow_r \subseteq Q_r \times (\Sigma_r \cup \{\tau\}) \times Q_r$ such that:
  - $(l, r) \xrightarrow{a}_r (l', r)$, where $a \in \Sigma$ if, for all $\nu \in r$, $(l, \nu) \xrightarrow{a}_{lts} (l', \nu)$;
  - $(l, r) \xrightarrow{X}_r (l, r[X])$, where $X \subseteq \mathcal{C}$, if, for all $\nu \in r$, $(l, \nu) \xrightarrow{X}_{lts} (l, \nu[X])$;
  - $(l, r) \xrightarrow{\delta_X}_r (l, succ(r))$ if, for all $\nu \in r$, there exists $d$ such that $(l, \nu) \xrightarrow{d}_{lts} (l, \nu')$ with $\nu' \in succ(r)$ and $X = ch(r)$.

It is easy to observe that a transition $(l, r) \xrightarrow{a}_r (l', r)$ is enabled when there is an edge $(a, \phi)$ leaving location $l$ such that $r \models \phi$. Likewise, a transition $(l, r) \xrightarrow{\delta_X}_r (l, succ(r))$ is enabled if it is possible to let time elapse in the location $l$, that is, $succ(r) \models \mathcal{I}(l)$. Finally, a clock reset action is enabled upon entering a location.

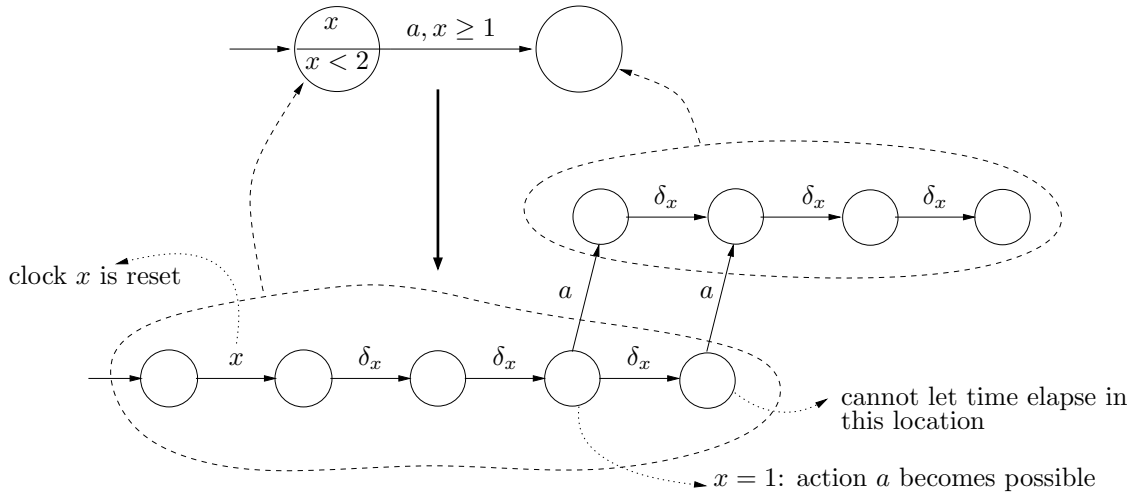



Figure 4.3: A timed automaton (top) $\mathcal{A}$ and the corresponding region automaton $R(\mathcal{A})$ (bottom).

**Example 4.3 (A region automaton).** Figure 4.3 shows a small timed automaton and the corresponding region automaton constructed according to our definitions. The states of the $R(\mathcal{A})$ in the bottom row correspond to the first location of $\mathcal{A}$, while those in the top row correspond to the second location. The first action of $R(\mathcal{A})$ is a reset action, corresponding to the reset of $x$ in the initial location of $\mathcal{A}$. After such action, the value of $x$ is 0, and, with each $\delta$ action, the value of $x$ increases in order to reach the next region; so, in the next state we have $0 < x < 1$, $x = 1$ in the following one, and so on. Before action $a$ is enabled, at least two delay actions must be executed, reaching the condition $x \geq 1$ of the guarded action of $\mathcal{A}$. The two (maximal) traces of $R(\mathcal{A})$ are the following:

$$\langle \{x\}\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x\, \delta_x \rangle$$
$$\langle \{x\}\, \delta_x\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x \rangle$$

Together, they carry the information that $a$ was possible only while $1 \leq x < 2$.

Region automata are the basis for any algorithm to model check timed automata, and they are an important technique to discretise the infinite state space of the induced transition system. Complexity is their main drawback, as the number of regions is exponential in the number of clocks and in the magnitude of the maximal constants. For this reason, more efficient representations have been devised (e.g. zones, see [Yov96] for an introduction).

### 4.2.4 Timed Bisimulation

It is possible to define a notion of bisimulation for timed automata that takes into account the passage of time. The resulting relation is called *timed bisimulation* and is easily defined by employing the standard notion of bisimulation (see Definition 2.3) on the labelled transition system corresponding to a timed automaton.

**Definition 4.4 (Timed Bisimulation).** Given two timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are timed bisimilar if there exists a bisimulation between $LTS'_{\mathcal{A}_1}$ and $LTS'_{\mathcal{A}_2}$.

It is quite surprising that timed bisimulation, despite being defined on an infinite transition system, is decidable [Čer93]; the algorithm uses the region automaton construction to achieve this result. The notion of timed simulation is defined similarly, and it is also decidable. Timed bisimulation is often considered the preferred equivalence relation for timed automata, as it considers both the timing and the branching aspects of a system.

## 4.3 Clocked CSP

### 4.3.1 Aims

When defining a timed process algebra, we are guided by the desire to combine the features of timed automata and those of CSP. The aim is to *extend CSP* with timing constructs to obtain a calculus that is *inspired* by timed automata and that describes them. We want to define a discrete *denotational semantics* on the language that extends CSP semantics and that is compositional, leading to *decidable refinement relations* that can be checked by using FDR2, and also ensuring that many algebraic properties of CSP can be inherited. We also often directly "import" results of CSP whenever we can find a direct correspondence between our algebra and CSP. In the next sections, we describe how we achieved this, justifying the restrictions that we had to impose on the language.

### 4.3.2 The Language

$$
\begin{array}{lcl}
P & ::= & \{\!|X|\!\}P \mid \phi \triangleright P \mid T \\
\\
T & ::= & STOP \mid SKIP \mid \Box_{i=1}^{N}(a_i, \varphi_i) \to P_i \mid \sqcap_{i=1}^{N} P_i \mid \\
 & & P \underset{A}{||} P \mid Z \mid \mathsf{rec}\, Z.P \mid P \setminus A \mid f[P] \mid P; P
\end{array}
$$

Table 4.2: Syntax of Clocked CSP

We define a language for describing timed automata, called Clocked CSP (CCSP), as an extension of CSP, thus retaining its choice operators, the hiding operator and the multi-way nature of the parallel composition. Clocked CSP terms with alphabet $\Sigma$ and set of clocks $\mathcal{C}$ are obtained from the syntax of Table 4.2, where $a \in \Sigma$, $A \subseteq \Sigma$, $\phi \in \mathcal{B}_c(\mathcal{C})$, $\varphi \in \mathcal{B}(\mathcal{C})$, $X \subseteq \mathcal{C}$, and $f : \Sigma \to \Sigma$. Here, we use the notation of [DB96] to denote clock resets and location invariants. By convention, visible actions will be ranged over by $a, b, \ldots$, generic actions (including $\tau$) by $\mu$, clocks by $x, y, \ldots$, sets of clocks by $X, Y, \ldots$ and clock constraints by $\phi, \gamma, \cdots$. We denote the set of processes generated by the above grammar by **CCSP**. For simplicity, we write $\{\!|x, y \ldots|\!\}$ instead of $\{\!|\{x, y \ldots\}|\!\}$ when a list of clocks is given explicitly; we also omit the interface alphabet of parallel composition when implicit or not relevant. The syntax above is divided into two parts: the first part (the $P$ terms) introduces the new constructs for handling clocks, that is, clock resets and invariants. In the second part, we find the usual constructs of CSP, with some changes regarding external choice. We use the following abbreviations: $(a, \phi) \to P$ (guarded action) for external choice with $N = 1$, and $|||$ for parallel composition with empty interface alphabet, that is, $\underset{\emptyset}{||}$.

Let us consider each construct in detail:

- *deadlock*: $STOP$ represents the deadlocked process, that can let time elapse, but cannot synchronise on any action.
- *successful termination*: $SKIP$ represents successful termination; $SKIP$ can let time elapse before terminating.
- *reset*: $\{\!|X|\!\}P$ resets the clocks in $X$ instantaneously and then behaves as $P$.
- *invariant*: $\phi \triangleright P$ must evolve (i.e. $P$ must perform an action) while $\phi$ is satisfied. This corresponds to invariants in timed automata.
- *external choice*: $\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i$ can execute any $a_i$ action while $\varphi_i$ is satisfied. Time can elapse before the choice is resolved.
- *internal choice*: $\sqcap_{i=1}^{N} P_i$ non-deterministically chooses one $P_i$ and performs it instantaneously.
- *interface parallel*: $P_1 \underset{A}{||} P_2$ represents the parallel composition of $P_1$ and $P_2$ on the interface alphabet $A$: the two processes must synchronise on all the actions in $A$, while they execute all other actions independently. $P_1$ and $P_2$ always agree on the current value of clocks.
- *process name:* $Z$ represents the process name $Z$, which is to be bound by a recursive definition.
- *recursion*: $\mathsf{rec}\, Z.P$ defines recursive behaviour, binding the process variable $Z$ whenever it occurs in $P$.
- *hiding:* $P \setminus A$ behaves as $P$, but all actions in $A$ are hidden, that is, they become internal actions $\tau$.
- *renaming:* $f[P]$ behaves like $P$, with actions renamed according to $f$.
- *sequential composition:* $P_1; P_2$ behaves first like $P_1$ and then, if $P_1$ terminates successfully, as $P_2$.

### 4.3.3 Design Choices

In the design of our process algebra, we have made a few design choices that differentiate CCSP from the traditional treatment of standard CSP. One of the main issues we had to address was how to treat clocks, which can be seen as a particular kind of shared variables.

Recall that, in standard CSP, external choice is defined in an unguarded way as $P_1 \,\Box\, P_2$ (see Section 2.2). If we used this definition, the interaction between the presence of clocks and external choice would create problems since one of the two component processes is allowed to make internal moves before the choice is resolved, and an internal move could cause the reset of some clocks. By changing the value of some clocks, a process would influence the behaviour of the other component; this would in turn create problems for compositional semantics. Let us show this by means of an example: consider the process $P \,\Box\, (Q \setminus \{b\})$, where:

$$\begin{aligned} P &= \{\!|x|\!\}(a, x > 1) \rightarrow SKIP \\ Q &= (b, \mathtt{tt}) \rightarrow \{\!|x|\!\}(a, \mathtt{tt}) \rightarrow SKIP \end{aligned}$$

$$\mathcal{C}_i(STOP) = \emptyset \qquad \mathcal{C}_i(SKIP) = \emptyset$$

$$\mathcal{C}_i(\{\!|X|\!\}P) = X \cup \mathcal{C}_i(P) \qquad \mathcal{C}_i(\phi \triangleright P) = clocks(\phi) \cup \mathcal{C}_i(P)$$

$$\mathcal{C}_i(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i) = \bigcup_{i=1}^{N}(clocks(\varphi_i) \cup \mathcal{C}_i(P_i))$$

$$\mathcal{C}_i(\sqcap_{i=1}^{N} P_i) = \bigcup_{i=1}^{N} \mathcal{C}_i(P_i) \qquad \mathcal{C}_i(P_1 \underset{A}{||} P_2) = \mathcal{C}_i(P_1) \cup \mathcal{C}_i(P_2)$$

$$\mathcal{C}_i(\mathsf{rec}\, Z.P) = \mathcal{C}_i(P)$$

$$\mathcal{C}_i(P \setminus A) = \mathcal{C}_i(P) \qquad \mathcal{C}_i(P_1; P_2) = \mathcal{C}_i(P_1) \cup \mathcal{C}_i(P_2)$$

Table 4.3: Rules for internal clocks.

It is possible for process $Q$ to reset the clock $x$ after $P$ has reset it and begun waiting for the guard $x > 1$ to become true. So, the behaviour of $P$ is influenced by $Q$'s internal actions, and it is not possible to define $P$'s semantics without knowing the context, placing some restrictions on the processes or modifying the semantics. A possible solution would have been to impose disjoint sets of clocks on the components (as we do for the parallel operator, see below), but this would rule out conditions on actions over the same clocks. For instance, it would be impossible to have common conditions, such as the following: do an $a$ action if $x < 1$ and a $b$ action otherwise. It would be possible to impose that any clock reset resolves the choice, as we did in [CK03]; this would lead to a compositional semantics, but is counter-intuitive and the resulting semantics would be quite involved.

As a solution to this problem we have chosen to only allow guarded external choice. This is a significant restriction to the classical CSP way to model external choice, but we have chosen to adopt it because it gives a clean, compositional semantics, and also because we think that this definition respects our intuition of what we want to model with external choice, especially in a setting with clocks that function as shared variables.

Similar problems hold in the case of the parallel operator, as also in this case a component can move independently from the other. We have adopted a different solution in this case, as explained in the following section.

### 4.3.4 Internal and External Clocks

External choice is not the only operator whose definition is made hard by the presence of clocks: the same difficulties arise for parallel composition if we want to obtain compositionality. Since the interaction between processes that modify the value of clocks is problematic, we make the following simplification: each process can handle only a subset of clocks, so that its behaviour depends only on them. The remaining clocks can be used by other processes that interact with it through the parallel operator.

More formally, given the global set of clocks $\mathcal{C}$, we define the internal set of clocks $\mathcal{C}_i(P)$ for a process $P$ as the set of clocks that are explicitly referred to within $P$ either inside clock resets or invariants. $\mathcal{C}_e(P)$ is defined as the complement $\mathcal{C} \setminus \mathcal{C}_i(P)$. Table 4.3 shows the inductive definition of $\mathcal{C}_i(P)$. We restrict the parallel operator to work only with pairs of processes $P_1$ and $P_2$ with disjoint sets of internal clocks, that is, such that $\mathcal{C}_i(P_1) \cap$

| Location Invariants | | |
|---|---|---|
| $\mathcal{I}(STOP) = \mathtt{tt}$ | $\mathcal{I}(SKIP) = \mathtt{tt}$ | $\mathcal{I}(\varphi \triangleright P) = \varphi \wedge \mathcal{I}(P)$ |
| $\mathcal{I}(\{\!\|X\|\!\}P) = \mathcal{I}(P)$ | $\mathcal{I}(\mathsf{free}(P)) = \mathcal{I}(P)$ | $\mathcal{I}(\mathsf{rec}\, Z.P) = \mathcal{I}(P)$ |
| $\mathcal{I}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i\right) = \mathtt{tt}$ | $\mathcal{I}\left(\sqcap_{i=1}^{N} P_i\right) = \mathtt{ff}$ | $\mathcal{I}\left(P \underset{A}{\|\|} Q\right) = \mathcal{I}(P) \wedge \mathcal{I}(Q)$ |
| $\mathcal{I}(P \setminus A) = \mathcal{I}(P)$ | $\mathcal{I}(f[P]) = \mathcal{I}(P)$ | $\mathcal{I}(P;Q) = \mathcal{I}(P)$ |

Table 4.4: Rules for invariants

$\mathcal{C}_i(P_2) = \emptyset$. The internal clocks of the parallel composition are defined as $\mathcal{C}_i(P_1 \,||\, P_2) = \mathcal{C}_i(P_1) \cup \mathcal{C}_i(P_2)$. When the process $P$ is implicit from the context, we denote the set of its internal and external clocks simply by $\mathcal{C}_i$ and $\mathcal{C}_e$, respectively, instead of $\mathcal{C}_i(P)$ and $\mathcal{C}_e(P)$.

The idea behind this restriction is that, when defining the semantics of a process, we have to assume that, while the process has full control of its internal clocks, any action on external clocks is possible at any time, caused by any process running in parallel. A process must be willing to synchronise on any possible set of (external) clock resets at any moment. This is why we treat clock resets as visible actions: when a process resets a set of clocks $X$, the parallel processes are willing to synchronise on this reset action. In this way processes always agree on the value of clocks. We believe this is a reasonable restriction since most parallel composition constructs for timed automata use disjoint sets of clocks.

A similar solution was adopted in [Bro93], where a compositional semantics is given for a language with parallelism and shared variables. In this case, the semantic model assumes that each step of the computation of a process can be interleaved by an arbitrary finite number of steps by other processes running in parallel that modify the values of the shared variables.

## 4.4 Operational Semantics

Clocked CSP terms are assigned operational semantics in terms of timed automata in a natural way, as we have designed the language to model them.

For the purpose of giving semantics to Clocked CSP, we introduce an extra operator, $\mathsf{free}(P)$, representing a process that behaves exactly like $P$ but which does not perform any initial reset (i.e. the start state has been stripped of its resets). This performs a similar function to the operator $free$ introduced to give semantics to timed automata. We denote the set of all CCSP processes extended with the $\mathsf{free}(\bullet)$ operator by $\mathbf{CCSP}^+$. Given a CCSP term $P$, we define the corresponding timed automaton $\mathcal{A}(P) = (L, \overline{l}, \Sigma \cup \{\tau\} \cup \{\checkmark\}, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, where $L = \mathbf{CCSP}^+$, $\overline{l} = P$, the sets of clocks and actions are the same and $\rightarrow$, $\kappa$ and $\mathcal{I}$ are defined according to the rules of Table 4.6, Table 4.4 and Table 4.5, respectively. We have already introduced the silent action $\tau$, which is the result of some internal computation, usually caused by internal choice or hiding. Following the CSP convention, we use an additional special label $\checkmark$, with a true guard, to denote successful termination and an additional process $\Omega$ that denotes the process that has successfully terminated.

| Clock Resets | | |
|---|---|---|
| $\kappa(STOP) = \emptyset$ | $\kappa(SKIP) = \emptyset$ | $\kappa(\phi \triangleright P) = \kappa(P)$ |
| $\kappa(\{\!\lvert X \rvert\!\}P) = X \cup \kappa(P)$ | $\kappa(\mathsf{free}(P)) = \emptyset$ | $\kappa(\mathsf{rec}\, Z.P) = \kappa(P)$ |
| $\kappa\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i\right) = \emptyset$ | $\kappa\left(\sqcap_{i=1}^{N} P_i\right) = \emptyset$ | $\kappa\left(P \underset{A}{\lvert\lvert} Q\right) = \kappa(P) \cup \kappa(Q)$ |
| $\kappa(P \setminus A) = \kappa(P)$ | $\kappa(f[P]) = \kappa(P)$ | $\kappa(P; Q) = \kappa(P)$ |

Table 4.5: Rules for clock resets

| Transitions |
|---|
| $\dfrac{}{SKIP \xrightarrow{\checkmark, \mathtt{tt}} \Omega}$ $\qquad$ $\dfrac{P \xrightarrow{a,\varphi} P'}{\mathsf{rec}\, Z.P \xrightarrow{a,\varphi} P'[Z/\mathsf{rec}\, Z.P]}$ |
| $\dfrac{P \xrightarrow{a,\varphi} P'}{\phi \triangleright P \xrightarrow{a,\varphi} P'}$ $\qquad$ $\dfrac{P \xrightarrow{a,\varphi} P'}{\{\!\lvert X \rvert\!\}P \xrightarrow{a,\varphi} P'}$ $\qquad$ $\dfrac{P \xrightarrow{a,\varphi} P'}{\mathsf{free}(P) \xrightarrow{a,\varphi} P'}$ |
| $\dfrac{}{\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i\right) \xrightarrow{a_j, \varphi_j} P_j}\ j \in \{1..N\}$ |
| $\dfrac{}{\sqcap_{i=1}^{N} P_i \xrightarrow{\tau, \mathtt{tt}} P_j}\ j \in \{1..N\}$ $\qquad$ $\dfrac{P \xrightarrow{a,\varphi} P'}{f[P] \xrightarrow{f(a),\varphi} f[P']}$ |
| $\dfrac{P \xrightarrow{a,\varphi} P'}{P \setminus A \xrightarrow{\tau,\varphi} P' \setminus A}\ a \in A$ $\qquad$ $\dfrac{P \xrightarrow{\mu,\varphi} P'}{P \setminus A \xrightarrow{\mu,\varphi} P' \setminus A}\ \mu \notin A$ |
| $\dfrac{P \xrightarrow{\mu,\varphi} P'}{P \underset{A}{\lvert\lvert} Q \xrightarrow{\mu,\varphi} P' \underset{A}{\lvert\lvert} \mathsf{free}(Q)}\ \mu \notin A$ $\qquad$ $\dfrac{Q \xrightarrow{\mu,\varphi} Q'}{P \underset{A}{\lvert\lvert} Q \xrightarrow{\mu,\varphi} \mathsf{free}(P) \underset{A}{\lvert\lvert} Q'}\ \mu \notin A$ |
| $\dfrac{P \xrightarrow{a,\varphi_1} P' \quad Q \xrightarrow{a,\varphi_2} Q'}{P \underset{A}{\lvert\lvert} Q \xrightarrow{a,\varphi_1 \wedge \varphi_2} P' \underset{A}{\lvert\lvert} Q'}\ a \in A \cup \{\checkmark\}$ |
| $\dfrac{P \xrightarrow{\mu,\varphi} P'}{P; Q \xrightarrow{\mu,\varphi} P'; Q}\ \mu \neq \checkmark$ $\qquad$ $\dfrac{P \xrightarrow{\checkmark, \mathtt{tt}} \Omega}{P; Q \xrightarrow{\tau, \mathtt{tt}} Q}$ |

Table 4.6: Rules for transitions

The operational semantics is intuitive and it extends CSP semantics in the obvious way for most operators. One design choice that we have made was to have the invariant of

internal choice to be false; the reason for this is that internal choice is resolved immediately; in this way the semantics of internal choice follows that of internal choice in standard CSP exactly. We explain the introduction of the free(•) operator by means of the following example [D'A99].

**Example 4.4.** Consider the process $(\{\!|x|\!\}(a, \varphi_1) \rightarrow STOP) \,|||\, (\{\!|y|\!\}(b, \varphi_2) \rightarrow STOP)$. At the beginning of the execution, both components reset their clocks; when one of the components executes, we do not want the other component to reset its clock again. The operational semantics of this term is given by the automaton of Figure 4.4; without the free operator, the clocks would be reset in the middle locations as well, even if they have both been reset at the start.

The timed automaton corresponding to a CCSP term might have an infinite number of locations, thus also leading to an infinite state region automaton. For automatic verification purposes this should be avoided, and it is necessary to restrict to finite state processes, that is, processes whose operational model has a finite number of reachable states; the same restrictions as those applied to standard CSP apply in our case; see [Ros94] for a discussion of finite state processes.

**Region automaton.** In the following, we will use the region automaton obtained from the operational semantics as the model we have in mind for the denotational models for CCSP. Given a CCSP term $P$ and the associated timed automaton $\mathcal{A}(P)$, we denote the region automaton corresponding to $\mathcal{A}(P)$ with initial region $r$ by $R(P, r)$. If we want to add an extra invariant $\varphi$ to the initial location, i.e. $\mathcal{I}(P) = \mathcal{I}(P) \wedge \varphi$, we denote the resulting region automaton by $R(P, r, \varphi)$. We use $R(P)$ as an abbreviation for $R(P, r_0, \mathtt{tt})$.

### 4.4.1 Clocked CSP and Timed Automata

We analyse the expressive power of Clocked CSP and show that CCSP is a complete language for the description of timed automata, that is, every timed automaton can be described by a CCSP term.

**Theorem 4.5.** *For all (finitely branching) timed automata $\mathcal{A}$, there exists a CCSP term $P$ such that $\mathcal{A}(P)$ is isomorphic to the reachable part of $\mathcal{A}$.*

*Proof.* The proof is similar to the proof in [D'A99], but we also have to deal with internal $\tau$ actions. Given a timed automaton $\mathcal{A} = (L, \bar{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, we create a CCSP term with the same alphabet and set of clocks in the following way: for each location $l$, we define a process name $P_l$. A process is associated with each name, that is, for each $l \in L$, $P_l = \{\!|\kappa(l)|\!\}\mathcal{I}(l) \triangleright \left(\Box_{(l,a_i,\phi_i,l_i)\in\rightarrow}(a_i, \phi_i) \rightarrow P_{l_i}\right)$. This does not work if $\mathcal{A}$ contains internal actions. This problem can be solved by using an extra action $b$ in the construction above

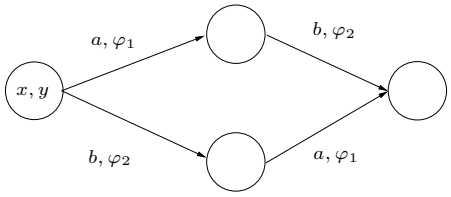



Figure 4.4: Explanation of the free(•) operator.

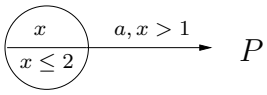



Figure 4.5: Timed automaton induced by $(x \leq 2) \triangleright \{\!|x|\!\}(a, x > 1) \rightarrow P$. Clock $x$ is bound by the initial reset.

whenever a $\tau$ is encountered. This action can then be hidden at the top level. Clearly, the function $f : L \rightarrow CCSP$ defined as $f(l) = P_l$ is an isomorphism for the reachable part of the timed automaton. □

**Example 4.5.** Consider the timed automaton $\mathcal{A}$ of Example 4.3: the CCSP term corresponding to $\mathcal{A}$ is given by $\{\!|x|\!\}(x < 2) \triangleright (a, x \geq 1) \rightarrow STOP$.

### 4.4.2 Equivalence Relations via the Operational Model

It is possible to define equivalences for CCSP processes by using the operational model: in this case two processes would be equivalent according to some relation if the corresponding timed automata are equivalent according to such relation. For example, we can define bisimulation for CCSP processes as follows.

**Definition 4.6 (Bisimulation).** Two CCSP processes $P_1$ and $P_2$ are bisimilar if there exists a timed bisimulation between $\mathcal{A}(P_1)$ and $\mathcal{A}(P_2)$, according to Definition 4.4.

Other relations can be defined, for example isomorphism or structural equivalence of the timed automata corresponding to CCSP terms. We are not interested in this approach: since we want to give Clocked CSP a semantics and equivalence/refinement relations in the style of CSP. This was the approach of [D'A99], where such equivalence relations were defined and axiomatised.

### 4.4.3 Normal Form

We need to impose several restrictions on the syntax of terms, so that we get terms whose structure makes the semantics more intuitive and which also allow for a simpler definition of the semantic rules. Consider the following term, $(x \leq 2) \triangleright \{\!|x|\!\}(a, x > 1) \rightarrow P$, and its corresponding timed automaton (Figure 4.5). From the timed automaton, we can see that the invariant $x \leq 2$ is bound by the reset of clock $x$, but this contradicts the intuition, since we use $x$ before it is reset (this corresponds to the idea of *conflict* in [D'A99]) and we would like to rewrite the term as $\{\!|x|\!\}(x \leq 2) \triangleright (a, x > 1) \rightarrow P$.

Also, consider the term, $\{\!|x, y|\!\}((x \leq 2) \triangleright P_1 \,||\, (y < 1) \triangleright P_2)$, where $x$ is a clock of the first component and $y$ is a clock of the second. Both clock $x$ and clock $y$ are immediately reset (by applying the rules of Table 4.5). In order to simplify the semantics, and also to emphasise that each component handles its own clocks, we would like to rewrite this term as: $\{\!|x|\!\}((x \leq 2) \triangleright P_1) \,||\, (\{\!|y|\!\}(y < 1) \triangleright P_2)$.

The two examples above explain our need for a *normal form* for CCSP terms by which the structure of CCSP terms reflects their semantics, and we prefix terms that can execute actions by resets and invariants all grouped together. We obtain the alternative and equivalent (up to operational semantics) syntax of Table 4.7.

A CCSP term is either a $T$ term prefixed by a (possibly empty) set of clock resets and a (possibly true) invariant, the parallel compositions of such terms without any leading clock operator, or some other term with some actions hidden or renamed.

$$
\begin{array}{lcl}
P & ::= & \{\!|X|\!\}\phi \triangleright T \mid P \underset{A}{||} P \mid P \setminus A \mid f[P] \mid P;P \mid \mathsf{rec}\, Z.P \mid Z \\
T & ::= & STOP \mid SKIP \mid \Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i \mid \sqcap_{i=1}^{N} P_i
\end{array}
$$

Table 4.7: Normal form for CCSP processes.

$$
\begin{array}{cc}
P = \{\!|\emptyset|\!\}P & P = \mathtt{tt} \triangleright P \\
\phi_1 \triangleright \phi_2 \triangleright P = (\phi_1 \wedge \phi_2) \triangleright P & \{\!|X|\!\}\{\!|Y|\!\}P = \{\!|X \cup Y|\!\}P \\
\phi \triangleright \{\!|X|\!\}P = \{\!|X|\!\}\phi \triangleright P & \phi \triangleright (P_1 \,||\, P_2) = (\phi \triangleright P_1) \,||\, (\phi \triangleright P_2)
\end{array}
$$

$$
\{\!|X_1 \cup X_2|\!\}(P_1 \,||\, P_2) = (\{\!|X_1|\!\}P_1) \,||\, (\{\!|X_2|\!\}P_2) \text{ with } X_1 \subseteq \mathcal{C}_i(P_1) \text{ and } X_2 \subseteq \mathcal{C}_i(P_2)
$$

Table 4.8: Rewrite rules to obtain CCSP terms that are equivalent up to the operational semantics, i.e., they induce the same timed automaton.

**Proposition 4.7.** *Every CCSP term can be rewritten to an equivalent (up to operational semantics) CCSP term generated by the syntax of Table 4.7.*

*Proof.* Let us use structural induction on the terms generated by the syntax of Table 4.2. Clearly, *STOP*, *SKIP* and all the $T$ terms are already compatible with the alternative syntax if their components are. We need to prove that the result is true for all the $P$ terms. This can be done by considering the sequence of resets and invariants and by applying the rules of Table 4.8 to rewrite the leftmost elements, until we are left with a unique reset operation followed by a unique invariant operation. By using induction, the result is proved if the $P$ term is followed by any $T$ term except for parallel composition. If the term is followed by parallel composition, the syntax of Table 4.7 is not respected because the parallel operator cannot be preceded by resets and invariants. This case can be solved by using the last two rules of Table 4.8 "pushing" the resets and invariants inside the parallel. □

In the following, we assume that all processes are in normal form and we rely on this fact to define the denotational semantics and to prove properties about it.

# Chapter 5

# Denotational Semantics

## 5.1 Introduction

In the previous chapter we introduced Clocked CSP, as an extension to classical CSP with clocks, and we gave it operational semantics in terms of timed automata and region automata. By working on the transition system induced by the operational model, one could apply known equivalence relations for timed automata (e.g. timed bisimulation) or use traditional model checking techniques to verify whether a given process meets some temporal logic property. Since we are following the CSP approach, we want to give a denotational semantics to the language as an extension of the usual trace/failures semantics, together with *refinement* relations. In doing so, we would like such semantics to be *decidable* (the refinement relations must be decidable) and *compositional* (the semantics of terms must be definable in terms of its components). As we will see, these two objectives are not straightforward to achieve.

The most natural model would be that of timed traces (and timed refusals), that is, recording the absolute time of execution of each action in the trace. This is the model of Timed CSP. Unfortunately, this solution conflicts with one of the two requirements that we postulated on our semantic model, namely that the resulting refinement relations are decidable, as language inclusion for timed automata is not decidable [AD94]. As discussed in Chapter 3, it is possible to put constraints on timed automata to achieve decidability, but this would cause loss of generality and we have decided to avoid this approach. As for compositionality, the difficulties are caused by clocks acting as shared variables, and we have already put appropriate restrictions on the language, explaining why the use of clocks causes interactions among components that were not present in the original CSP.

For this reason, we have to identify a suitable level of abstraction between the infinite and undecidable semantic level of the transition system and some appropriate high-level description. Our choice was to model the semantics on the region automaton. The idea is to record the clock constraints met when an action is taken, rather than the absolute time, since, in general, we might not be interested in the absolute timed behaviour of a system, but rather by the relationship between the executions of actions and the value of some clocks; this is essentially the idea behind timed automata.

This approach can be seen as somewhere in between a coarse structural semantics that is defined only at the timed automata level, and the timed relations defined on the labelled transition systems corresponding to timed automata that would result into undecidability.

The resulting denotational semantics, both in terms of traces and stable failures, turns out to be compositional and decidable. Of course, this comes at a price: on the one hand, this semantics fails to be general enough, but we argue that this problem can be solved

by identifying those timed properties that are preserved through refinement; we show an example of this kind and indicate directions for further work. On the other hand, by using regions, we inherit the problems with complexity; it is possible that this can be improved by using more advanced and efficient techniques, such as zones, but this is left as subject of future research.

**Chapter Outline**

In Section 5.2, we introduce the first semantics model for CCSP, that of Region Traces. In Section 5.3, we extend this model and consider Region Failures. Both this models are considered and compared with the operational semantics in Section 5.4. In Section 5.5, we explain how CCSP could be used to verify some simple timed properties automatically by using FDR2. Finally, in Section 5.6, we draw some conclusions about our approach to timed process algebras by analysing the various aspects of Clocks CSP.

## 5.2 Trace Semantics

We extend the traditional trace model of CSP to include information about the passage of time. This is done by recording how the values of clocks evolve during the execution of the process. Such information is of two kinds: we record both when a set of clocks is reset and when clocks change their values as they a change the region. In this way, given a trace, we can know which clock constraints were satisfied when an action was performed.

### 5.2.1 Region Traces

We give Clocked CSP processes a semantics in terms of region traces. A region trace is an element of $(\Sigma \cup \Delta \cup 2^{\mathcal{C}})^*$, where $\Sigma$ is the process alphabet, $\Delta$ is the set of delay actions and $\mathcal{C}$ is the set of clocks. We denote the extended alphabet $\Sigma \cup \Delta \cup 2^{\mathcal{C}}$ by $\overline{\Sigma}$, $\Sigma \cup \{\checkmark\}$ by $\Sigma^{\checkmark}$ and $\overline{\Sigma} \cup \{\checkmark\}$ by $\overline{\Sigma}^{\checkmark}$. RTraces is the semantic model we study and it denotes the set of non empty, prefix-closed subsets of $\overline{\Sigma}^{\checkmark *}$. An element of a trace either denotes an action ($a \in \Sigma$), the passage of time (delay action $\delta_X \in \Delta$) or the reset of some set of clocks ($X \in 2^{\mathcal{C}}$).

By following the sequences of delays and resets of a trace it is always possible to know the current region and therefore also which clock constraints are met when an action is taken. This makes the synchronisation between traces possible, as two traces must synchronise on all delay and reset actions, thus making sure that they always agree on the values of all clocks.

Note that not all elements of $\overline{\Sigma}^{\checkmark *}$ are possible traces for some process, as the sequence of clocks resets and delays must satisfy certain properties. For example, if we have two clocks $x$ and $y$ and they are reset at the same time, it is not possible for $x$ to advance its value without also $y$ doing so; therefore $\{x, y\}\delta_{\{x,y\}}$ is a possible trace, while $\{x, y\}\delta_{\{x\}}$ is not. These conditions are automatically enforced by the definitions of region traces. Of course this is not the case in standard CSP, where for all traces in $\Sigma^*$ it is possible to construct a process performing such a trace.

We need to be able to define the semantics of a process from any possible starting region and under any possible initial invariant. Consider, for example, the process $(a, x \leq 1) \rightarrow P$: its semantics depends on the semantics of the process $P$ starting from 3 possible regions ($x = 0$, $0 < x < 1$ and $x = 1$), depending on when $a$ is taken. For this reason, the semantics of a process is the set of possible behaviours under any possible starting condition. The

refinement relation is extended accordingly as inverse set inclusion under every starting condition. Formally, let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of regions corresponding to a term and $\mathcal{B}_c(\mathcal{C}) = \{\phi_1, \phi_2, \ldots, \phi_m\}$ the set of possible invariants (note that this set is finite as an invariant is the union of a set of regions). We define a function that returns the region traces of a process, assuming it starts from a particular region under a particular invariant:

$$\mathcal{RT} : \mathbf{CCSP} \times R \times \mathcal{B}_c(\mathcal{C}) \to \mathsf{RTraces} \tag{5.1}$$

**Definition 5.1 (Region trace semantics).** The region trace semantics of a process $P$ is given by the tuple of sets of traces:

$$\begin{aligned} \mathit{RegionTraces}(P) = \; & (\mathcal{RT}(P, r_1, \phi_1), \mathcal{RT}(P, r_1, \phi_2), \ldots, \mathcal{RT}(P, r_1, \phi_m), \\ & \ldots \\ & \mathcal{RT}(P, r_n, \phi_1), \mathcal{RT}(P, r_n, \phi_2), \ldots, \mathcal{RT}(P, r_n, \phi_m)) \end{aligned} \tag{5.2}$$

for all regions $r_i \in R$ and invariants $\phi_j \in \mathcal{B}_c(\mathcal{C})$, where the function $\mathcal{RT}$ is defined according to the rules of Section 5.2.2.

The function $\mathcal{RT}$ is defined inductively on the syntax of terms along the same lines as the derivation rules for traces for classical CSP. The definition of the function $\mathcal{RT}$ is given in the following subsection. The semantic domain for processes in the region trace model is given by $\mathsf{RTraces}^{mn}$, where $m$ is the number of possible invariants and $n$ is the number of possible regions. It is clear that this forms a *complete lattice* under inverse inclusion of components, where the least element is given by the process $STOP$.

We can now define the equivalence and refinement relations between processes based on the notion of region trace semantics.

**Definition 5.2 (Region trace equivalence).** Two CCSP processes $P$ and $Q$ are region trace equivalent if they induce the same region traces, that is,

$$P \equiv_{\mathcal{RT}} Q \text{ iff } \mathit{RegionTraces}(P) = \mathit{RegionTraces}(Q) \tag{5.3}$$

**Definition 5.3 (Region trace refinement).** Given two CCSP processes $P$ and $Q$, we say that $Q$ refines $P$ if every behaviour of $Q$ is a possible behaviour of $P$:

$$\begin{aligned} P \sqsubseteq_{\mathcal{RT}} Q \text{ iff } & \mathit{RegionTraces}(P) \supseteq \mathit{RegionTraces}(Q) \\ \text{iff } & \forall r_i \; \forall \phi_j \; \mathcal{RT}(P, r_i, \phi_j) \supseteq \mathcal{RT}(Q, r_i, \phi_j) \end{aligned} \tag{5.4}$$

### 5.2.2 Rules for Region Traces

We give the definition of the function $\mathcal{RT}$ that yields the set of region traces of a process. First, a few auxiliary definitions are needed. The following operator is necessary to concatenate two traces in the case of sequential composition.

**Definition 5.4.** Given a region trace $t$ and starting region $r$, the region after the execution of $t$ is defined inductively as follows:

- $\mathsf{after}(\langle\rangle, r) = r$
- $\mathsf{after}(a \hat{} t', r) = \mathsf{after}(t', r)$ if $a \in \Sigma$
- $\mathsf{after}(\delta_X \hat{} t', r) = \mathsf{after}(t', \mathit{succ}(r))$ if $\delta_X \in \Delta$

- $\mathsf{after}(X \hat{} t', r) = \mathsf{after}(t', r[X])$ if $X \subseteq \mathcal{C}$

We need to be able to synchronise two traces in the case of the parallel operator. The $\mathsf{synch}_A$ operator is the same as for standard CSP (e.g. [Sch99]); the only difference is that traces are implicitly synchronised not only on the interface alphabet $A$ but also on delay actions $\Delta$ and on clock resets. This guarantees that actions are synchronised only if they are performed under the same clock constraints, which is the very idea behind the region semantics of Clocked CSP. In the following definition *head* and *tail* are the standard operators on traces, returning the first element of a trace and the remainder of a trace, respectively.

**Definition 5.5.** We define $\mathsf{synch}_A$, a relation describing when a trace $tr$ can be a synchronisation of two traces $tr_1$ and $tr_2$ with interface alphabet $A$, by induction on $tr$ as follows:

- $\langle\rangle \, \mathsf{synch}_A \, (tr_1, tr_2)$ iff $tr_1 = tr_2 = \langle\rangle$
- $\langle\checkmark\rangle \, \mathsf{synch}_A \, (tr_1, tr_1)$ iff $tr_1 = tr_2 = \langle\checkmark\rangle$
- $\langle a\rangle \hat{} tr \, \mathsf{synch}_A \, (tr_1, tr_2)$ with $a \neq \checkmark$ iff
  - $a \in (A \cup \Delta \cup \mathcal{C}) \wedge head(tr_1) = head(tr_2) = a \wedge tr \, \mathsf{synch}_A \, (tail(tr_1), tail(tr_2))$ or
  - $a \notin (A \cup \Delta \cup \mathcal{C}) \wedge (head(tr_1) = a \ \wedge tr \, \mathsf{synch}_A \, (tail(tr_1), tr_2)$
    $\vee \ head(tr_2) = a \ \wedge tr \, \mathsf{synch}_A \, (tr_1, tail(tr_2)))$

Given a trace $tr$ and a set of actions $A \subseteq \Sigma$, we denote the subsequence of $tr$ that excludes all the elements in $A$ by $tr \setminus A$.

The function $\mathcal{RT}$ is now defined recursively on CCSP terms by the following rules, which assume the syntactic restrictions described in the previous chapter and that processes are in normal form. Remember that $ch(r)$ denotes the set of clocks whose value is going to change from region $r$ and that $r_{max}$ denotes the region where the value of all clocks exceed the maximal constant against which clocks are compared.

- *Reset*: process $\{\!|X|\!\}P$ first visibly resets the clocks in $X$ and then behaves like $P$ from the region induced by the clock resets. While resetting $X$, it must also be willing to synchronise on the reset of any set of external clocks:

$$\mathcal{RT}(\{\!|X|\!\}P, r, I) = \{\langle\rangle\} \cup \{(X \cup X_e) \hat{} t \mid t \in \mathcal{RT}(P, r[X \cup X_e], I), \forall X_e \subseteq \mathcal{C}_e(\{\!|X|\!\}P)\}$$

- *Invariant*: $\phi \triangleright P$ performs no action but takes into account that $P$'s behaviour is now subject to the new invariant $\phi$.

$$\mathcal{RT}(\phi \triangleright P, r, I) = \mathcal{RT}(P, r, I \wedge \phi)$$

- *STOP*: this process cannot perform any action, but it can synchronise on external clock resets and it can let time elapse (via actions in $\Delta$), while its invariant is satisfied in the current region ($r \models I$) and in the next region ($succ(r) \models I$), and it has not reached the maximal region ($r \neq r_{max}$), that is, only when the predicate $progress(r, I)$ is satisfied.

$$
\begin{aligned}
&\mathcal{RT}(STOP, r, I) = \\
&\quad = \quad \{\langle\rangle\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}(STOP, r[X_e], I), \forall X_e \subseteq \mathcal{C}_e(STOP)\} \cup \\
&\qquad \{\delta_{ch(r)} \hat{} t \mid t \in \mathcal{RT}(STOP, succ(r), I)\} \\
&\qquad\qquad\qquad\qquad \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\
&\quad = \quad \{\langle\rangle\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}(STOP, r[X_e], I), \forall X_e \subseteq \mathcal{C}_e(STOP)\} \qquad \text{otherwise}
\end{aligned}
$$

- *SKIP*: this process behaves like *STOP*, except that at each point it can successfully terminate by executing ✓.

$$
\begin{aligned}
&\mathcal{RT}(SKIP, r, I) = \\
&\quad = \quad \{\langle\rangle, \langle\checkmark\rangle\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}(SKIP, r[X_e], I), \forall X_e \subseteq \mathcal{C}_e(SKIP)\} \cup \\
&\qquad \{\delta_{ch(r)} \hat{} t \mid t \in \mathcal{RT}(SKIP, succ(r), I)\} \\
&\qquad\qquad\qquad\qquad \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\
&\quad = \quad \{\langle\rangle\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}(SKIP, r[X_e], I), \forall X_e \subseteq \mathcal{C}_e(SKIP)\} \qquad \text{otherwise}
\end{aligned}
$$

- *External choice*: in this case, the process can let time elapse (under the same conditions as for *STOP*), synchronise on external clocks or execute one of the possible choice actions whose invariant is currently true.

$$
\begin{aligned}
&\mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, r, I\right) \\
&\quad = \quad \{\langle\rangle\} \cup \{a_i \hat{} t \mid t \in \mathcal{RT}\left(P_i, r, \mathtt{tt}\right) \wedge r \models \varphi_i\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, r[X_e], I\right), \forall X_e \subseteq \mathcal{C}_e(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i)\} \\
&\qquad \left\{\delta_{ch(r)} \hat{} t \mid t \in \mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, succ(r), I\right)\right\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\
&\quad = \quad \{\langle\rangle\} \cup \{a_i \hat{} t \mid t \in \mathcal{RT}\left(P_i, r, \mathtt{tt}\right) \wedge r \models \varphi_i\} \cup \\
&\qquad \{X_e \hat{} t \mid t \in \mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, r[X_e], I\right), \forall X_e \subseteq \mathcal{C}_e(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i)\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise}
\end{aligned}
$$

- *Internal choice*: since internal choice is resolved immediately and internally, the traces of this process are given by union of the traces of the $P_i$ processes starting from the same region. Note that after the choice is resolved, the new process starts under the invariant `tt`.

$$\mathcal{RT}(\sqcap_{i=1}^{N} P_i, r, I) = \bigcup_{i=1}^{N} \mathcal{RT}(P_i, r, \mathtt{tt})$$

- *Interface parallel*: the traces of the parallel construct are obtained by synchronising the traces of both the processes involved. Since clock resets and delays are recorded in the traces, two traces that synchronise always agree on the values of clocks at each point.

$$
\begin{aligned}
\mathcal{RT}(P_1 \underset{A}{||} P_2, r, I) = &\{t \mid \exists t_1, t_2 \,.\, t_1 \in \mathcal{RT}(P_1, r, I) \ \wedge t_2 \in \mathcal{RT}(P_2, r, I) \\
&\wedge t \,\mathsf{synch}_A\, (t_1, t_2)\}
\end{aligned}
$$

- *Hiding*: the traces of $P \setminus A$ are those of $P$ with all the actions in $A$ removed from each trace.

$$\mathcal{RT}(P \setminus A, r, I) = \{t \setminus A \mid t \in \mathcal{RT}(P, r, I)\}$$

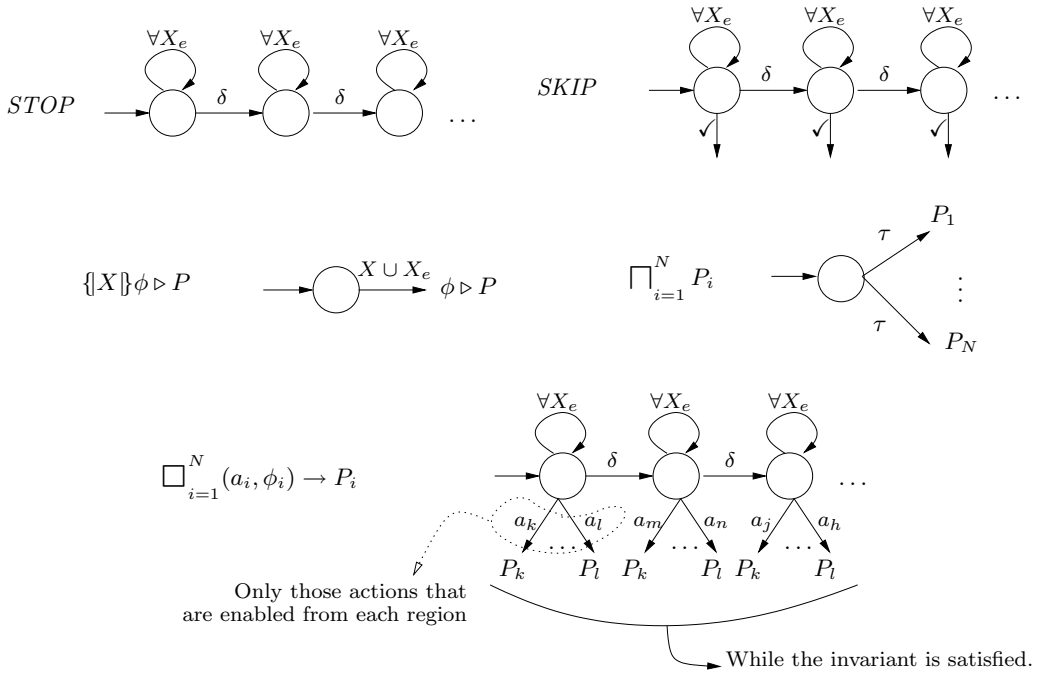



Figure 5.1: The region automata corresponding to some operators.

- *Renaming*: the traces of a $f[P]$ are those of $P$ renamed according to the renaming function $f$.

$$\mathcal{RT}(f[P], r, I) = \{t \mid f^{-1}(t) \in \mathcal{RT}(P, r, I)\}$$

- *Sequential composition*: the traces of $P_1; P_2$ are those of $P_1$ that end in $\checkmark$ concatenated to those of $P_2$, that executes only when $P_1$ terminates successfully and also all of the traces of $P_1$ that do not successfully terminate.

$$\begin{aligned}\mathcal{RT}(P_1; P_2, r, I) = &\{t \mid \exists t_1, t_2 \,.\, t = t_1 \hat{}\, t_2 \;\wedge\; t_1 \hat{}\, \checkmark \in RT(P_1, r, I) \\ &\qquad \wedge\; t_2 \in \mathcal{RT}(P_2, \mathsf{after}(t_1, r), \mathtt{tt})\} \\ &\cup \{t \mid t \in \mathcal{RT}(P_1, r, I) \;\wedge\; t \neq t' \hat{}\, \langle\checkmark\rangle \;\forall t' \in \overline{\Sigma}^{\checkmark *}\}\end{aligned}$$

Figure 5.1 helps to understand these rules by giving the intuition of the operational model underlying a term. Some CCSP operators behave exactly as in CSP. For example, internal choice is resolved immediately as its invariant is false (see Table 4.4) and a process is chosen via a $\tau$ transition. The behaviour of clock reset is quite simple: the process $\{\!|X|\!\}\phi \triangleright P$ first resets the set of clocks $X$ (this is a visible action and it appears in the trace, see Table 4.1), then it behaves like $P$ under the invariant $\phi$. $\{\!|X|\!\}\phi \triangleright P$ must also be able to synchronise with other processes that want to reset their clocks; by performing the action $X \cup X_e$, for all $X_e \subseteq \mathcal{C}_e$, the process takes into account the possibility of external clocks being reset. The behaviour of external choice is quite involved when compared with the corresponding rule for standard CSP. The reason for this is that, in Clocked CSP, time can elapse ($\delta$ actions) before the choice is resolved. This explains why there are several states in the operational model to represent choice. The actions whose guards are satisfied are enabled from each region. Finally, from each state, the process must be willing to synchronise with resets of external clocks of other processes. Reset actions of external clocks would in general lead to different states and would not correspond to self

loops, but we represent them this way for conciseness since they do not alter the values of internal clocks and the same actions are enabled after external clock resets. Since no delay action is possible from $r_{max}$, we cannot have an infinite number of delays without actions or clock resets taking place. The case for *SKIP* is similar, as this process can let time elapse while its invariant is true and must successfully terminate by executing a ✓ action. *STOP* behaves similarly, but without the possibility to successfully terminate via a ✓ action.

**Recursion.** We treat recursion in the same way as standard CSP and the semantics of $\mathsf{rec}\, Z.P$ is given by the least fixed point of the term $P$. Let $P^0(STOP) = STOP$ and $P^{n+1}(STOP) = P[P^n(STOP)/Z]$, that is, the $(n+1)$-th unwinding of $P$ starting from the least process $STOP$; then the least fixed point is given by $\mathit{RegionTraces}(\mathsf{rec}\, Z.P) = \bigcup_{n=0}^{\infty} \mathit{RegionTraces}(P^n(STOP))$ and by $\mathcal{RT}(\mathsf{rec}\, Z.P,\, r,\, I) = \bigcup_{n=0}^{\infty} \mathcal{RT}(P^n(STOP), r, I)$. Since the trace domain is a complete lattice, we need to show that all operators are monotonic with respect to the inverse set inclusion in order to prove that recursion is well defined for guarded processes and there exists a least fixed point [Ros98].

**Theorem 5.6.** *RegionTraces is a monotonic function with respect to all operators, and the least fixed point operator exists for guarded processes.*

*Proof.* The result is straightforward for most operators following the standard arguments for CSP [Ros98]. Guardedness ensures the existence of a unique fixed point. Let us then prove monotonicity for the added operators:

- (Reset) $P_1 \sqsubseteq_{\mathcal{RT}} P_2 \implies \{\!|X|\!\}P_1 \sqsubseteq_{\mathcal{RT}} \{\!|X|\!\}P_2$: we have $\{\!|X|\!\}P_1 \sqsubseteq_{\mathcal{RT}} \{\!|X|\!\}P_2$ iff $\forall r, I.\ \mathcal{RT}(\{\!|X|\!\}P_1, r, I) \supseteq \mathcal{RT}(\{\!|X|\!\}P_2, r, I)$. By applying the rule for reset, we get that this inclusion is true if and only if $\mathcal{RT}(P_1, r[X \cup X_e], I) \supseteq \mathcal{RT}(P_2, r[X \cup X_e], I)$, which is true by induction.

- (Invariant) $P_1 \sqsubseteq_{\mathcal{RT}} P_2 \implies \phi \triangleright P_1 \sqsubseteq_{\mathcal{RT}} \phi \triangleright P_2$. The same argument as above applies for this operator.

- (External choice) We can consider external choice with two terms as the argument easily extends to the finitary case. Let us consider $P_1 = (a_1, \phi_1) \to Q_1 \,\Box\, (a_2, \phi_2) \to Q$ and $P_2 = (a_1, \phi_1) \to Q_2 \,\Box\, (a_2, \phi_2) \to Q$ such that $Q_1 \sqsubseteq_{\mathcal{RT}} Q_2$, then we have to prove that $P_1 \sqsubseteq_{\mathcal{RT}} P_2$.

  Note that the rule for external choice is defined in terms of itself: this is because, before taking an action, a process can allow to let time elapse a finite number of times, and, moreover, delay actions can be interleaved by a sequence of external clock resets. The idea is that both processes can execute the same sequence of delay and resets, and then an action is taken and the inductive step can be applied. Assume that such sequences are finite; if they are not, the two processes clearly have the same set of traces. So, before enabling an action, both processes can execute the same sequences of delays (they have the same invariant and the actions are guarded by the same condition) and external resets (neither process has control of this part) and they both end up in the same region. Assume some sequence of delays and resets $\delta X_1 \cdots \delta X_j$; the region after this sequence is given by $\mathit{succ}(\mathit{succ}(\cdots \mathit{succ}(r)[X_1] \cdots)[X_{j-1}])[X_j]$. If at this point $a_1$ is taken, then the inductive step can be applied, else if $a_2$ is taken, we trivially get the same trace.

- The proofs for *SKIP* and *STOP* proceed in the same way.

□

**Example 5.1 (Region Traces).** Consider the process $P = \{\!|x|\!\}(x < 2) \triangleright (a, x \geq 1) \rightarrow STOP$ from Example 4.5. Assume $\mathcal{C}_e(P) = \emptyset$, then we can use the rules given above to find the region traces of this process from the initial region $r_0 = (x = 0)$:

$$\mathcal{RT}(P, x = 0, \mathtt{tt}) = \{\langle\rangle\}$$
$$\quad \cup \{\{x\}^\frown t \mid t \in \mathcal{RT}((x < 2) \triangleright (a, x \geq 1) \rightarrow STOP, x = 0, \mathtt{tt})\}$$
$$\mathcal{RT}((x < 2) \triangleright (a, x \geq 1) \rightarrow STOP, x = 0, \mathtt{tt}) = \mathcal{RT}((a, x \geq 1) \rightarrow STOP, x = 0, x < 2)$$
$$\mathcal{RT}((a, x \geq 1) \rightarrow STOP, x = 0, x < 2) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}((a, x \geq 1) \rightarrow STOP, 0 < x < 1, x < 2)\}$$
$$\mathcal{RT}((a, x \geq 1) \rightarrow STOP, 0 < x < 1, x < 2) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}((a, x \geq 1) \rightarrow STOP, x = 1, x < 2)\}$$
$$\mathcal{RT}((a, x \geq 1) \rightarrow STOP, x = 1, x < 2) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}((a, x \geq 1) \rightarrow STOP, 1 < x < 2, x < 2)\}$$
$$\quad \cup \{a^\frown t \mid t \in \mathcal{RT}(STOP, x = 1, \mathtt{tt})\}$$
$$\mathcal{RT}((a, x \geq 1) \rightarrow STOP, 1 < x < 2, x < 2) = \{\langle\rangle\}$$
$$\quad \cup \{a^\frown t \mid t \in \mathcal{RT}(STOP, 1 < x < 2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x = 1, \mathtt{tt}) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}(STOP, 1 < x < 2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, 1 < x < 2, \mathtt{tt}) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}(STOP, x = 2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x = 2, \mathtt{tt}) = \{\langle\rangle\}$$
$$\quad \cup \{\delta_x{}^\frown t \mid t \in \mathcal{RT}(STOP, x > 2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x > 2, \mathtt{tt}) = \{\langle\rangle\}$$

since $r_{\max} = x > 2$. From the equations above we get that the two maximal traces

$$\langle \{x\}\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x\, \delta_x \rangle$$
$$\langle \{x\}\, \delta_x\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x \rangle$$

of Example 4.3 are in $\mathcal{RT}(P, r_0, \mathtt{tt})$, as expected. $\mathcal{RT}(P, r_0, \mathtt{tt})$ contains these two traces and all of their possible prefixes.

## 5.3 Failure Semantics

Having defined the semantic model for region traces, the next natural step is to extend it to a finer semantics that distinguishes between stable failures.

We augment the standard notion of failure for an untimed process to include information on clock resets and delays: the same actions that have been included in a trace can also be refused by a process. In this way, we include some branching information in our semantics and thus refine the linear approach of region traces in order to verify certain timed liveness properties. As an example, consider the case when a process refuses a delay action: refusing a delay action means refusing to let time elapse, so one could specify that

an action $a$ must happen before some bound by refusing to let time elapse until $a$ has been performed.

Our approach differs from the timed failures semantics of Timed CSP: in our case a failure explicitly records the impossibility to let time elapse by including delay actions in its refusal. Timed CSP, on the other hand, does not record this information explicitly and instead records the time intervals after a trace during which certain actions are refused. Moreover, by recording time information in the traces and the refusals as real values, timed failures are a more general model; however, like in the case of timed traces, this leads to a more complex semantics, especially from the decidability point of view.

### 5.3.1 Region Failures

We define *region failures*, again having in mind the operational model of region automata. A region refusal set $F$ is a subset of $\overline{\Sigma}^{\checkmark}$ that describes the set of actions that a process can refuse after a specified trace. A region failure is a pair $(t, F)$, where $t$ is a region trace and $F$ is a refusal set. We denote the semantic domain of region failures by RFailures. Following a line of reasoning similar to that for region traces, we obtain a new semantic model for Clocked CSP processes, given by the *RegionFailures* function which is once again congruent with the operational model, and a refinement relation $\sqsubseteq_{\mathcal{RF}}$.

Formally, we define a function $\mathcal{RF}$ that maps a process to its set of region failures under an invariant $\phi$ from a region $r$ by:

$$\mathcal{RF} : \mathbf{CCSP} \times R \times \mathcal{B}_c(\mathcal{C}) \to \mathsf{RFailures} \tag{5.5}$$

$\mathcal{RF}$ is defined inductively on the syntax of CCSP terms in a similar way to the function $\mathcal{RT}$.

**Definition 5.7 (Region failure semantics).** The region failure semantics of a process $P$ is given by the ordered set of sets of failures:

$$\begin{aligned} \mathit{RegionFailures}(P) =& (\mathcal{RF}(P, r_1, \phi_1), \mathcal{RF}(P, r_1, \phi_2), \ldots, \mathcal{RF}(P, r_1, \phi_m), \\ & \ldots \\ & \mathcal{RF}(P, r_n, \phi_1), \mathcal{RF}(P, r_n, \phi_2), \ldots, \mathcal{RF}(P, r_n, \phi_m)) \end{aligned} \tag{5.6}$$

for all regions $r_i$ and invariants $\phi_j$, where the function $\mathcal{RF}$ is defined according to the rules of Section 5.3.2.

We can define the equivalence and refinement relations between processes based on the notion of region failure semantics:

**Definition 5.8 (Region failure equivalence).** Two CCSP processes $P$ and $Q$ are region failure equivalent if they induce the same same region failures, that is,

$$P \equiv_{\mathcal{RT}} Q \text{ iff } \mathit{RegionFailures}(P) = \mathit{RegionFailures}(Q) \tag{5.7}$$

**Definition 5.9 (Region failure refinement).** Given two CCSP processes $P$ and $Q$, we say that $Q$ refines $P$ if every behaviour of $Q$ is a possible behaviour of $P$:

$$\begin{aligned} P \sqsubseteq_{\mathcal{RF}} Q &\text{ iff } \mathit{RegionFailures}(P) \supseteq \mathit{RegionFailures}(Q) \\ &\text{iff } \forall r_i \; \forall \phi_j \; \mathcal{RF}(P, r_i, \phi_j) \supseteq \mathcal{RF}(Q, r_i, \phi_j) \end{aligned} \tag{5.8}$$

As for standard CSP, we introduce the maximal element for stable failures, $DIV$, that executes no action and does not refuse anything, whose semantics is given by $\mathcal{RF}(DIV, r, I) = \{(\langle\rangle, \{\})\}$. The semantic domain for region failures forms a complete lattice under reverse inclusion, like the stable failures domain of standard CSP.

### 5.3.2 Rules for Region Failures

The rules for region failures are similar to those for region traces; Figure 5.1 illustrates the correspondence with the operational model and Theorem 5.11 will relate the rules above with the operational model in more detail. In the following, we explain the refusals for each operator, as the corresponding traces are those of the Section 5.2.

- *Reset*: $\{\!|X|\!\}P$ can refuse any set of actions, except the actions of resetting sets of clocks that do not include $X$.

$$\mathcal{RF}(\{\!|X|\!\}P, r, I) = \Big\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark} \wedge X \not\subseteq F\Big\} \cup \\ \{(X \cup X_e)\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(\{\!|X|\!\}P \wedge (t, F) \in \mathcal{RF}(P, r[X \cup X_e], I)\}$$

- *Invariant*: as for traces, $\phi \triangleright P$ performs no actions. The invariant can cause the process to refuse actions in $\Delta$; this is recorded by the new invariant $I \wedge \phi$.

$$\mathcal{RF}(\phi \triangleright P, r, I) = \mathcal{RF}(P, r, I \wedge \phi)$$

- *STOP*: this process can synchronise on external clock resets or let time elapse, if possible. Therefore, it can refuse all clock reset actions including internal clocks and all delay actions, except for the one it possibly enables.

$$\begin{array}{lll} \mathcal{RF}(STOP, r, I) = & & \\ = & \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(STOP) \wedge (t, F) \in \mathcal{RF}\,(STOP, r[X_e], I)\} \cup & \\ & \Big\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark} \setminus \big(2^{\mathcal{C}_e} \cup \{\delta_{ch(r)}\}\big)\Big\} \cup & \\ & \big\{(\delta_{ch(r)}\hat{}t, F) \mid (t, F) \in \mathcal{RF}\,(STOP, succ(r), I)\big\} & \\ & & \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\ = & \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(STOP) \wedge (t, F) \in \mathcal{RF}\,(STOP, r[X_e], I)\} \cup & \\ & \Big\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark} \setminus 2^{\mathcal{C}_e}\Big\} & \text{otherwise} \end{array}$$

- *SKIP*: this process behaves like *STOP*, except that it cannot refuse ✓.

$$\begin{array}{lll} \mathcal{RF}(SKIP, r, I) = & & \\ = & \{(\langle\checkmark\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark}\} \cup & \\ & \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(SKIP) \wedge (t, F) \in \mathcal{RF}\,(SKIP, r[X_e], I)\} \cup & \\ & \big\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma} \setminus \big(2^{\mathcal{C}_e} \cup \{\delta_{ch(r)}\}\big)\big\} \cup & \\ & \big\{(\delta_{ch(r)}\hat{}t, F) \mid (t, F) \in \mathcal{RF}\,(SKIP, succ(r), I)\big\} & \\ & & \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\ = & \{(\langle\checkmark\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark}\} \cup & \\ & \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(SKIP) \wedge (t, F) \in \mathcal{RF}\,(SKIP, r[X_e], I)\} \cup & \\ & \big\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma} \setminus 2^{\mathcal{C}_e}\big\} & \text{otherwise} \end{array}$$

- *External choice*: at each point, $\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i$ can refuse any action $a$ that is not currently enabled (that is, $a$ is different from all the $a_i$'s or $a$'s guards are all false for all $a_i = a$), all clock resets that include some internal clock and all delay actions, except the next delay action, if possible.

$$
\begin{aligned}
&\mathcal{RF}\left(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i, r, I\right) = \\
&\quad = \quad \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i) \wedge \\
&\qquad\qquad (t,F) \in \mathcal{RF}(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i, r[X_e], I)\} \cup \\
&\qquad \{(a_i\hat{}t, F) \mid (t,F) \in \mathcal{RF}\,(P_i, r, \mathtt{tt}) \wedge r \models \varphi_i\} \cup \\
&\qquad \left\{(\langle\rangle, F) \mid F \subseteq \left(\Sigma^{\checkmark} \setminus \{a_j \mid r \models \varphi_j\}\right) \cup 2^{\mathcal{C}_i} \cup \left(\Delta \setminus \{\delta_{ch(r)}\}\right)\right\} \cup \\
&\qquad \left\{(\delta_{ch(r)}\hat{}t, F) \mid (t,F) \in \mathcal{RF}\left(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i, succ(r), I\right)\right\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{if } r \models I \wedge r \neq r_{max} \wedge succ(r) \models I \\
&\quad = \quad \{(X_e\hat{}t, F) \mid X_e \subseteq \mathcal{C}_e(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i) \wedge \\
&\qquad\qquad (t,F) \in \mathcal{RF}(\Box_{i=1}^{N}(a_i,\varphi_i)\rightarrow P_i, r[X_e], I)\} \cup \\
&\qquad \{(a_i\hat{}t, F) \mid (t,F) \in \mathcal{RF}\,(P_i, r, \mathtt{tt}) \wedge r \models \varphi_i\} \cup \\
&\qquad \left\{(\langle\rangle, F) \mid F \subseteq \left(\Sigma^{\checkmark} \setminus \{a_j \mid r \models \varphi_j\}\right) \cup 2^{\mathcal{C}_i} \cup \Delta\right\} \qquad\qquad \text{otherwise}
\end{aligned}
$$

- *Internal choice*: since internal choice is resolved immediately, its failures are given by the union of the failures of the components.

$$\mathcal{RF}(\sqcap_{i=1}^{N} P_i, r, I) = \bigcup_{i=1}^{N} \mathcal{RF}(P_i, r, \mathtt{tt})$$

- *Interface parallel*: failures of the composition between two processes are obtained by synchronising the failures of the components in the following way: traces are synchronised in the same way as for region traces; then, after a trace, the composition can refuse actions in the interface alphabet $A$ if one of the component refuse them (they must be executed by both components in synchronisation), and actions not in $A$ if both components refuse them (or else one component can execute it independently). This applies also to clock resets and delay actions, on which processes must synchronise and therefore are implicitly part of the interface.

$$
\begin{aligned}
\mathcal{RF}(P \underset{A}{||} Q, r, I) = \{&(t, F \cup G) \mid F \setminus (A \cup \{\checkmark\}) = G \setminus (A \cup \{\checkmark\}) \\
&\wedge\ \exists t_1, t_2 \,.\, (t_1, F) \in \mathcal{RF}(P, r, I)\ \wedge\ (t_2, G) \in \mathcal{RF}(Q, r, I) \\
&\wedge\ t\ \mathsf{synch}_A\,(t_1, t_2)\}
\end{aligned}
$$

- *Hiding*: $(t, F)$ is a failure of $P \setminus A$ if, after trace $t$, process $P$ can refuse all the actions in $F$ and also those in $A$; this last condition ensures that no actions in $A$ was enabled in $P$ and that the corresponding state remains stable after the hiding of $A$.

$$\mathcal{RF}(P \setminus A, r, I) = \{(s \setminus A, F) \mid (s, A \cup F) \in \mathcal{RF}(P, r, I)\}$$

- *Renaming*: the behaviour of this operator is simple, as the failures of $f[P]$ are obtained by renaming both the traces and the refusal sets of $P$ according to the bijection $f$.

$$\mathcal{RF}(f[P], r, I) = \{(t, F) \mid (f^{-1}(t), f^{-1}(F)) \in \mathcal{RF}(P, r, I)\}$$

- *Sequential composition*: the failures of $P; Q$ are given by the failures of $P$ whose trace does not terminate successfully and by the traces of $P$ that terminate successfully followed by the failures of $Q$.

$$
\begin{aligned}
\mathcal{RF}(P; Q, r, I) = &\{(s, F) \mid (s, F \cup \{\checkmark\}) \in \mathcal{RF}(P, r, I)\} \\
&\cup \{(s\hat{}t, F) \mid s\hat{}\langle\checkmark\rangle \in \mathcal{RT}(P, r, I) \wedge (t, F) \in \mathcal{RF}(Q, \mathsf{after}(s, r), \mathtt{tt})\}
\end{aligned}
$$

**Recursion.** Again, we define the semantics of recursion $\mathsf{rec}\, Z.P$ as the least fixed point of $P$, that is, $\mathcal{RF}(\mathsf{rec}\, Z.P, r, I) = \bigcup_{n=0}^{\infty} \mathcal{RF}(P^n(DIV), r, I)$. The following theorem proves that recursion is well defined for guarded processes.

**Theorem 5.10.** *RegionFailures is a monotonic function with respect to all operators and the least fixed point operator is well-defined for guarded processes.*

*Proof outline.* The proof follows closely the proof or Theorem 5.6. Again, we consider only the novel operators as the result is clear for standard CSP operators.

- Reset: let us consider two processes, $\{\!|X|\!\}P_1$ and $\{\!|X|\!\}P_2$, such that $P_1 \sqsubseteq_{\mathcal{RF}} P_2$. From the rule for reset, the first set of region failures, defined as $\{(\langle\rangle, F) \mid F \subseteq \Sigma^{\checkmark} \cup \Delta \cup 2^{C \setminus X}\}$, is common between the two processes, while, clearly, we have inclusion of $\{(X \cup X_e \hat{} t, F) \mid X_e \subseteq \mathcal{C}_e \wedge (t, F) \in \mathcal{RF}(P, r[X \cup X_e], I))\}$ if and only if we have inclusion of the failure sets of the components, which is true by induction.

- Invariant: this case is immediate from the definition of region refusals for the invariant operator.

- External choice: we can use the same arguments as for the region trace semantics; the operator is defined in terms of itself, but before using the recursive rule both processes can take the same actions and refuse the same sets. If we analyse the rule for external choice, both processes can execute a finite sequence of delays, possibly interleaved by external clock reset; moreover, they can refuse the same sets for each action. Then both processes can execute the same action, under the same starting conditions, and the recursive step can be applied, as we did for the trace semantics. The arguments used for region traces apply in this case.

□

## 5.4 Analysing the Semantic Models

The semantics we described in the previous section is discrete and compositional: we have abstracted from the dense representation of time and we have managed to define the behaviour of each operator in terms of its components. In order to be able to automatically verify the refinement relations defined, we relate the denotational model to the operational one, thus allowing us to use the existing techniques used, for example, by FDR2.

### 5.4.1 Comparison with the Operational Semantics

We show that the denotational semantics we have given in this section is congruent with the operational semantics of Section 4.4. This is not surprising since we have modelled the trace semantics on region automata, but it is nevertheless important as it permits model checking of refinement relations using the operational model as done for CSP [Ros94].

#### Region Automata with External Clocks

We will prove the congruence result by structural induction on the terms. We have to show that, for every process, the failure semantics of the region automaton induced by the timed automaton obtained with the operational semantics is the same as the failure semantics obtained by applying the denotational rules.

In order to this, we have to add the notion of external clocks to the definition of region automata given in Section 4.2. As shown in Figure 5.1, the idea of external clocks is that, at any point in the computation, a component must be able to synchronise with external clock reset, thus not being able to refuse such actions. Given a timed automaton $\mathcal{A} = (L, \overline{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, we partition its set of clocks $\mathcal{C}$ into the set of internal clocks $\mathcal{C}_i$, that is, those clocks that are explicitly used by $\mathcal{A}$ and its complement, the external clocks $\mathcal{C}_e$. Then the rules for transitions of region automata of Definition 4.3 have to be modified as follows: from any state $(l, r)$, the transitions $(l, r) \xrightarrow{X_e}_r (l, r[X_e])$, for all $X_e \subseteq \mathcal{C}_e$, must be enabled, and the rule for reset of clocks $X \in \mathcal{C}_i$ must be modified by allowing external resets, that is, $(l, r) \xrightarrow{X \cup X_e}_r (l, r[X \cup X_e])$, for all $X_e \subseteq \mathcal{C}_e$. We keep explicit information about reset of the empty set of clocks in order to be able to synchronise with other clocks. The other types of action are not modified; moreover, note that the same actions are enabled from different regions if they disagree only on the values of external clocks.

### Congruence

We can now prove the main result of this section.

**Theorem 5.11.** *The function $\mathcal{RF}$ is a congruence with respect to the operational semantics. That is, for all CCSP terms in normal form, regions $r$ and invariants $I$, the following holds:*

$$\mathcal{RF}(P, r, I) = \mathcal{F}(R(P, r, I))$$

*where $\mathcal{F}(R(P, r, I))$ denotes the set of stable failures of the region automaton corresponding to process $P$ under the given starting conditions.*

*Proof.* We prove the result by structural induction on the syntax of processes in normal form. In Figure 5.1 the region automata corresponding to some operators are shown, to highlight the correspondence between the operational model and the denotational rules for region failures. We start from the base cases:

- *STOP*: this process can only let time elapse or synchronise on external clock resets; these are the clauses of the rule for $\mathcal{RF}(STOP)$. All actions except for external clock resets or the next delay can be refused.
- *SKIP*: in this case the process can let time elapse (only while the invariant remains satisfied), synchronise on external clocks or successfully terminate by executing a ✓ action. All these are the clauses of the rules for failures. At each step the process can refuse any action except that ✓ and external clock resets; after the ✓, any set of actions is refused.

The inductive steps:

- $\{\!|X|\!\}\phi \triangleright P$: we deal with these two operators together. These operators introduce the invariant $\phi$ and reset the clocks in $X$. From the initial state of the operational model, clocks $X \cup X_e$, for all $X_e \subseteq \mathcal{C}_e$, can be reset, and all other actions must be refused; this corresponds to the denotational rule. After the transition, the inductive step applies. From the structure of the syntax, we know that the sub-term $P$ has no clock reset; this shows the importance of having a normal form, so that we have a unique reset action.
- $P = \bigsqcap_{i=1}^{N} P_i$: in the operational model a $\tau$ action must be taken immediately, therefore the only traces allowed are those of the components, and the process can refuse everything that the components can refuse.

- $P = \Box_{i=1}^{N}(a_i, \varphi_i) \to P_i$: Figure 5.1 helps to understand the rule for external choice and why it is congruent to the operational semantics. Similarly to *SKIP*, before the choice is resolved and an action $a_j$ is executed, thus reducing to the inductive step, the operational model can reset external clocks or let time elapse; for instance, from any state $(P, r)$ of the figure, the automaton can either execute the actions whose guards are satisfied, or it can let time elapse and behave like $(P, succ(r))$, or reset some external clocks $X_e$ and behave like $(P, r[X_e])$. All other actions are refused. All these cases correspond to the clauses of the denotational rule.

- $P = P \underset{A}{||} Q$: by induction, we know that $\mathcal{RF}(P, r, \mathtt{tt}) = \mathcal{F}(R(P, r))$ and that $\mathcal{RF}(Q, r, \mathtt{tt}) = \mathcal{F}(R(Q, r))$ (the invariants must be true for components of the parallel operator because we deal with processes in normal form). Fix the interface alphabet $A$, which we are going to omit in the following for sake of simplicity. Following standard CSP arguments, it can be shown that $\mathcal{RF}(P \,||\, Q, r, \mathtt{tt})$ has the same failures obtained by synchronising $R(P, r)$ and $R(Q, r)$ on the actions in $A \cup \Delta \cup 2^{\mathcal{C}}$, that is, $\mathcal{F}(R(P, r) \,||\, R(Q, r)) = \mathcal{RF}(P \,||\, Q, r, \mathtt{tt})$. In order to show this equality, we show that the region automaton of the composition of two terms is equal to the composition of the region automata of each term, that is, $R(P \,||\, Q, r) = R(P, r) \,||\, R(Q, r)$.

  Recall that the state space for region automata corresponding to a CCSP term is given by the terms in $\overline{\mathbf{CCSP}^+}$ extended with the operator $free$ of timed automata, which we denote by $\overline{\mathbf{CCSP}^+}$. Note the different role in this case of $\mathsf{free}$, the operator on CCSP terms, and $free$, the operator on the locations of timed automata; for instance, from the operational semantics and the definition of region automata, the region automaton corresponding to the process $\mathsf{free}(P)$ has an initial reset action on the empty set of clocks. We get $R(P, r) \,||\, R(Q, r) = ((\overline{\mathbf{CCSP}^+} \times R) \times (\overline{\mathbf{CCSP}^+} \times R), \Sigma \cup \Delta \cup 2^{\mathcal{C}}, ((P, r), (Q, r)), \longrightarrow)$ and $R(P \,||\, Q, r) = ((\overline{\mathbf{CCSP}^+} \times R), \Sigma \cup \Delta \cup 2^{\mathcal{C}}, (P \,||\, Q, r), \to)$. Let us show that each transition of one is matched by a transition of the other.

  - $((P', r'), (Q', r')) \xrightarrow{a} ((P'', r'), (Q'', r'))$, with $a \in A$, is a transition for the parallel if both components can make this transition, which is true if both $P' \xrightarrow{a,\varphi_1} P''$ and $Q' \xrightarrow{a,\varphi_2} Q''$ with $r' \models \varphi_1$ and $r' \models \varphi_2$. Note that actions do not cause a change of region, and therefore if the two processes have the same initial region, they also have the same unchanged final region. This is also the condition for synchronisation for the parallel: $P' \,||\, Q' \xrightarrow{a,\varphi_1 \wedge \varphi_2} P'' \,||\, Q''$ allows an $a$ action in the corresponding region automaton if both guards are satisfied and if both components are willing to take an $a$ action.
  - $((P', r'), (Q', r')) \xrightarrow{a} ((P'', r'), (Q', r'))$, with $a \in \overline{A} \cup \{\tau\}$, is a transition for the parallel if $(P', r')$ can independently execute this action, which is true if $P' \xrightarrow{a,\varphi} P''$. After executing $a$, $P''$ can reset its set of clocks, and $Q'$ will be able to synchronise on any external set of clocks. The parallel composition, $P \,||\, Q$ can make the same transition under the same conditions, that is, $P' \,||\, Q' \xrightarrow{a,\varphi} P'' \,||\, \mathsf{free}(Q')$. At this point $P''$ can reset its set of clocks, thus behaving in the same way as above.
  - $((P', r'), (Q', r')) \xrightarrow{X} ((P'', r'[X]), (Q'', r'[X]))$ is a transition for the parallel if both components can reset the clocks in $X$; suppose that $\kappa(P') = X_1$, $\kappa(Q') = X_2$ and that we can partition $X$ into three sets $X_1 \cup X_2 \cup X_3$, such that $X_1$ and $X_2$ are internal clocks for $P'$ and $Q'$, respectively, and $X_3$ are external to

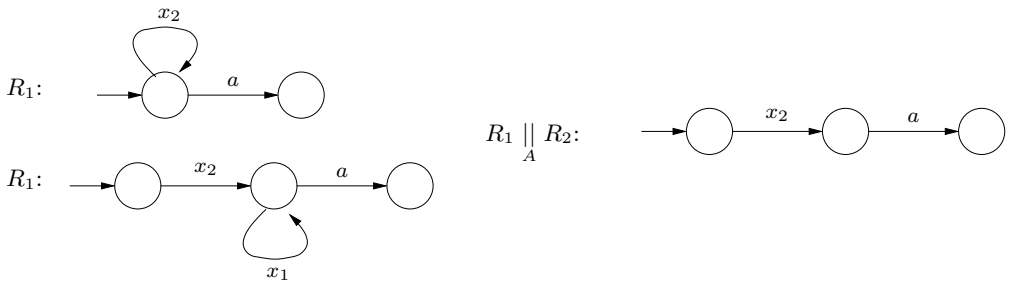



Figure 5.2: Two region automata, $R_1$ with internal clock $x_1$ and $R_2$ with internal clock $x_2$, and their parallel composition $R_1 \underset{A}{||} R_2$.

both. Then $(P', r) \xrightarrow{X} (P', r[X])$ because $P$ resets its clocks $X_1$ and is willing to synchronise on any external clock reset; the same applies to $Q'$. Therefore, they can synchronise on this action.

On the other hand, the composition $P' \,||\, Q'$ will have $X_1 \cup X_2$ as the set of internal clocks to be reset ($\kappa(P' \,||\, Q') = X_1 \cup X_2$), and $X_3$ as the set of external clocks on which it is willing to synchronise. Therefore $(P' \,||\, Q', r')$ will reset its own clocks $X_1 \cup X_2$ and it will also be able to synchronise on all other clocks, thus is must be able to execute $X$.

– $((P', r'), (Q', r')) \xrightarrow{\delta} ((P', succ(r')), (Q', succ(r')))$, for $\delta \in \Delta$, if both components can execute such action, that is if they can both let time elapse and $succ(r') \models I(P')$ and $succ(r') \models I(Q')$, which is the condition for the $P' \,||\, Q'$ to let time elapse. Therefore also $(P' \,||\, Q', r') \xrightarrow{\delta} (P' \,||\, Q', succ(r))$.

Note that it is not possible to define a transition for which the two components end up in states with different regions. Therefore we can consider the state spaces of the two automata as equivalent. The inverse inclusion is shown using exactly the same arguments.

We point out that after two terms are combined by parallel composition, they join their internal clocks into a new set of internal clocks (see Table 4.3); for this reason, they synchronise on the reset operations and they discard all the reset operations regarding internal clocks that have not been used; if they have no more external clocks, then there are no more "dangling" external clock reset actions (see Figure 5.2). Also note that, from the structure of the syntax, the first actions that both processes execute are reset actions, on which they immediately synchronise.

- $P; Q$: this case is trivial as the failures of the sequential composition are those of the first component (excluding those ending in ✓) and those of the second concatenated to the first. This is exactly what happens in the operational model. Note we assume that $Q$ will start the execution under a true invariant: this is the case because the syntax of CCSP allows only $P$ terms after the ; operator and all $P$ terms have no invariant.

- $f[P]$: this case is trivial as the same renaming function is applied both to the failures of the semantic model and to the transitions in the operational model.

- $P \setminus A$: consider a trace $t \in \mathcal{RF}(P, r, I)$. It is easy to see that if $t$ is a trace of $R(P, r, I)$ then $t \setminus A$ is a trace of $\mathcal{RF}(P \setminus A, r, I)$ as all actions in $A$ have become $\tau$

actions. Moreover, since $(t, F \cup A)$ is a refusal of $\mathcal{RF}(P, r, I)$, no actions in $A$ are enabled after $t$, therefore no $\tau$ action is introduced from the state after $t$, which therefore remains a stable state and refuses all the actions in $F$ that have not been hidden.

Finally, we prove the result for recursion. The proof follows very closely the proof for the congruence theorem for CSP as described in [Ros98, Chapter 9]. The main difference with our proof is that we have to rearrange the original one in order to consider tuples (CCSP process, starting region, starting invariant) as our basic object. These are the states of the labelled transition system of the operational semantics and also the arguments of the semantic function $\mathcal{RF}$. Once this correspondence is defined, the proof follows easily.

So far in this proof, we have ignored that a given CCSP term $P$ might contain free process identifiers. Denote the set of identifiers by $Ide$, and we say that an identifier $Z$ is free if it is not bound by a recursive definition $\mathsf{rec}\, Z.P$, for some $P$. We need a semantics that keeps track of these identifiers and introduce *environments*, that is, functions that associate a tuple of set of failures to an identifier (a set of failures for each possible starting condition). Formally, an environment is a function defined as:

$$Env : Ide \to \mathsf{RFailures}^{nm}$$

We obtain the following semantic function for region failures, the equivalent of the function *RegionFailures*, but with environments:

$$S_{\mathcal{RF}} : \mathbf{CCSP} \to Env \to \mathsf{RFailures}^{nm}$$

This function associates a CCSP term and a given environment to its corresponding semantics. We define the equivalent operator in the operational semantics, mapping identifiers to CCSP terms with no identifiers (denoted by $\overline{\mathbf{CCSP}}$):

$$\xi : Ide \to \overline{\mathbf{CCSP}}$$

Congruence between the semantic models with environments is proved by showing that for all substitutions $\xi$ and all CCSP terms $P$, the following holds:

$$F(subst(P, \xi)) = S_{\mathcal{RF}}(P)\overline{\xi} \tag{5.9}$$

where $subst(P, \xi)$ is the term obtained by replacing each free identifier $Z$ appearing in $P$ by $\xi(Z)$, $\overline{\xi}(P) = F(\xi(P))$ and $F(P)$ denotes the tuple of sets of failures, each set corresponding to the failures of the region automaton $R(P, r, \phi)$, for each region $r$ and invariant $\phi$.

The proof of congruence that we have given above for all operators except recursion still holds in this case; the same arguments we have used still apply if we use this new semantic function instead, but we preferred to keep the notation simple where possible.

For recursion, we have to prove that $F(subst(\mathsf{rec}\, Z.P, \xi))$ is the least fixed point for the function $\Upsilon : \mathsf{RFailures}^{nm} \to \mathsf{RFailures}^{mn}$ defined as $\Upsilon(\alpha) = S_{\mathcal{RF}}(P)\overline{\xi}[\alpha/Z]$. The fixed point of this function gives the semantics of recursion. For any $\xi$, let $\mathsf{rec}\, Z.P'$ be the term where all free identifiers have been substituted according to $\xi$, so that $subst(\mathsf{rec}\, Z.P, \xi) = \mathsf{rec}\, Z.P'$, since $Z$ is not free in $P$.

Then:

$$
\begin{aligned}
&\text{(a)} && F(\mathit{subst}(\mathsf{rec}\, Z.P, \xi)) = F(\mathsf{rec}\, Z.P') \\
&\text{(b)} && = F(\mathit{subst}(P, \xi[\mathsf{rec}\, Z.P'/Z])) \\
&\text{(c)} && = F(\mathit{subst}(P, \xi[P'/Z])) \\
&\text{(d)} && = S_{\mathcal{RF}}(P)\overline{\xi[P'/Z]} \\
&\text{(e)} && = S_{\mathcal{RF}}(P)\overline{\xi}[F(P')/Z] \\
&\text{(f)} && = S_{\mathcal{RF}}(P)\overline{\xi}[F(\mathsf{rec}\, Z.P')/Z] \\
&\text{(g)} && = S_{\mathcal{RF}}(P)\overline{\xi}[F(\mathit{subst}(\mathsf{rec}\, Z.P, \xi))/Z] \\
&\text{(h)} && = \Upsilon(F(\mathit{subst}(\mathsf{rec}\, Z.P, \xi)))
\end{aligned}
$$

where (a) and (b) follow from the definition of $P'$, (c) from the fact the the semantics of $\mathsf{rec}\, Z.P'$ is the same as $P'$, (d) and (e) by applying the definition of $\overline{\xi}$, (f) by binding $Z$ to $P'$, (g) by applying the equality of (a) again, and, finally, (h) from the definition of $\Upsilon$. This proves that $F(\mathit{subst}(\mathsf{rec}\, Z.P))$ is a fixed point for $\Upsilon$.

We still have to prove that it is the *least* fixed point. We do this by considering each component of $S_{\mathcal{RF}}(P)$; given a tuple of sets of failures $\alpha$, we denote the component corresponding to the initial region $r$ and invariant $\phi$ by $\alpha_{r,\phi}$. We show that for all regions $r$ and invariants $\phi$, if a failure $f = (t, F)$ is in $\mathcal{F}(\mathit{subst}(\mathsf{rec}\, Z.P, \xi), r, \phi)$, then $f \in \Upsilon^n(DIV)_{r,\phi}$ for some $n$, as $DIV$ is the minimal element in the failures model.

Consider such failure $(t, F)$, then there must be some $t' \in (\overline{\Sigma}^{\checkmark})^*$ such that $t = t' \setminus \{\tau\}$ ($t$ with its $\tau$ actions removed) and $R(\mathit{subst}(\mathsf{rec}\, Z.P, \xi), r, \phi) \xrightarrow{t'} Q$, with $Q$ some other state of the automaton (the region and invariant are not relevant). Let $N = \#t$, then the recursion cannot be unfolded more than $N$ times during the derivation of $(t, F)$, since the process is guarded. Define the following processes:

$$
\begin{array}{rcll}
Z_n & = & P'[Z_{n+1}/Z] & n \in \mathbb{N} \\
Y_n & = & P'[Y_{n+1}/Z] & n \in \{0..N-1\} \\
Y_N & = & DIV &
\end{array}
$$

Clearly, $Z_0$ has the same operational semantics as $\mathit{subst}(\mathsf{rec}\, Z.P, \xi)$; it just keeps track of the number of times recursion has been unfolded. $Y_n$ behaves like $Z_n$ for the first $N-1$ unfoldings of recursion, then it diverges. Since $\#t = N$, $(t, F)$ must also be a behaviour of $Y_0$, that is $(t, F) \in \mathcal{F}(Y_0, r, \phi)$. We also have $S_{\mathcal{RF}}(Y_N) = \{(\langle\rangle, \{\})\}^{nm}$, $S_{\mathcal{RF}}(Y_{N-1}) = \Upsilon(\{(\langle\rangle, \{\})\}^{nm})$, ... and $S_{\mathcal{RF}}(Y_0) = \Upsilon^N(\{(\langle\rangle, \{\})\}^{nm})$, because $\mathcal{F}(\mathit{subst}(P, \xi[Q/Z])) = S_{\mathcal{RF}}(P\overline{\xi}[\mathcal{F}(Q)/Z]$ for every closed term $Q$. It follows that $(t, F) \in \Upsilon^N(\{(\langle\rangle), \{\}\})$, which is what we had to prove. □

This result is, of course, also true for trace semantics, since stable failure semantics is strictly stronger than region failure semantics.

## 5.5 Model Checking Clocked CSP

So far, we have given the theoretical foundations for Clocked CSP, guided by the desire to formulate the foundations of automatic verification of timed properties for CSP processes extended with time. CCSP cannot be model checked directly and further work is necessary to achieve this. In this section we outline how this can be done and describe possible future research.

### 5.5.1 The Main Idea

We would like to be able to automatically verify the refinement relations described above by using the FDR2 model checker for CSP [For93]. This is possible for finite-state CCSP terms and systems thanks to the congruence result. We also consider only processes without external clocks. In this section we investigate the usefulness of this result, that is, what kind of properties can be verified by using the region refinement.

Our approach is limited, the main reason being that our semantics makes explicit references to clock names: we argued that this was necessary in order to obtain compositionality and decidability, but this turns out to limit the power of region refinement. Consider the following example:

$$\begin{aligned} P_1 &= \{\!|x|\!\}(a, x = 1) \rightarrow \{\!|x|\!\}(a, x = 1) \rightarrow \mathit{SKIP} \\ P_2 &= \{\!|x|\!\}(a, x = 1) \rightarrow (a, x = 2) \rightarrow \mathit{SKIP} \end{aligned}$$

$P_1$ and $P_2$ would be distinguished as $P_1$ resets clock $x$ twice, while $P_2$ does it only once. Clearly, we would like to identify them (they are timed bisimilar).

A first step to overcome this problem is to define refinement "up to" clock renaming. Next, we need to find a way to use the refinement in a meaningful way. The main idea is to distinguish between functional properties (dependent only on actions) and timed properties (dependent on the clocks). Then we can highlight only the timed properties we are interested in verifying by using auxiliary processes, and use refinement in such a way that a process $P$ refines a process $Q$ if it preserves some timed properties and if it functionally refines it in the traditional sense. This is similar to the idea of timewise refinement [Sch97], where system analysis distinguishes between timed and functional aspects.

To explain this, let us consider the region traces model and note that, if we hide all clock actions (either resets or delays) from traces, we would be able to verify functional (untimed) refinement. If we hide only those clock actions pertaining to a given subset of clocks, then we would be able to verify the refinement with respect to only that subset of clocks, possibly the subset that describes the timed behaviour that we are interested in.

The hiding of clocks can be easily defined for $\mathcal{RT}$ and $\mathcal{RF}$ in the same way as we have defined hiding and renaming: if we want to hide a set of clocks $X$, then all clock reset actions $Y$ are renamed to $Y \setminus X$ if $Y \setminus X \neq \emptyset$ and hidden otherwise, while a delay action $\delta_Y$ is renamed to $\delta_{Y \setminus X}$ if $Y \setminus X \neq \emptyset$ and hidden otherwise. The hiding of clocks can be defined only as a top level operator for processes with no external clocks, as otherwise most states would become unstable and different components would not be able to synchronise. This is why we talk about refinement between two processes $P$ and $Q$ with respect to a certain set of clocks $X$, meaning refinement in which all clocks not in $X$ are hidden for each of the processes.

This approach is made clear by the following example.

### 5.5.2 A Model Checking Example

We give an example (also used in [OW02]) to illustrate model checking with the region semantics approach. Assume that we want to check if a process respects the bounded invariance property $\Box(a \Rightarrow \Box_I \neg b)$ where $I = [0, n]$ is a closed interval starting from 0. This means that, whenever an action $a$ is performed, the process cannot execute a $b$ for $n$ successive time units. This is a safety property that can be specified as a trace property. In particular, note that with the region automata approach this property can be checked and enforced with the use of a single clock, whose value is reset every time an $a$ action is

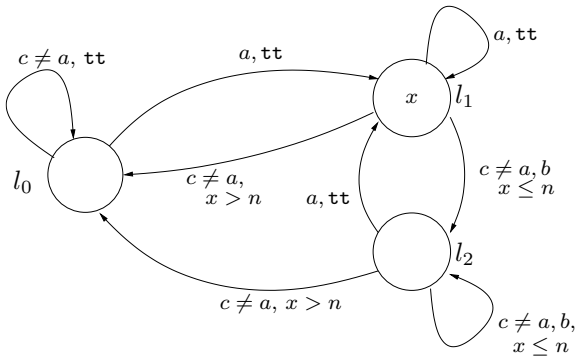



Figure 5.3: The timed automaton corresponding to the bounded invariance property.

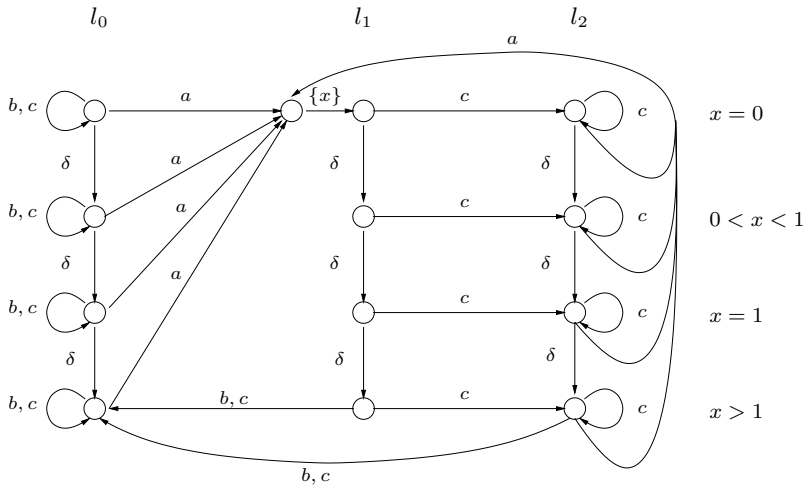



Figure 5.4: The region automaton corresponding to the bounded invariance property.

performed, while $b$ actions are forbidden until the appropriate number of delay actions $\delta$ have occurred.

The most non-deterministic process that respects this property is:

$$\begin{aligned} S &= (a, \mathtt{tt}) \to \{\!|x|\!\} S_1 \,\Box\, \left(\Box_{c \neq a} (c, \mathtt{tt}) \to S\right) \\ S_1 &= (a, \mathtt{tt}) \to \{\!|x|\!\} S_1 \,\Box\, \left(\Box_{c \neq a, b} (c, x \leq n) \to S_1\right) \,\Box\, \left(\Box_{c \neq a} (c, x > n) \to S\right) \end{aligned}$$

$S$ will serve as the specification that must be refined by processes that respect the bounded invariance property.

The timed automaton corresponding to this process is pictured in Figure 5.3. Every time an $a$ action is performed, the system moves to the top-right location, where clock $x$ is reset. The control stays on the right side of the system until the value of $x$ exceeds $n$ and an action which is not $a$ is performed. No $b$ action can be executed while $x \leq n$, that is, within $n$ time units after the last $a$. This is reflected in the corresponding region automaton for $n = 1$ (Figure 5.4), where no $b$ action is allowed until enough $\delta$ actions have occurred.

How do we deal with the fact that an implementation process $P$ might not reset a clock each time an $a$ action is performed and can use more clocks? The idea is that we

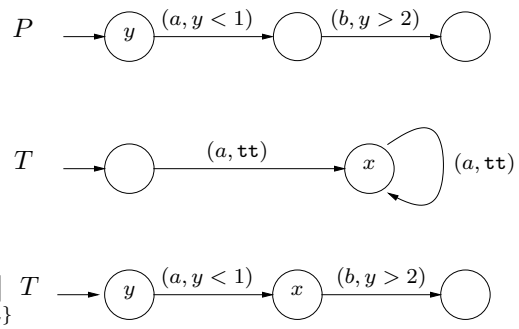



Figure 5.5: A possible implementation process $P$ and its parallel composition with the test automaton $T$ corresponding to the bounded invariance property.

define a *test* process that uses clock $x$, which is external to $P$, and resets it each time an $a$ is performed. This is achieved through the parallel composition of $T$ and $P$ synchronising on the interface alphabet $\{a\}$, that is, $P \underset{\{a\}}{||} T$. The test process $T$ is defined as follows (mid timed automaton of Figure 5.5):

$$T = (a, \mathtt{tt}) \rightarrow \{\!|x|\!\}\, T$$

In order to check whether $P$ meets the bounded invariance property, we need to check the following refinement, with respect to clock $x$ only:

$$S \sqsubseteq_{\mathcal{RT}} \quad P \underset{\{a\}}{||} T$$

This works because the specification $S$ does not allow the execution of any $b$ before the value of clock $x$ is greater than $n$, and the passage of time is recorded by delay actions. Clock hiding is defined as a combination of hiding and renaming on the set of traces (and in the operational model): if we hide a clock $x$, then we need to hide all the reset or delay actions that involve it (or rename the action if other clocks are reset or change region at the same time). This is why we encode the information as to which clocks cause a region change in delay actions instead of having a single generic delay action that would not permit the hiding of clocks.

Finally, our approach admits successive refinements: considering the example above, if the following two refinements hold:

$$S \sqsubseteq_{\mathcal{RT}} \quad P_1 \underset{\{a\}}{||} T \sqsubseteq_{\mathcal{RT}} \quad P_2 \underset{\{a\}}{||} T$$

then both $P_1$ and $P_2$ respect the bounded invariance property and, in addition, there is functional refinement between $P_1$ and $P_2$.

As a possible implementation process, consider the process $P = \{\!|y|\!\}(a, y < 1) \rightarrow (b, y > 2) \rightarrow STOP$. Both $P$ and its composition with $T$ are pictured in Figure 5.5. Clock $x$ is reset immediately after $a$ is executed and this can happen only while $y = 0$ or $0 < y < 1$. In both cases, since $b$ is performed only when $y > 2$, $x$ will be greater than 1 and the bounded invariance is respected.

The region automaton corresponding to $P \underset{\{a\}}{||} T$ is shown in Figure 5.6; it is possible to appreciate how quickly the size of the region automaton grows with the addition of one

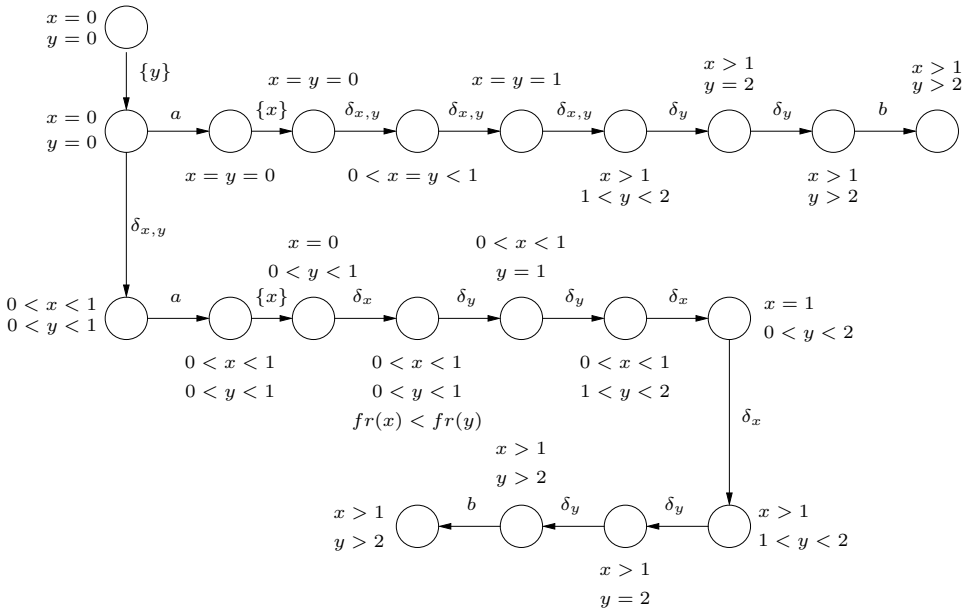



Figure 5.6: The region automaton corresponding to the process $P \underset{\{a\}}{||} T$ of Figure 5.5.

extra clock, with maximal constant 2. The two maximal traces of such automaton are:

$$\langle \{y\}\ a\ \{x\}\ \delta_{x,y}\ \delta_{x,y}\ \delta_{x,y}\ \delta_y\ \delta_y\ b \rangle$$
$$\langle \{y\}\ \delta_{x,y}\ a\{x\}\ \delta_x\ \delta_y\ \delta_y\ \delta_x\ \delta_x\ \delta_y\ \delta_y\ b \rangle$$

By hiding all the reset and delay actions involving clock $y$ we get the following trace

$$\langle a\ \{x\}\ \delta_x\ \delta_x\ \delta_x\ b \rangle$$

which is one of the traces of the specification, as expected, and therefore process $P$ refines the implementation and respects the bounded invariance property.

**Beyond safety properties: using failures.** The traces model suffices to verify safety properties, since one can consider all the traces that do not contain undesired behaviour. If we want to verify liveness properties, we need to use the failures model. The idea is the same as described above, that is, we verify refinement with respect to a subset of clocks only. It is possible to verify properties such as strong bounded response using a similar idea. For instance, the strong bounded response property, requiring an action $b$ to be executed within $n$ time units from the last $a$ action, could be verified with region failures. This is not a safety property, since it requires the execution of an action. After $n$ time units from the execution of an $a$, the specification refuses to let time elapse (i.e. refuses all actions in $\Delta$), until a $b$ is performed.

**Using FDR2.** One could attempt the above examples with FDR2, but the complexity of manually translating CCSP processes into the equivalent CSP processes makes this task error prone and tedious. Given the complexity of building a region automaton, and its resulting size, it would be desirable to make the process above automatic, for which more research is needed. We tested our ideas with small toy examples in the following way:

firstly, we defined small CCSP processes, then we translated them into the equivalent timed automaton and, from here, into the resulting region automaton. At this point we manually specified CSP processes equivalent to the region automata and verified refinements between them with FDR2.

### 5.5.3 Considerations

We have shown how the semantics based on regions can be used to verify refinement relations between processes; by using appropriate additional processes and by checking refinement only with respect to some clocks, we were able to verify both functional properties of processes and also some timed properties, though, admittedly, a subclass of the latter. One advantage of our approach is that we have been able to obtain a decidable semantics that also allows chains of refinements.

It is worth pointing out that here we are not concerned with efficiency: region automata can be exponential in size, thus making the checking of refinement exponential. Instead, the main objective was to find a decidable semantics that could be used to automatically verify certain timed properties of processes through refinement. It would also be interesting to find out whether more efficient techniques, e.g. zones, can be applied to improve our approach. The fact that we verify refinement only with respect to a subset of clocks could be exploited too.

Another drawback is the fact that we have to manually write process specifications for each timed property, as we did in the example above. A possible line of research would be to automatically generate such processes from some appropriate timed logic. Something similar to this was proposed in [ABL98, ABBL03], where properties of a safety model logic are automatically verified through testing by synthesising a test automaton to run in parallel, thus reducing the verification problem to reachability.

## 5.6 Discussion

In this part, we have described a proposal for a timed extension to CSP, called Clocked CSP. We have defined its semantics and the corresponding refinement relations, demonstrating how one could use the model checker FDR2 to verify such relations, and also certain timed properties of systems. Future work will include improving the complexity of model checking, and investigating whether it is possible to use known efficient techniques for timed automata in our case. It would also be interesting to define a temporal logic that could be verified using our technique.

# Chapter 6

# Conclusions

In this part of the thesis we have proposed a timed version of CSP, called Clocked CSP (CCSP), inspired by and modelling timed automata, achieved by the addition of clocks and constructs to handle them.

The language is given operational semantics in terms of timed automata. In Chapter 4 we presented the algebra and analysed the restrictions that are needed in order to achieve compositionality: CCSP differs from standard CSP and also from Timed CSP since it handles shared variables, that is, clocks. This creates problems when considering some operators (external choice and interface parallel) that allow components to alter the value of clocks independently from the other components. We solved this problems by imposing syntactic restrictions: external choice must be guarded by visible actions, while parallel processes can only manipulate disjoint sets of clocks.

In Chapter 5 trace and failure semantics were defined for CCSP, by using the region graph as the operational model we had in mind. In this way, we have obtained a discrete denotational semantics that can be defined compositionally and that is congruent with the operational model. Congruence allows us to use existing techniques for the automatic verification of trace and failure equivalence/refinement.

Finally, we showed how our approach could be used for the verification of timed properties of CCSP processes and timed automata.

## 6.1 Future Work

The last observation made above, that is, the verification of timed properties of CCSP processes, could be the starting point for further research: in this thesis we have only presented a simple manually constructed example of how the problem of verification of safety formula could be approached. It would be worthwhile to find an automatic procedure to generate CCSP processes for temporal formulae, following an approach similar to that of [ABL98] for testing of timed automata. Moreover, it would be necessary to find a suitable logic that can be used to verify properties preserved in the trace or failure models.

In order to make our approach feasible in practice, it is necessary to tackle the state space explosion; a natural approach would be to check whether it possible to extend existing symbolic approaches for timed automata (e.g. zones [HNSY92]) to our setting.

In the development of CCSP we took a rather pragmatic approach, our goal being the automata-theoretic verification of trace-based relations. Therefore, the analysis of the equational laws that our language respects was not one of our concerns. This constitutes another possible direction to further develop our proposal of a CSP-based process algebra for timed automata.

# Part II

# Stochastic Systems

# Chapter 7

# Stochastic Systems

In this chapter we give an overview of models for probabilistic systems. Probabilities arise in the modelling of concurrent systems in many ways: for example, probabilities can model unreliable behaviour, they can be used in distributed algorithms as a way to break symmetry, or, in the context of real time systems, they can be used to describe the distributions corresponding to time delays.

As in the case of real-time systems, extensive research has been done aimed at extending virtually every model for classical concurrent systems with probabilities: we find probabilistic extensions of labelled transition system [LS89b, Seg95b], CSP [Low95, Sei92, Nor97, MMSS95], CCS [GJS90, LS91b], and so on.

We again focus on the analysis of automata-based and process-algebraic models: firstly, we describe the approaches based on discrete probabilities, then we analyse the approaches that consider continuous probability distributions and continuous time and/or state spaces. The main feature shared by these two approaches, on which we are going to focus, is the role that non-determinism will have and how it relates to the probabilistic behaviour. Though these two aspects can be seen as orthogonal, the way they interact in different models leads to fundamental distinctions.

## 7.1 Discrete Probabilities

#### Probabilistic Choice and Non-deterministic Choice

Traditional models are usually extended with probabilities by generalising the notion of transitions, whose target is not a single state any more, but a probability distribution over the set of states; we call such transitions probabilistic transitions.

A classification of possible probabilistic systems was proposed in [GSST90], where the authors distinguish three types of probabilistic processes: reactive, generative and stratified. In the reactive model [LS89a], each label enables a transition whose target is a distributions over the set of states, thus leading to a restricted non-determinism, as the choice of action determines the (probabilistic) transition. In the generative model, e.g. [GSS95], probabilities replace non-determinism and a single probability distribution over the set of labels and states is enabled from each state. Finally, in the stratified model, each state either enables one labelled transition whose target is another state, or an unlabelled transition whose target is a probability distribution over the set of states. In [GSST90], a process algebra extending CCS with probabilities is introduced and given semantics in terms of each of the three models described above. Under this framework, the model of discrete time Markov chains of probability theory can be seen as an unlabelled generative

system.

Other approaches are possible: for example, one could partition the space of states into non-deterministic states, enabling one or more labelled non-probabilistic transitions, and probabilistic states, enabling one unlabelled probabilistic transition; this is called the alternating model [HJ89, HJ90, Han94], derived from [Var85]. Probabilistic automata [Seg95b] extend labelled transition systems so that each state enables several probabilistic transitions, with no constraints on the labelling.

All of the models above present different degrees of non-determinism: at one end of the spectrum, generative processes present no non-determinism as the only possible choice is probabilistic. Reactive models present "more" non-determinism, even if the choice of action determines the next transitions. The alternating model and probabilistic automata are "fully" non-deterministic as they both present a choice of actions to the environment (external choice), and, for a given action, several different transitions are possible (pure non-determinism). This is the type of non-determinism that we are interested in modelling, as the most suitable to represent a process algebra.

All of these models are endowed with appropriate equivalence relations. In particular, bisimulation relations were extended to the probabilistic case in [LS89a, LS91a] for a reactive model. Here, bisimulation was defined as an equivalence relation on states such that two states are bisimilar if, for each label, they enable equivalent probabilistic transitions (that is, distributions), where two distributions are considered as equivalent if they assign the same probability to each equivalent class. States are then shown to be bisimilar if and only if they satisfy the same properties of probabilistic modal logic, and if they can be distinguished through an appropriate notion of probabilistic testing. In [LS92a] a process algebra is proposed, together with a complete axiomatisation with respect to this notion of bisimulation. Based on this idea, bisimulation was extended to other models: a notion of bisimulation was defined in [GJS90] for a probabilistic variant of CCS based on a generative model, where the choice operator of CCS is replaced by probabilistic choice. Strong and weak bisimulations have been defined for probabilistic automata [LS95b], showing that they extend the equivalent notions for the non probabilistic case, if only Dirac distributions (that is, distributions that assign probability 1 to a single state) are allowed, and for the reactive model of [LS89a]. Weak bisimulation has also been defined for the alternating model [PLS00].

In the context of probabilistic and non-deterministic systems, the notion of *schedulers*, or adversaries, was introduced [Var85, FZ88, Han91] to denote the entity that resolves non-determinism. The interaction of a system with a scheduler results in a purely probabilistic system (no non-determinism) that describes a possible run of the system. In a randomised environment, the scheduler itself can choose the next transition in a probabilistic way by combining the possible transitions enabled from each state. The use of randomised schedulers lead to different equivalence relations: for instance, bisimulation for probabilistic automata preserves the logic PCTL only under the control of randomised schedulers [SL95].

The interaction between non-determinism and probabilistic choice often plays a key role: consider for instance the alternating model and probabilistic automata, that is, the two main models that, according to our terminology, are fully non-deterministic. They are equivalent with respect to strong relations (they have the same axiomatisation in the recursion-free setting [BS01]); however, when introducing internal actions and considering weak bisimulations, the axiomatisations are not comparable and, while weak bisimulation for the alternating model [PLS00] and for DTMCs/CTMCs is decidable in polynomial time, it requires exponential time in the case of probabilistic automata under randomised

schedulers [CS02].

In this part of the thesis, we will define a fully non-deterministic model and use the probabilistic automata approach to non-determinism. This is motivated by the fact that we aim to define a CSP-inspired stochastic process algebra, equipped with the different types of choice operators. In addition to this, there are well understood reasons for including non-determinism, such as the need for underspecification or for allowing interactions with the environment.

### Trace Semantics

[Seg95a] extends the trace semantics of labelled transition systems to probabilistic automata; in the probabilistic setting, a trace is represented by a probability distribution over the set of traces, and the trace semantics of an automaton is given by the set of trace distributions obtained for each way non-determinism is resolved. Such semantics induces the notion of trace pre-order, defined as inclusion of the set of trace distributions of two automata. [SV03] shows that the trace distribution pre-order is equivalent to a notion of testing (see [DH84]) for probabilistic automata, where the testing scenario of [LS91a] is extended to probabilistic automata.

Unfortunately, the notion of trace pre-order fails to be compositional with respect to parallel composition. For this reason, the notion of trace pre-congruence is introduced as the coarsest congruence included in the trace distribution pre-order. Such pre-congruence has been shown to coincide with the notion of probabilistic forward bisimulation [LSV03]. To our knowledge, no decidability results are available for the equivalences or pre-orders described above.

### Denotational Models

The approach to trace semantics of [Seg95a] is based on the operational model of probabilistic automata. This is the approach that we will take in this part of the thesis as we believe it is more intuitive and it has shown to be more promising in the discrete case. This approach, however, differs from the classical treatment of CSP, where semantics is given in terms of domain equations. This is the conventional approach of several denotational models extending CSP.

In [Low93b], CSP is extended with a probabilistic operator, and the terms of this language are mapped to an operational model, and, by eliminating non-determinism, a distribution over traces is derived. The denotational semantics of a process is then given in terms of sets of such distributions. This semantics fails to be compositional with respect to several operations and it is therefore unsuitable for a denotational model. This approach is similar to that of [Seg95a] for probabilistic automata described above, and the same compositionality problems surface. Another approach by the same author is that of [Low93a]: in this case both types of choice are replaced by probabilistic variants, thus resulting into a (complex) semantic model with no non-determinism.

[Sei92] proposes a version of CSP where internal choice is replaced by probabilistic choice. A process is given semantics in terms of probability distributions over traces and in terms of conditional probabilities given a trace representing what the environment proposes; this model, however, is unable to represent hiding. [Nor97] presents a probabilistic extension to CSP that retains both internal and external choice and processes are given denotational semantics by means of a metric space; this approach, however, fails to model some operators. Another approach was proposed in [MMSS95], where a probabilistic operator was added to CSP and the resulting language was given semantics by

using the probabilistic power-domain construction of [JP89, Jon89]. Informally, in this model, the semantics of a process is given by a probability distribution over the set of non-probabilistic CSP processes. This approach is not completely satisfactory as it fails to obey several standard laws. Such problems have been addressed in successive works [Mis00, OMW03], that aim to achieve an axiomatisation of a probabilistic calculus using the power-domain construction.

## 7.2 Continuous Systems

More recently, the analysis of probabilistic systems has been extended to systems with continuous state spaces or a continuous time representation. Such models can be used, for instance, to represent systems whose progress depends on continuously distributed real-time delays.

### Markovian Models

In real-time modelling, one can associate a real-valued delay to each transition; likewise, in a stochastic setting, one can associate a probability distribution to model the continuously distributed delays associated to actions. Such systems can model, for example, queueing systems or communication networks.

Using this approach, only exponential distributions were initially considered since they are tractable thanks to their memoryless property. In this model, a transition is annotated with both an action $a$ and a rate $\lambda$: the probability to perform $a$ before time $t$ is equal to $1 - e^{-\lambda t}$. The memoryless property states that residual probability distribution of an action after time $t$ is the same as the initial distribution, that is, the probability that an action is performed by time $t + t'$, given that time $t$ has already elapsed, is the same as the probability of the action being performed before time $t'$ from the initial time point. The models that restrict to exponential distributions are also called *Markovian*.

The typical operational model for the exponential distribution is that of Continuous Time Markov Chains (CTMCs): this is a discrete state transition system where the passage of time is represented implicitly by the addition of rates to transitions. Extensive research has been done on CTMCs: there have been studies employing standard performance analysis techniques, analysing steady state and transient probabilities. Equivalence relations extending the bisimulation of [LS91a] have been defined, leading to notions equivalent to the lumpability of queueing theory. Strong and weak bisimulation and simulation for CTMCs have been related to the logic CSL [ASSB96, BKH99]. A discussion of several bisimulation and simulation relations for CTMCs (and DTMCs) can be found in [BKHW03].

CTMCs have proved successful as a model for process algebras for performance evaluation. PEPA [Hil96] was one of the first such proposals. In this model non-determinism is replaced by what is called a *race condition*: given a choice between two actions with two different rates, the choice is resolved in favour of the first action to be executed. Interleaving is modelled neatly thanks to the memoryless property: given two action to be executed in parallel, the distribution of the second action to be executed is independent of the amount of time that the first action took to be performed. Other Markovian process algebra have been proposed [BDG94, GHR92]. See [Hil94] for a discussion of different approaches to synchronisation.

Pure non-determinism has also been considered for Markovian models: the model of Interactive Markov Chains [Her02] introduces non-determinism by separating actions and

delays, and allowing a non-deterministic choice operator to be defined on actions.

### Models for General Distributions

Although exponential distributions can model several types of stochastic behaviour and are often preferred since they are easier to handle, they are not always sufficient and general distributions are needed. Unfortunately, it is not possible to model general distributions in the same way as exponential distributions, that is, by specifying the distributions associated with each transition, as several desirable properties would be lost. Consider, in particular, the *expansion law*, which is important as it allows to rewrite parallel composition in terms of the choice operators: for example, in traditional process algebras, using the CSP notation, the process $a \rightarrow STOP \,|||\, b \rightarrow STOP$ can be rewritten as $a \rightarrow b \rightarrow STOP \,\Box\, b \rightarrow a \rightarrow STOP$. This law holds in the case of exponential distributions thanks to its memoryless property; indeed, this is the way that process algebras like PEPA model parallelism. This law does not hold for general distributions as, once an action is executed after some delay, the delay associated to the second action is not independent of the time elapsed. Refer to [KD01] for an introduction to the issues related to general distributions in process algebras.

The key idea to solve this problem is to keep track of residual times after the execution of actions by means of constructs such as clocks. Generalised Semi-Markov Processes (GSMPs) [Whi80] are a model for general distributions that takes this approach by keeping track of stochastically triggered events, whose residual lifetime is recorded across changes of states. GSMPs do not model non-determinism, which is replaced by race conditions.

Interactive GSMPs [BG02] extend GSMPs with non-determinism by using clocks and by distinguishing different types of transitions (actions, clock start and clock termination transitions); actions denote the interactive behaviour on which non-determinism is possible. A process algebra for general distributions is introduced in [Bra02], with a semantic model based on IGSMPs.

Stochastic automata [D'A99] incorporate ideas from timed automata and GSMPs and can also model non-determinism. Stochastic automata are labelled transition systems augmented with clocks, whose values are set according to some distribution and then decrease at the same rate as real time. Actions are triggered when the value of clocks reaches zero. [D'A99] presents the stochastic process algebra *Spades*, modelled on stochastic automata, together with a sound axiomatisation with respect to several notions of bisimulation.

Another approach is that of [Str93], presenting a process algebra that uses timers and an infinite space semantic model.

A unified approach for stochastic process algebra is studied in [BD04], where a common framework for stochastic process algebras is analysed with respect to bisimulation equivalences.

The models described above are high-level, as they use notions like clocks for the handling of time and of probability distributions. Similarly to timed automata, they are given semantics in terms of infinite-state labelled transition systems, that must be capable of representing both probabilistic and non-deterministic behaviour. The state space is infinite because it must record both the symbolic locations and the values of clocks. This emphasises the need for models that extend the discrete systems of Section 7.1 to the continuous domain. Several such models have been introduced to give concrete semantics to the process algebras described above. For example, [D'A99] presents an alternating model on which probabilistic bisimulation is defined. [Bra02] introduces another probabilistic model with different types of transitions (action, time or probabilistic transition).

A notion of weak bisimulation is defined as the concatenation of several action or time transitions followed by a probabilistic transition.

All of these models are studied from the point of view of bisimulation relations, that are defined point-wise. Since we are interested in defining a trace semantics, we need to consider the behaviour of a system globally and across several steps of executions. This introduces several new problems when dealing with the measure-theoretic issues caused by the use of continuous state spaces.

#### Measure Theory and Models for Continuous State Spaces

The importance of measure theory in the analysis of stochastic concurrent systems has been highlighted [Pan97]. This has led to the definition of the model of Labelled Markov Processes (LMPs) [DGP02], that can be seen as a reactive model for continuous state spaces.

There are two main ideas: the use of analytic spaces as state spaces, that is, spaces that have some extra topological properties (e.g. the reals), thus requiring some structure to the model instead of allowing arbitrary measure spaces, and the use of Markov Kernels as transition functions for each label: given a set of states expressed as an analytic space $(Q, \mathcal{F}_Q)$, a Markov Kernel is a function

$$\kappa : Q \times \mathcal{F}_Q \to [0, 1]$$

such that

1. for all $q \in Q$, $\kappa(q, \cdot)$ is a probability measure, and
2. for all $X \in \mathcal{F}_Q$, $\kappa(\cdot, X)$ is a measurable function.

While the first condition is just the probabilistic extension of a transition relation, the second is essential to guarantee the measurability of executions of a system. LMPs define a different Markov kernel for each action, and therefore do not model pure non-determinism as the choice of action determines the next probabilistic transition. This makes LMPs unsuitable as a semantic model for a stochastic process algebra. The addition of non-determinism adds new measurability issues; this will be the topic of the next few chapters.

Bisimulation has been defined for LMPs and given a characterisation in terms of a very simple logic without neither negation nor infinite conjunction [DGP02, DEP98], made possible by the absence of non-determinism. [DGJP99] proposes a metric for LMPs that equates bisimilar processes. Recently, the requirement that the state spaces are analytic has been relaxed by proposing an alternative notion of bisimulation [DDP04]. LMPs are subject of extensive research, covering topics like approximation [DGJP00] and testing [vBMOW04].

Probabilistic extensions to timed automata have also been presented, both by adding discrete probabilistic choice [KNSS02] and by considering continuously distributed resets for clocks [KNSS00]. This latter work introduces continuous time probabilistic automata and studies the verification of PTCTL formulae. Since this model includes non-determinism, measurability problems arise and the authors restrict to the set of executions that are tractable from a mathematical point of view, but neither study the class such of executions nor the schedulers generating them.

## 7.3 Our Aims and Contribution

While our aim is to define a trace-based process algebra for real-time stochastic systems, questions concerning the study of the behaviour of probabilistic systems with continuous

state spaces and non-determinism across several steps of executions have led us to shift our focus towards the operational model. Therefore, the main topic and the main contributions of this part of the thesis concern the study of these issues. We introduce the model of stochastic transition systems, and lay the foundation for the study of trace semantics.

We further highlight the importance of measure theory, and we study the restrictions needed regarding the way non-determinism is resolved in order to rule out executions that are not tractable from a mathematical point of view; this is done by identifying the correct class of schedulers in Chapter 8. Based on these restrictions, we are able to define trace semantics in Chapter 9. Again, the use of continuous state space introduces new subtleties, and a more general definition of bisimulation relations is required in order for them to be sound with respect to trace semantics.

We finally introduce a simple stochastic process algebra in Chapter 10: this language serves as a motivation for the theoretical developments on stochastic transition systems, and we give it semantics in terms of traces and bisimulation, by extending the corresponding notions to the algebra. The trace semantics is not compositional with respect to the operators (thus extending the result of [Low93b, Seg95a] to the continuous setting). Further research can be aimed at developing equivalences and pre-orders that satisfy the expected properties based on the framework that we have developed.

# Chapter 8

# Stochastic Transition Systems

## 8.1 Introduction

In this chapter, we introduce a model for systems with continuous state spaces presenting both non-deterministic and probabilistic behaviour. We call this model *stochastic transition systems*. It can be seen as a generalisation of probabilistic automata [Seg95b] to a continuous setting: we allow continuous state spaces and sets of labels and also uncountable branching. The model also generalises labelled Markov processes [DGP02] by the addition of "pure" non-determinism.

### The Model

A stochastic transition system is an extension of labelled transition systems to arbitrary distributions: the target of a transition is a probability measure on the set of states, that for this reason has to be equipped with a $\sigma$-algebra. Likewise, the set of actions is continuous and has a corresponding $\sigma$-algebra. We extend several concepts, familiar in the context of probabilistic systems, like that of combined transitions and schedulers, to our setting. However, the construction of a probability measure on the set of executions cannot be extended in a straightforward way because we have to take into account the issues of measurability that arise from the use of continuous state spaces.

### The Problem of Measurability

Since we are interested in defining a trace semantics for our systems, we study the behaviour of a run of a system over several steps of executions. This introduces the problem of measurability: if non-determinism is resolved in an arbitrary way, then the resulting run of the system could be intractable from the mathematical point of view, in the way that it is not possible to define a measure on executions. We want to rule out such "bad" behaviour and restrict to the runs that we can deal with. As in the discrete case, we use the notion of scheduler as the entity that resolves non-determinism. We have to restrict the power of schedulers in order to generate all and only those executions that are tractable, and identify the class of such schedulers.

Consider for example the probability of reaching some set of states $X$ after two steps of computation. Informally, such probability is given by the probability of reaching any state $q$ from the initial state $\overline{q}$, multiplied by the probability of reaching $X$ from $q$. Define a function $f$ that, given a state $q$, returns the probability of reaching $X$ from $q$, and define $\mu$ to be the probability measure leaving the initial state. Then the probability of reaching $X$ after two steps of computation is given by $\int_Q f(q)\mu(dq)$. This integral is defined only when

function $f$ is measurable. The function $f$ is determined by a scheduler, that decides which transition is enabled from each state $q$. We therefore want to rule out those schedulers that generate non-measurable functions $f$.

We show that this class of schedulers has the important property to be preserved under parallel composition: a "good" run of the parallel composition between two systems can be obtained from "good" runs of the components.

The problem of measurability of executions was identified in [KNSS00], where the power of schedulers was restricted so that only tractable executions are generated, but a theoretical treatment of the properties of such schedulers was not included. Labelled Markov processes also address this problem; however, since no no-determinism is present, measurability is easily achieved by imposing measurability conditions on the transition relation.

By restricting to measurable schedulers, we can define a probability measure on executions and also define weak transitions in a straightforward manner, extending the equivalent notions for probabilistic automata. These results lay the foundations on which the equivalence relations of Chapter 9 are defined.

**Chapter Outline**

Section 8.2 introduces the model of stochastic transition systems together with the relevant notions. In Section 8.3, the notion of a run is formalised and the problem of measurability is addressed by identifying the class of measurable schedulers that generate all and only the "good" runs. Section 8.4 introduces parallel composition for stochastic transition systems and shows that measurability properties are preserved by this operator.

This and the next chapter is heavily based on the definitions and results on measure theory of Section 2.3.

## 8.2 The Model

In this section we introduce our model, called *stochastic transition systems*, which features both non-deterministic and probabilistic behaviour. The model can be seen as an extension of probabilistic automata [Seg95b] to continuous state and label spaces and to continuous probability measures. Stochastic transition systems are fully non-deterministic, and thus also generalise labelled Markov processes [DEP98].

In this section we introduce the fundamental concepts of our continuous model, most of which are an adaptation of [Seg95b] to the continuous setting.

### 8.2.1 Stochastic Transition Systems

We define stochastic transition systems as an extension of labelled transition systems to a continuous setting with general probabilistic executions.

**Definition 8.1 (Stochastic transition systems).** A stochastic transition system (sts) $S$ is a tuple

$$S = ((Q, \mathcal{F}_Q), \overline{q}, (L, \mathcal{F}_L), \rightarrow) \tag{8.1}$$

where

- $(Q, \mathcal{F}_Q)$ is the analytic space of states;
- $\overline{q} \in Q$ is the initial state;

- $(L, \mathcal{F}_L)$ is the analytic space of labels;
- $\rightarrow \subseteq Q \times L \times \mathcal{D}(Q, \mathcal{F}_Q)$ is the set of probabilistic transitions.

We say that a transition $(q, a, \mu)$ is labelled by $a$ and enabled from $q$, and denote it by $q \xrightarrow{a} \mu$; transitions are ranged over by $t$. We denote the elements of an STS $S$ by $Q$, $\mathcal{F}_Q$, $\overline{q}$, $L$, $\mathcal{F}_L$ and $\rightarrow$ and we propagate indices when necessary; thus, the elements of $S_i$ are $Q_i$, $\mathcal{F}_{Q_i}$, $\overline{q}_i$, $L_i$, $\mathcal{F}_{L_i}$ and $\rightarrow_i$ . We denote the set of possible transitions by $\mathcal{T} = Q \times L \times \mathcal{D}(Q, \mathcal{F}_Q)$ and define a $\sigma$-algebra on it as the product of the $\sigma$-algebras of the components, that is, $\mathcal{F}_\mathcal{T} = \mathcal{F}_Q \otimes \mathcal{F}_L \otimes \mathcal{F}_{\mathcal{D}(Q,\mathcal{F}_Q)}$. The set of transitions enabled from a state $q$ is denoted by $\mathcal{T}(q) = \{(q', a, \mu) \in \rightarrow \mid q = q'\}$.

The space of labels is partitioned into two measurable sets, $L^e$ and $L^\tau$, of external and internal actions, respectively. Internal actions denote the computation of a system that is not visible from the outside and from which we abstract when defining the notion of weak transitions and weak bisimulation. Assume, for simplicity, that $L^\tau$ is composed of the single transition $\tau$, that is, $L^\tau = \{\tau\}$.

**Remark 8.1.** Probabilistic automata correspond to the subset of stochastic transition systems with countable set of labels and discrete $\sigma$-algebra on states. Labelled Markov processes correspond to the subclass with one transition for each label from each state, plus the condition that the transition function is a Markov kernel, for each action. Since LMPs only allow for countable sets of actions, the space of labels is trivially analytic. Labelled transition systems are the subset with only Dirac measures.

Finally, stochastic transition systems could be used as the operational model for a stochastic process algebra as follows: the state space is given by the product of the space of possible algebraic terms, which is countable, and the space of possible clock valuations, which is $\mathcal{R}^n$, where $n$ is the number of clocks. Since both these spaces are analytic, their product is analytic. Likewise, since labels record both the actions and the corresponding real-valued delay, the space of labels is the product of a countable set and the reals, resulting into an analytic set.

We have placed no restrictions on the transition relation; as we will see, this freedom will cause problems, producing intractable behaviour, and we will place restrictions later when needed. We use analytic spaces because they offer a stronger framework than standard measure spaces; for example, conditional probabilities exist for analytic spaces, while they are not guaranteed to exist in general. This is a reasonable restriction as the most common spaces (finite, countable, the reals) are analytic and analytic spaces are closed under the usual operations (product, union, etc..). The state space generated by the process algebra of Chapter 10 is analytic.

### 8.2.2 Combined Transitions

We follow [Seg95b] and we resolve non-determinism in a randomised way, and combine the transitions leaving a state $q$ in order to obtain a new transition. Similarly to the discrete case, this induces a probability measure on the set of transitions leaving state $q$, that is, a measure $\pi$ on $\mathcal{T}$ with a support contained in $\mathcal{T}(q)$. Since different transitions have in general different labels, the combination of the transitions leaving a state results in a new distribution on both labels and target states.

**Definition 8.2 (Combined transitions).** Given a state $q$ and a sub-probability measure $\pi$ on $\mathcal{D}(\mathcal{T}, \mathcal{F}_\mathcal{T})$ with a support contained in $\mathcal{T}(q)$, the combined transition for $\pi$ from $q$ is

the pair $(q, \mu_\pi)$ (denoted by $q \to \mu$), where $\mu_\pi$ is the probability measure on $(L \times Q, \mathcal{F}_L \otimes \mathcal{F}_Q)$ defined as follows:

$$\mu_\pi(A \times X) = \int_{(q,a,\mu)\in\mathcal{T}} I_A(a)\mu(X)d\pi \tag{8.2}$$

The integral of Equation 8.2 is well defined for the $\sigma$-algebra $\mathcal{F}_\mathcal{T}$ on transitions, and the measure $\mu_\pi$ extends uniquely to the $\sigma$-algebra $\mathcal{F}_{L\times Q}$. It is easy to show that $\mu_\pi$ is a sub-probability measure under the condition that $\pi$ is one.

Observe that we require $\pi$ to be a sub-probability measure, therefore it is possible that no transition is scheduled with positive probability. We let this denote the probability to stop, which is defined as $\mu_\pi(\bot) = 1 - \mu_\pi(L \times Q)$.

### 8.2.3 Executions and Traces

Given an STS $S$, a possibly infinite alternating sequence of states and actions $\alpha = q_0 a_1 q_1 \dots$ is called an *execution*. Note that we do not place any restrictions on the sequence of actions and states: while for standard transition systems we require states and labels to form a transition, this condition would not make sense in the case of stochastic transition systems as all states are generally reached with probability zero.

We denote the set of executions by *Exec*, the set of finite executions ending with a state by $Exec^*$ and the set of infinite executions by $Exec^\omega$. Given a finite execution $\alpha$, $lstate(\alpha)$ denotes its last state. The *length* of an execution $\alpha$, denoted by $|\alpha| = n$, is the number of occurrences of actions in $\alpha$; if $\alpha$ is infinite, define $|\alpha| = \infty$. We denote a finite execution $\alpha$ that has terminated by $\alpha\bot$.

A trace is a possibly infinite sequence $t = a_1 a_2 \dots$ of visible actions. We denote the set of traces by *Traces*. Similarly to executions, define the length of a finite trace $t$ as $|t|$, the number of elements in $t$; if $t$ is infinite, then $|t| = \infty$. Given an execution $\alpha$, the function $trace(\alpha)$ returns the sequence of visible actions in $\alpha$.

### 8.2.4 A $\sigma$-algebra on Executions

We define the $\sigma$-algebra $\mathcal{F}_{Exec}$ over the set of executions. This is necessary to study the properties of system runs. In the discrete case, $\mathcal{F}_{Exec}$ is the $\sigma$-algebra generated by cones, that is, the set of executions that extend some finite prefix. This concept is generalised to the continuous case by using sets of executions called basic sets.

**Definition 8.3 (Basic Sets).** Given a non empty finite sequence of measurable sets $\Lambda = X_0 A_1 X_1 \dots A_n X_n$, $A_i \in \mathcal{F}_L$, $i \in \{1..n\}$ and $X_i \in \mathcal{F}_Q$, $i \in \{0..n\}$, the *basic set with base* $\Lambda$ is defined as follows:

$$\mathcal{C}_\Lambda = \{q_0 a_1 \dots q_n \alpha \mid \forall i \in \{0..n\}\ q_i \in X_i,\ \forall i \in \{1..n\}\ a_i \in A_i \text{ and } \alpha \in Exec\} \tag{8.3}$$

The length of a basic set $\mathcal{C}_\Lambda$ (also called a cone) is given by the number of occurrences of elements of $\mathcal{F}_L$ in $\Lambda$. We prove the following important property about basic sets.

**Proposition 8.4.** *The family of basic sets of Definition 8.3 forms a semi-ring.*

*Proof.* Let us check that the three properties of a semi-ring are met.

1. The empty set is generated by any basic set with a base $X_0 A_1 \dots A_n X_n$ such that at least one measurable set of actions or states is empty.

2. Closure under finite intersection: consider two bases $X_0A_1 \dots A_nX_n$ and $Y_0B_1 \dots B_nY_n$ (we can assume the two bases have the same length, or else we can "complete" the shorter one with the measurable sets $L$ and $Q$ up to the length of the longer one). Then the intersection of the cones generated by these two bases is the cone generated by the base $(X_0 \cap Y_0)(A_1 \cap B_1) \dots (A_n \cap B_n)(X_n \cap Y_n)$.

3. For all $C_1$ and $C_2$ such that $C_1 \subseteq C_2$, there must exist a finite family of pairwise disjoint sets $C_i$ such that $C_2 \setminus C_1 = \cup_i C_i$. Consider two basic sets $C_1$ and $C_2$, with bases $X_0A_1 \dots A_nX_n$ and $Y_0B_1 \dots B_nY_n$, respectively, such that $C_1 \subseteq C_2$. Again, we can assume that the two bases have the same length. Since $C_1 \subseteq C_2$, it must follow that $X_i \subseteq Y_i$ for all $i = 0 \dots n$ and that $A_j \subseteq B_j$ for all $j = 1 \dots n$. For all non-empty sets of indices $I, J$, such that $I \subseteq \{0, \dots, n\}$ and $J \subseteq \{1 \dots n\}$, define $\Lambda_{I,J} = Z_0C_1Z_1 \dots C_nZ_n$, where:

$$Z_i = \begin{cases} Y_i \setminus X_i & \text{if } i \in I \\ X_i & \text{otherwise} \end{cases}$$

$$C_j = \begin{cases} B_j \setminus A_j & \text{if } j \in J \\ A_j & \text{otherwise} \end{cases}$$

Clearly, all the basic sets $C_{\Lambda_{I,J}}$ generated by such sequences are disjoint, as there always is at least one index that distinguishes two sets.

Let us show that any $\alpha \in C_1 \setminus C_2$ belongs to one such basic set: if $\alpha = q_0a_1 \dots a_nq_n \in C_1 \setminus C_2$, then there must exist two sets of indices $I_\alpha$ and $J_\alpha$, at least one of which non-empty (or else $\alpha \in C_1$), such that $q_i \in Y_i \setminus X_i$ for all $i \in I_\alpha$ and $a_i \in B_j \setminus A_j$ for all $j \in J_\alpha$. This implies that $\alpha \in C_{\Lambda_{I_\alpha,J_\alpha}}$.

□

The result above allows us to construct measures on cones by defining their value only for the basic sets, since such measures uniquely extend to the $\sigma$-algebra generated. We denote the $\sigma$-algebra generated by basic sets by $\mathcal{F}_{Exec}$.

**A $\sigma$-algebra on finite executions.** We define the $\sigma$-algebra $\mathcal{F}_{Exec^*}$ on finite executions in a similar way as the $\sigma$-algebra generated by the sets of the form

$$Q_0A_1 \dots Q_n = \{q_0a_1 \dots q_n \mid \forall i = 0 \dots n \; q_i \in Q_i \; \wedge \; \forall j = 1 \dots n \; a_j \in A_j\} \tag{8.4}$$

where $Q_0 \dots Q_n \in \mathcal{F}_Q$ and $A_1 \dots A_n \in \mathcal{F}_L$. $(Exec^*, \mathcal{F}_{Exec^*})$ is the measurable space of finite executions. Note that $\mathcal{F}_{Exec^*}$ is the restriction of $\mathcal{F}_{Exec}$ to finite executions. The family of sets defined above forms a semi-ring, and the proof is similar to that of Proposition 8.4; however, we will not need this result in the following.

**A $\sigma$-algebra on traces.** The $\sigma$-algebra $\mathcal{F}_{Traces}$ is defined in the same way as the $\sigma$-algebra $\mathcal{F}_{Exec}$ on executions, in terms of basic sets.

**Definition 8.5 (Basic sets for traces).** Given a sequence of measurable sets of visible labels $A_1 \dots A_n$, $A_i \in \mathcal{F}_L$ and $A_i \subseteq L^e$, the basic set for traces with base $A_1 \dots A_n$, is defined as follows:

$$C_{A_1 \dots A_n} = \{a_1 \dots a_nt' \mid \forall i \in \{0..n\} \; a_i \in A_i \; \wedge \; t' \in Traces\} \tag{8.5}$$

The $\sigma$-algebra $\mathcal{F}_{Traces}$ on traces is defined as the $\sigma$-algebra generated by the basic sets above. The following result follows from this definition.

**Proposition 8.6.** *The family of basic sets of Definition 8.5 forms a semi-ring.*

*Proof.* The proof follows the same steps as the proof of Proposition 8.4. □

We prove that the function $trace : Exec \to Traces$ is measurable with respect to the $\sigma$-algebras defined in this section. This property is important as it allows us to define the measure on $(Traces, \mathcal{F}_{Traces})$ corresponding to any measure on $(Exec, \mathcal{F}_{Exec})$.

**Theorem 8.7.** *The function* $trace : (Exec, \mathcal{F}_{Exec}) \to (Traces, \mathcal{F}_{Traces})$ *is measurable.*

*Proof.* Let $C_{A_1 \dots A_n}$ be a basic set of traces. We define its counter-image under the function *trace* in terms of basic sets of executions:

$$trace^{-1}(C_{A_1 \dots A_n}) = \{\alpha \in exec \mid trace(\alpha) \in C_{A_1 \dots A_n}\}$$

The elements of this set are of the form:

$$q(\tau q)^* a_1 (\tau q)^* \dots a_n (\tau q) *$$

for arbitrary states $q$. The set of such executions can be expressed as follows:

$$trace^{-1}(C_{A_1 \dots A_n}) = \bigcup_{j_1 \geq 0} \bigcup_{j_2 \geq 0} \cdots \bigcup_{j_{n+1} \geq 0} Q(L^\tau Q)^{j_1} A_1 Q(L^\tau Q)^{j_2} \dots A_n Q(L^\tau Q)^{j_{n+1}}$$

This is a countable union of basic sets; it follows that $trace^{-1}(C_{A_1 \dots A_n}) \in \mathcal{F}_{Exec}$. □

### 8.2.5 Schedulers

We use schedulers as the entities that resolve non-determinism. Given a history in the form of a sequence of states and labels that the system has visited, a scheduler chooses the next transition from the current state by assigning a sub-probability measure to the enabled transitions.

**Definition 8.8 (Schedulers).** A scheduler is a function

$$\eta : Exec^* \to sub\mathcal{D}(\mathcal{T}) \tag{8.6}$$

such that, for all $\alpha \in Exec^*$, $\mathcal{T}(lstate(\alpha))$ is a support for $\eta(\alpha)$.

We denote the set of schedulers by $\mathcal{A}$. Since a scheduler $\eta$ returns a distribution on transitions for each finite execution $\alpha$, it induces a combined transition $(lstate(\alpha), \mu_{\eta(\alpha)})$ leaving the last state of each execution. Note that we use *randomised* schedulers; originally introduced for discrete systems, they have been shown to have important properties, for example the probabilistic temporal logic PCTL is preserved by bisimulation under randomised schedulers [SL95]. Randomised schedulers are also necessary to obtain compositionality under parallel compositions (see Section 8.4). Non-randomised (deterministic) schedulers can be seen as the subclass of schedulers that return a Dirac distribution after each execution.

According to the above definition, a scheduler can make arbitrary choices at each point of the computation. We define a class of schedulers whose global behaviour respects measurability properties.

**Definition 8.9 (Measurable schedulers).** A scheduler $\eta$ is measurable if the function $f_\eta(\alpha) = \mu_{\eta(\alpha)}$ (called the *flattening* of $\eta$) is a measurable function from $(Exec^*, \mathcal{F}_{Exec^*})$ to $(sub\mathcal{D}(L \times Q), \mathcal{F}_{sub\mathcal{D}(L\times Q)})$. We denote the class of measurable schedulers by $\mathcal{A}_{meas}$.

We have defined two different classifications of schedulers, depending on whether they are measurable or not, or whether they are randomised or not. Other classes of schedulers are identified in the literature, most notably depending how the past history, that is, the execution so far, affects the decision of the schedulers. Here, we assume the most general case, where the decisions of a scheduler depend on all the past history. These aspects of schedulers can be seen as orthogonal to measurability issues and therefore are not relevant for us.

## 8.3 Probabilistic Executions

The interaction of an STS $S$ and a scheduler $\eta$ results in a system with no non-determinism, i.e. a purely probabilistic process. We call this object a *probabilistic execution* following [Seg95b].

**Definition 8.10 (Probabilistic executions).** Given an STS $S$ and a scheduler $\eta$, the probabilistic execution for $S$ and $\eta$ rooted in $q$ is the tuple

$$P = (Exec^*, \mathcal{F}_{L\times Q}, q, \omega) \tag{8.7}$$

where $\omega : Exec^* \times \mathcal{F}_{L\times Q} \to [0,1]$ such that for each $\alpha \in Exec^*$ $\omega(\alpha, \cdot)$ is a sub-probability measure over $L \times Q$ defined by $\mu_{\eta(\alpha)}$.

A probabilistic execution defines the transitions induced by the scheduler $\eta$: given a finite execution it returns the combined transition scheduled by $\eta$. We write $\omega_X$, $X \in \mathcal{F}_{L\times Q}$, whenever we fix $X$ and $\omega$ is a function on $Exec^*$; similarly, we write $\omega_\alpha$ $(= \mu_{\eta(\alpha)})$, whenever we fix $\alpha$ and $\omega$ is a measure on $\mathcal{F}_{L\times Q}$.

We denote the transition function of probabilistic execution $P$ by $\omega$ and its start state by $q$ (in general, the set of executions and $\sigma$-algebra on labels and states are fixed), and we propagate indices when necessary; thus, the elements of the probabilistic execution $P_i$ are $\omega_i$ and $q_i$. Whenever the root state is the start state of the STS, that is, $q = \overline{q}$, we omit the root state from the definition of a probabilistic execution.

Not all probabilistic executions are "good"; our objective is to define a measure on executions, essential to define weak transitions, and the measurability of the function $\omega$ is necessary for this purpose. In the purely probabilistic case (no non-determinism) this problem is solved by using Markov kernels (e.g. [DEP98]). We adapt this idea to our setting by defining measurable probabilistic executions and by studying the conditions under which they are generated.

**Definition 8.11 (Measurable probabilistic executions).** A probabilistic execution $P_{S,\eta} = (Exec^*, \mathcal{F}_{L\times Q}, q, \omega)$ is measurable if $\omega : Exec^* \times \mathcal{F}_{L\times Q} \to [0,1]$ is such that $\omega(\cdot, X)$ is a measurable function for each $X \in \mathcal{F}_{L\times Q}$.

When $\omega$ has such measurability property, we can see it as a generalisation of Markov kernels to the history-dependent case.

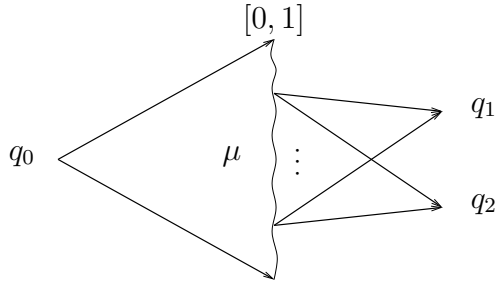



Figure 8.1: A simple stochastic transition system.

### 8.3.1 Measurable Schedulers and Probabilistic Executions

We aim to extend the results of the discrete case to stochastic transition systems and define the measure on executions induced by a scheduler. This requires the corresponding probabilistic execution to be measurable. In this section we show that the class of measurable schedulers identifies all and only the measurable probabilistic executions. The following example shows that arbitrary schedulers could produce "bad" executions and explains why considering the point-wise behaviour of a scheduler is not enough in the continuous setting; instead, it is necessary to consider its global behaviour.

**Example 8.1.** Consider the system of Figure 8.1: the initial state $q_0$ enables a single transition with some measure $\mu$ on the interval $[0, 1]$. From each state in $[0, 1]$ two Dirac transitions are enabled: one to $q_1$ and the other to $q_2$. Labels are not relevant. The probability of moving to $q_1$ after two steps under a scheduler $\eta$ is given by $\int_{[0,1]} \mu_{\eta(q)}(\{q_1\})\mu(dq)$. Let $\eta$ be the scheduler that chooses $q_1$ from a non-measurable subset $A$ of $[0, 1]$, and $q_2$ from its complement. The integral above is not defined as $\mu_{\eta(q)}(\{q_1\})$ is not a measurable function, that is, the probabilistic execution is not measurable. We want to rule out such a probabilistic execution as "pathological" and disallow the scheduler generating it.

We restrict our analysis to measurable executions only, as they represent "well behaved", feasible, schedulers and allow us to define probability measure on paths. We think this is not an unreasonable restriction since schedulers that produce non-measurable executions represent pathological cases and thus can be discarded. A similar approach has been adopted in [KNSS00], where only the schedulers that preserve the measurability of logical formulae are considered, though without studying the nature of such schedulers.

**Proposition 8.12.** *Given an STS $S$, and a scheduler $\eta$, $\eta$ is measurable if and only if the probabilistic execution $P$ for $\eta$ is measurable.*

*Proof.* We prove the two directions:

- *If:* Let $f_\eta$ be the flattening of $\eta$ as in Definition 8.9. We have to show that $f_\eta^{-1}(D) \in \mathcal{F}_{Exec^*}$ for all $D \in \mathcal{F}_{sub\mathcal{D}(L\times Q)}$. Firstly, we prove it for the generators $D_{X,I}$ of $\mathcal{F}_{sub\mathcal{D}(L\times Q)}$, for all $X \in \mathcal{F}_{L\times Q}$ and $I \in \mathcal{B}([0, 1])$. Consider one of such $D_{X,I}$. Since $P$ is measurable, it follows that $\omega_X^{-1}(I) = Y \in \mathcal{F}_{Exec^*}$. We show that $Y = f_\eta^{-1}(D_{X,I})$:
  - $f_\eta^{-1}(D_{X,I}) \supseteq Y$: consider $\alpha \in Y$, then $\omega(\alpha, X) \in I$; this is equivalent to $\mu_{\eta(\alpha)}(X) \in I$, which implies $\mu_{\eta(\alpha)}(X) \in D_{X,I}$. It follows that $\alpha \in f_\eta^{-1}(D_{X,I})$.
  - $f_\eta^{-1}(D_{X,I}) \subseteq Y$: consider $\alpha \in f_\eta^{-1}(D_{X,I})$. Then $\mu_{\eta(\alpha)}(X) \in I$, that is, $\omega(\alpha, X) \in I$. This, of course, means that $\alpha \in \omega_X^{-1}(I) = Y$.

We now extend this result to the algebra generated by the family $F$ of sets $D_{I,X}$ according to the construction of Theorem 2.7, that is, by successively closing the set $F$ under complementation, finite interaction and finite union.

- $F$ is already closed under complementation: given $D_{X,I}$, its complement is given by the set $D_{X,[0,1]\setminus I}$ of distribution that assign $X$ a probability distribution not in $I$.
- Let $F_1$ be the set obtained from closing $F$ under finite intersections. Consider a set $D = D_{X_1,I_1} \cap D_{X_1,I_2}$.

$$\begin{aligned} f_\eta^{-1}(D) &= \{\alpha \in Exec^* \mid \mu_{\eta(\alpha)}(X_1) \in I_1 \ \wedge \ \mu_{\eta(\alpha)}(X_2) \in I_2\} \\ &= \{\alpha \in Exec^* \mid \mu_{\eta(\alpha)}(X_1) \in I_1\} \ \cap \ \{\alpha \in Exec^* \mid \mu_{\eta(\alpha)}(X_2) \in I_2\} \\ &= f_\eta^{-1}(D_{X_1,I_1}) \ \cap \ f_\eta^{-1}(D_{X_2,I_2}) \ \in \mathcal{F}_{Exec^*} \end{aligned}$$

  This proves that each scheduled of $F_1$ is enabled from a measurable set of executions.
- Let $F_2$ be the set obtained by closing $F_1$ under finite union of disjoint sets. Consider $D_1, D_2 \in F_1$, such that $D_1 \cap D_2 = \emptyset$. We need to prove that the set of executions from which a distribution in $D_1 \cup D_2$ is scheduled is measurable:

$$\begin{aligned} f_\eta^{-1}(D_1 \cup D_2) &= \{\alpha \in Exec^* \mid \omega(\alpha,\cdot) \in D_1 \ \vee \ \omega(\alpha,\cdot) \in D_1\} \\ &= \{\alpha \in Exec^* \mid \omega(\alpha,\cdot) \in D_1\} \ \cup \ \{\alpha \in Exec^* \mid \omega(\alpha,\cdot) \in D_2\} \\ &= f_\eta^{-1}(D_2) \ \cup \ f_\eta^{-1}(D_1) \ \in \mathcal{F}_{Exec^*} \end{aligned}$$

  We have proved that the function $f_\eta$ is measurable on the algebra generating the $\sigma$-algebra $\mathcal{F}_{sub\mathcal{D}(L\times Q)}$; Theorem 2.8 guarantees that $f_\eta$ is measurable with respect to the whole $\sigma$-algebra.

- *Only if:* consider $P_{S,\eta} = (Exec^*, \mathcal{F}_{L\times Q}, \omega)$; we have to show that for all $X \in \mathcal{F}_{L\times Q}$ and for all $I \in \mathcal{B}([0,1])$, $\omega_X^{-1}(I) \in \mathcal{F}_{Exec^*}$. It is easy to observe that $\omega_X^{-1}(I)$ corresponds to all the executions from which a distribution in the generator $D_{X,I}$ of $\sigma$-algebra on distributions (see Subsection 2.3.3) is scheduled. The measurability of the scheduler ensures that such set of executions is in $\mathcal{F}_{Exec^*}$, as required.

□

The proposition above shows that measurable schedulers generate all and only the measurable probabilistic executions, that is, all the probabilistic executions that we are interested in. We can therefore disallow non-measurable schedulers.

### 8.3.2 A Probability Measure on Executions

We can now define the measure on $(Exec, \mathcal{F}_{Exec})$ induced by a scheduler and show that it is defined only for measurable schedulers. We define the measure $\delta_{\eta,q}$ on basic sets induced by a scheduler $\eta$ from a start state $q$ inductively on the length of the basic sets as follows:

$$\delta_{\eta,q}(\mathcal{C}_X) = \begin{cases} 1 & \text{if } q \in X \\ 0 & \text{otherwise} \end{cases} \tag{8.8}$$

and, for the inductive step,

$$\delta_{\eta,q}(\mathcal{C}_{\Lambda A X}) = \int_{\alpha\in\Lambda} \mu_{\eta(\alpha)}(A,X)\delta_\eta(d\alpha) \tag{8.9}$$

The integral above is defined when the function $f(\alpha) = \mu_{\eta(\alpha)}(A, X)$ is measurable from the measure space of finite executions to $[0, 1]$. From Proposition 8.12, this is true whenever we deal with measurable schedulers. The measure $\delta_{\eta,q}$ extends uniquely to $\mathcal{F}_{Exec}$ since basic sets form a semi-ring (Theorem 2.10).

We get the following Proposition.

**Proposition 8.13.** *Given an STS $S$ and a scheduler $\eta$, the measure $\delta_{\eta,q}$ is defined for all basic sets if and only if $\eta$ is measurable.*

*Proof.* The proof is a consequence of the definition of $\delta_{\eta,q}$ and of Proposition 8.12. □

Using the measure defined above, and since schedulers use sub-probability distributions, we can define the probability of a set of finite executions that have terminated as the probability to stop after each execution. Formally, given a sequence $\Lambda = X_0 A_1 \dots A_n X_n$ of measurable sets of states and actions, we define the probability to stop after $\Lambda$ as $\delta_{\eta,q}(\mathcal{C}_{\Lambda\perp}) = \int_{\alpha\in\Lambda} \mu_{\eta(\alpha)}(\perp)$. The cones $\mathcal{C}_{\lambda\perp}$ are in fact the generators of $\mathcal{F}_{Exec^*}$. The probability of eventually terminating is the probability of finite executions, defined as $\delta_{\eta,q}(Exec^*) = \cup_{i\geq 0}\delta_{\eta,q}(\mathcal{C}_{Q(LQ)^i\perp})$.

Every measurable probabilistic execution $P = (Exec^*, \mathcal{F}_{L\times Q}, \omega)$ induces a measure on executions. Therefore, we stress this relationship by propagating the indices of the probabilistic execution to the corresponding measure on executions; thus, the measure corresponding to $P_i$ is denoted by $\delta_i$.

By restricting to measurable schedulers, we have obtained all and only the objects that we need in order to study the behaviour of runs of a system. Moreover, we have developed all the necessary tools to define weak transitions as the "collapse" of several transitions into a single one with only one visible action.

### 8.3.3 Weak Transitions

The results of this section allow us to define weak transitions for stochastic transition systems. Weak transitions were first defined in the context of CCS [Mil89] and abstract from internal behaviour by considering sequences of actions of the type $\tau^* a \tau^*$, where $\tau$ denotes a generic internal action. We extend the approach of probabilistic automata: in this case weak transitions are defined as probabilistic executions that terminate with probability 1 and assign non-zero probability only to executions whose trace is given by a single occurrence of the visible action $a$.

Weak transitions have not been defined for labelled Markov chains, since this model does not incorporate non-determinism. A notion of weak transitions (and weak bisimulation) is presented in [Bra02]; because of the differences between the models, though, a weak transition is defined as a sequence of non-probabilistic transitions followed by a single probabilistic step. For this reason, the problem of defining the measure resulting from several probabilistic steps and the related problems of measurability did not arise.

In our model, a weak transition is defined as a probabilistic execution such that the set of terminating executions whose trace is given by a single visible action is one.

Recall that the set $L$ of labels is partitioned into two measurable sets $L^e$ and $L^\tau$, to denote visible and invisible actions, respectively. We denote generic internal actions by $\tau$. Weak transitions are defined as a probabilistic execution which terminates with probability 1 and with a support contained in the set of executions containing exactly one visible action. In order to adapt this to the continuous setting, we need to use the results of Section 8.3 and impose the measurability of probabilistic executions. Let $\mathcal{W}_A$ denote

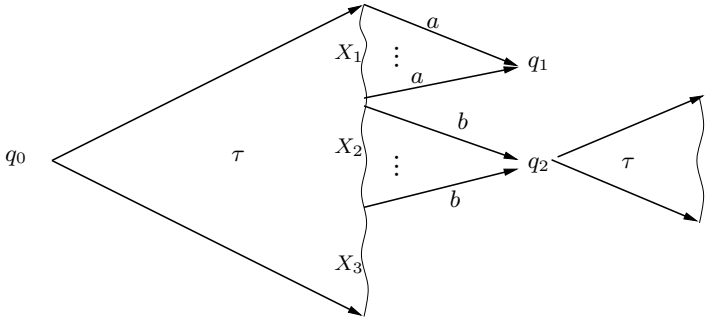



Figure 8.2: A simple weak transition.

the executions whose visible trace is exactly one action $a \in A \subseteq L$ and $\mathcal{W} = \mathcal{W}_L$. $\mathcal{W}$ is measurable and it can be constructed from basic sets as follows:

$$\mathcal{W}_A = \cup_{i \geq 0} \left( \cap_{j \leq 0} C_{Q(L^\tau Q)^i Q A (L^\tau Q)^j} \right)$$

$\mathcal{W}_A$ is given by the union of all the sets of executions with only one action $A$ at position $i$; the inner intersection is necessary to ensure that only $\tau$ actions follow the occurrence of the visible action.

**Definition 8.14 (Weak Transitions).** The pair $(q, \mu)$, $q \in Q$ and $\mu \in \mathcal{D}(L \times Q)$, is a weak transition (denoted by $q \Rightarrow \mu$) if there exists a measurable scheduler $\eta$ such that $\delta_{\eta,q}(Exec^*) = 1$, $\delta_{\eta,q}(\mathcal{W}) = 1$ and $\mu$ is defined as follows:

$$\mu(A, X) = \delta_{\eta,q}((\cup_{i \geq 0} \mathcal{C}_{Q(LQ)^i L X \perp}) \cap \mathcal{W}_A) \tag{8.10}$$

for all $A \in \mathcal{F}_L$ and $X \in \mathcal{F}_Q$.

According to Equation 8.10, the probability of a measurable set of actions $A$ and a measurable set of states $X$ for a weak transition $q \Rightarrow \mu$ is given by probability of the set of executions terminating in $A$ ($\cup_{i \geq 0} \mathcal{C}_{Q(LQ)^i X \perp}$) whose visible trace is an action in $A$ (intersection with $\mathcal{W}_A$). It is easy to show that, under the termination condition $\delta_{\eta,q}(Exec^*)$, $\mu$ is a probability measure on $L \times Q$. Note that it is possible to assign non zero probability to the set of executions $\mathcal{W}_{\{\tau\}}$ that only execute internal actions. We also allow different actions within one weak transition: in a continuous environment single actions usually have probability zero and, therefore, we allow transitions to be combined and we consider the probability of measurable sets of actions. However, since singletons are measurable in analytic spaces, the single-labelled weak transitions that are standard in the literature are a special case of those of Definition 8.14.

**Example 8.2.** A simple probabilistic execution under some scheduler $\eta$ generating a weak transition is shown in Figure 8.2. From the initial state $q_0$ a transition labelled by $\tau$ is enabled, with target distribution $\mu$, such that $\mu(X_1) = \mu(X_2) = \mu(X_3) = 1/3$. From each state $x_1 \in X_1$ a Dirac transition on $q_1$, labelled by $a$, is enabled, while from each state $x_2 \in X_2$ a Dirac transition on $q_2$, labelled by $b$, is enabled; no transition is enabled from the states in $X_3$. Finally, an arbitrary transition labelled by $\tau$ is enabled from $q_2$, after which the computation stops.

The target distribution is computed on the set of states from which no further transition is scheduled. For instance, the final probability of executing action $a$ is given by the probability of the set of executions whose corresponding sequence of actions is of the for $\tau^* a \tau^*$. In

our example, this probability is given by $\delta_{\eta,q_0}(\{q_0\}\,\{\tau\}X_1\,\{a\}\{q_1\})$, which is 1/3. The weak transition executes a $b$ with probability 1/3 (given by $\delta_{\eta,q_0}(\{q_0\}\,\{\tau\}X_2\,\{b\}\{q_2\}\,\{\tau\}Q)$), and only invisible actions with probability 1/3 (given by $\delta_{\eta,q_0}(\{q_0\}\,\{\tau\}X_3)$). Therefore, the target distribution assigns probability 1/3 to the action $a$, 1/3 to action $b$ and 1/3 to the silent action $\tau$.

## 8.4 Parallel Composition and Measurability

In this section we introduce a CSP-style parallel operator [Hoa85], under which two STSs synchronise on a common interface alphabet, and study the compositionality properties of schedulers and measurable executions.

### 8.4.1 Parallel Composition

Given an STS $S$, we further partition its label space to two measurable sets $L^p$ and $L^i$ of private and interface labels, respectively. We require $L^i$ to be contained in the set of visible actions $L^e$, as systems do not synchronise on internal actions. We say that two STSs $S_1$ and $S_2$ are *compatible* if $L_1^p \cap L_2 = \emptyset$ and $L_2^p \cap L_1 = \emptyset$. We denote the union of the measurable spaces of labels by $(L, \mathcal{F}_L)$. We can now define the parallel composition between two compatible STSs.

**Definition 8.15 (Parallel Composition).** Let $S_1$ and $S_2$ be two compatible labelled stochastic transition systems. The parallel composition $S$ of $S_1$ and $S_2$ is the system

$$S_1 \parallel S_2 = ((Q, \mathcal{F}_Q), \overline{q}, (L, \mathcal{F}_L), \rightarrow) \tag{8.11}$$

where:

- $(Q, \mathcal{F}_Q) = (Q_1 \times Q_2, \mathcal{F}_{Q_1} \otimes \mathcal{F}_{Q_2})$.
- $\overline{q} = (\overline{q}_1, \overline{q}_2)$.
- $(L, \mathcal{F}_L)$ is the union of the labels of the components.
- $\rightarrow \subseteq L \times Q \times \mathcal{D}(Q)$ such that $((q_1, q_2), a, \mu_1 \otimes \mu_2) \in \rightarrow$ iff, for all $i \in \{1, 2\}$, either:
  - $a \in L_i$ and $(q_i, a, \mu_i) \in \rightarrow_i$, or
  - $a \notin L_i$ and $\mu_i = \mathit{Dirac}(q_i)$.

Observe that $S_1 \parallel S_2$ is a well defined STS given the closure properties of analytic spaces. The condition on the transition relation states that two systems in parallel need to synchronise on common actions, and the resulting transition is given by the product measure of the original target measures; actions not in the interface are executed independently.

### 8.4.2 Compositionality Results

We define two families of functions $\pi_1$ and $\pi_2$ to be the left and right projections respectively. Given a state $q$ of $S$, the projection $\pi_i$ returns the $i$-th component of $q$. For an execution $\alpha$ of $S$, define the projection $\pi_i(\alpha)$ as the execution of $S_i$ obtained from $\alpha$ by projecting all the states and removing all the actions not in $L_i$ together with the subsequent state. Given a distribution $\mu$ on $Q_1 \times Q_2$, the projection $\pi_i(\mu)$ is the distribution on $Q_i$ induced by $\pi_i$. Finally, given a transition $t = ((q_1, q_2), a, \mu)$, its projection $\pi_i(t)$ is

$(q_i, a, \pi_i(\mu))$. If $a \notin L_i$ the projection $\pi_i(t)$ is still defined but it does not correspond to a possible transition of $S_i$. Note that all the variants of $\pi_1$ and $\pi_2$ are measurable functions.

The following two theorems are important for compositional reasoning.

**Theorem 8.16.** *Let $S_1$ and $S_2$ be two compatible STSs and $\alpha$ be an execution of $S_1 \mid\mid S_2$. Then $\pi_i(\alpha)$ is an execution of $S_i$, for $i \in \{1, 2\}$.*

*Proof.* The proof is straightforward as, by definition of $\pi_i$, $\pi_i(\alpha)$ returns a sequence of elements of $S_i$, which is an execution of $S_i$. □

The following theorem is more interesting and states that also all the probabilistic executions of the parallel composition of two systems can be expressed as the composition of the probabilistic executions of the components. Moreover, the measurability properties of schedulers and probabilistic executions are preserved through parallel composition. This is also the first result that uses the fact that the state spaces are analytic, since we need to define conditional probabilities.

**Theorem 8.17.** *Let $S_1$ and $S_2$ be two compatible STSs and $\eta$ a measurable scheduler for $S_1 \mid\mid S_2$; then there exists a measurable scheduler $\eta_1$ for $S_1$ such that $\delta_{\eta_1,\overline{q}_1} = \pi_1(\delta_{\eta,\overline{q}})$.*

*Proof outline.* We define the scheduler $\eta_1$ on the first component as follows

$$\eta_1(\alpha_1)(T) = \int_{\alpha \in \pi_1^{-1}(\alpha_1)} \eta(\alpha)(\pi_1^{-1}(T))\delta_{\eta,\overline{q}}(d\alpha|\alpha_1) \tag{8.12}$$

for all $T \in \mathcal{F}_{\mathcal{T}}$, where $\delta_{\eta,\overline{q}}(\cdot|\alpha_1)$ is the regular conditional probability for $\delta_{\eta,\overline{q}}$ with respect to $\pi_1$, whose existence follows from Theorem 2.12 since we work with analytic spaces. Firstly, we show that $\eta_1$ defines a legal scheduler for $S_1$ and its measurability follows from Theorem 2.13. Then, in order to prove that $\delta_{\eta_1,\overline{q}_1} = \pi_1(\delta_\eta, \overline{q})$, we need to show that

$$\delta_{\eta_1,\overline{q}_1}(\mathcal{C}_\Lambda) = \delta_{\eta,\overline{q}}(\pi^{-1}(\mathcal{C}_\Lambda)) \tag{8.13}$$

for all basic sets $\mathcal{C}_\Lambda$. Equation (8.13) is proved by exploiting the properties of regular conditional probabilities. Since the two measures agree on the basic sets, which form a semi-ring, and since they are $\sigma$-finite (all probability measures are), they extend to the same measure by Theorem 2.10. □

*Proof.* In order to simplify the notation, the measure induced by $\eta$ is denoted by $\delta$, the measure induced by $\eta_1$ by $\delta_1$ and the conditional probability for $\delta$ with respect to $\pi_1$ by $\delta(\cdot, \alpha)$. Consider $\eta_1$ as defined in Equation 8.12; it easy to see that $\eta_1(\alpha_1)$ is a sub-probability distribution on $\mathcal{T}_1$ for all $\alpha_1 \in Exec_1^*$.

1. First we show $\eta_1(\alpha_1)(\mathcal{T}_1) \leq 1$. By construction we have:

$$\begin{array}{rcl} \eta_1(\alpha_1)(\mathcal{T}_1) & = & \int_{\alpha \in \pi_1^{-1}(\alpha_1)} \eta(\alpha)(\pi_1^{-1}(\mathcal{T}_1))\delta(d\alpha|\alpha_1) \\ & = & \int_{\alpha \in \pi_1^{-1}(\alpha_1)} \eta(\alpha)(\mathcal{T})\delta(d\alpha|\alpha_1) \\ & \leq & 1 \end{array}$$

since $\eta(\alpha)(\mathcal{T}) \leq 1$ for all $\alpha$.

2. For countable additivity, again consider any family $\{T_i\}$ of pairwise disjoint measurable sets of transitions, then by construction of $\eta_1$ we have:

$$\begin{aligned}\eta_1(\alpha_1)(\textstyle\bigcup_i T_i) &= \textstyle\int_{\alpha\in\pi_1^{-1}(\alpha_1)} \eta(\alpha)(\pi_1^{-1}(\bigcup_i T_i))\delta(d\alpha|\alpha_1)\\ &= \textstyle\int_{\alpha\in\pi_1^{-1}(\alpha_1)} \eta(\alpha)(\bigcup_i(\pi_1^{-1}(T_i)))\delta(d\alpha|\alpha_1)\\ &= \textstyle\int_{\alpha\in\pi_1^{-1}(\alpha_1)} \sum_i \eta(\alpha)(\pi_1^{-1}(T_i))\delta(d\alpha|\alpha_1)\\ &= \textstyle\sum_i \eta_1(\alpha_1)(T_i)\end{aligned}$$

It is easy to observe that the support of $\eta_1(\alpha_1)$ is contained in $\mathcal{T}(lstate(\alpha_1))$: consider a set of transitions $T_1$ that is not contained in $\mathcal{T}(lstate(\alpha_1))$, that is, $T_1 \cap \mathcal{T}(lstate(\alpha_1)) = \emptyset$. Also the counter-images of these sets under $\pi_1$ have empty intersections, $\pi_1^{-1}(T_1) \cap \pi_1^{-1}(\mathcal{T}(lstate(\alpha_1))) = \emptyset$, and therefore $\eta(\pi_1^{-1}(T_1)) = 0$. It follows that $T_1$ must have probability zero under $\eta_1(\alpha_1)$:

$$\begin{aligned}\eta_1(\alpha_1)(T_1) &= \int\limits_{\alpha\in\pi_1^{-1}(\alpha_1)} \eta(\alpha)(\pi_1^{-1}(T))\delta(d\alpha|\alpha_1)\\ &= 0\end{aligned}$$

This proves that $\eta_1$ is a well defined scheduler for $S_1$.

We now prove that the measure that $\eta_1$ induces on the executions of $S_1$ is $\pi_1(\delta)$. It is sufficient to consider only basic sets, that is, to show that for all $X_0A_1\ldots A_nX_n$, $X_i \in \mathcal{F}_1$ for all $i$, the following holds:

$$\begin{aligned}\delta_1\left(C_{X_0A_1\ldots A_nX_n}\right) &= \pi_1(\delta)\left(C_{X_0A_1\ldots A_nX_n}\right)\\ &= \delta\left(\pi_1^{-1}\left(C_{X_0A_1\ldots A_nX_n}\right)\right)\end{aligned}$$

We prove this by induction. The basic step is trivial, since the scheduler has not taken any decision yet in order to reach the starting state, and the measure of a cone $\mathcal{C}_X$ of length one is independent of the scheduler (see Equation 8.8). For the inductive step, given a transition $t = (q, a, \mu)$, define $I_A(t) = I_A(a)$ and $p_X(t) = \mu(X)$, for $A \in F_L$ and $X \in \mathcal{F}_Q$. By applying the definition of basic sets (Definition 8.3) and that of the combined transition induced by a scheduler (Definition 8.2), we get:

$$\begin{aligned}\delta_1\left(C_{\Lambda AX}\right) &= \int\limits_{\alpha_1\in\Lambda} \mu_{\eta_1(\alpha_1)}(A, X)\delta_1(d\alpha_1)\\ &= \int\limits_{\alpha_1\in\Lambda} \left(\int\limits_{t_1\in\mathcal{T}_1} I_A(t_1)p_X(t_1)\eta_1(\alpha_1)(dt_1)\right)\delta_1(d\alpha_1)\end{aligned}$$

By applying the definition of $\eta_1$ and by denoting $\eta(\alpha)$ by $\eta_\alpha$, we get:

$$\begin{aligned}&= \int\limits_{\alpha_1\in\Lambda} \left(\int\limits_{t_1\in\mathcal{T}_1} I_A(t_1)p_X(t_1)\left(\int\limits_{\alpha\in\pi_1^{-1}(\alpha_1)} \eta_\alpha(\pi_1^{-1}(dt_1))\delta(d\alpha|\alpha_1)\right)\right)\delta_1(d\alpha_1)\\ &= \int\limits_{\alpha_1\in\Lambda} \left(\int\limits_{t_1\in\mathcal{T}_1}\int\limits_{\alpha\in\pi_1^{-1}(\alpha_1)} I_A(t_1)p_X(t_1)\pi_1(\eta_\alpha)(dt_1)\delta(d\alpha|\alpha_1)\right)\delta_1(d\alpha_1)\end{aligned}$$

where $\pi_1(\eta_\alpha)$ is the projection of $\eta_\alpha$ on $\mathcal{T}_1$, and therefore $\eta_\alpha(\pi_1^{-1}(dt_1)) = \pi_1(\eta_\alpha)(dt_1)$. The two inner integral signs can be swapped because their limits and measures are independent from each other.

$$= \int_{\alpha_1 \in \Lambda} \left( \int_{\alpha \in \pi_1^{-1}(\alpha_1)} \left( \int_{t_1 \in \mathcal{T}_1} I_A(t_1) p_X(t_1) \pi_1(\eta_\alpha)(dt_1) \right) \delta(d\alpha|\alpha_1) \right) \delta_1(d\alpha_1) \tag{8.14}$$

Let us focus on the innermost integral.

$$\int_{t_1 \in \mathcal{T}_1} I_A(t_1) p_X(t_1) \pi_1(\eta_\alpha)(dt_1) \tag{8.15}$$

Consider the conditional measure $\eta(t_1, dt_2)$ of $\eta$ on $\mathcal{T}_2$ for a given $t$ in $\mathcal{T}_1$. This measure exists since we work with analytic spaces.

$$\begin{aligned} &= \int_{t_1 \in \mathcal{T}_1} \left( \int_{t_2 \in \mathcal{T}_2} \eta_\alpha(t_1, dt_2) \right) I_A(t_1) p_X(t_1)\ \pi_1(\eta_\alpha)(dt_1) \\ &= \int_{t_1 \in \mathcal{T}_1} \int_{t_2 \in \mathcal{T}_2} I_A(t_1) p_X(t_1)\ \eta_\alpha(t_1, dt_2)\ \pi_1(\eta_{\alpha 1})(dt_1) \end{aligned}$$

Note that $p_X(t_1) = p_{X \times Q_2}(t_1, t_2)$ and that $I_A(t_1) = I_A((t_1, t_2))$ for all $t_2 \in \mathcal{T}_2$, which gives:

$$= \int_{t_1 \in \mathcal{T}_1} \int_{t_2 \in \mathcal{T}_2} I_A((t_1, t_2)) p_{X \times Q_2}(t_1, t_2)\ \eta_\alpha(t_1, dt_2)\ \pi_1(\eta_\alpha)(dt_1)$$

By applying Fubini's theorem we get:

$$\begin{aligned} &= \int_{t \in \mathcal{T}} I_A(t) p_{X \times Q_2}(t)\ \eta_\alpha(dt) \\ &= \int_{t \in \mathcal{T}} I_A(t) p_{\pi_1^{-1}(X)}(t)\ \eta_\alpha(dt) \end{aligned} \tag{8.16}$$

The term above is function of $\alpha$ and we denote it by $f(\alpha)$. We have rewritten Equation 8.15, which expressed how to combine the transitions of $\mathcal{T}_1$ according to the projection of $\eta_\alpha$ on $\mathcal{T}_1$, in terms of the transitions $\mathcal{T}$ of the composition (Equation 8.16). Equation 8.14 now becomes:

$$\delta_1(\Lambda A X) = \int_{\alpha_1 \in \Lambda} \left( \int_{\alpha \in \pi_1^{-1}(\alpha_1)} f(\alpha) \delta(d\alpha|\alpha_1) \right) \delta_1(d\alpha_1) \tag{8.17}$$

Equation 8.17 states the fact that the transition after an execution $\alpha_1$ can be expressed as the combination of the transitions after all the executions $\alpha$ in the counter-image of $\alpha_1$. By applying the inductive step we can substitute the measure $\delta_1$ by $\pi_1(\delta)$, since we are integrating only on executions of length $n-1$; we get:

$$= \int_{\alpha_1 \in \Lambda} \left( \int_{\alpha \in \pi_1^{-1}(\alpha_1)} f(\alpha) \delta(d\alpha|\alpha_1) \right) \pi_1(\delta)(d\alpha_1)$$

By using Theorem 2.14, we finally obtain:

$$
\begin{aligned}
&= \int_{\alpha \in \pi_1^{-1}(\Lambda)} f(\alpha)\delta(d\alpha) \\
&= \int_{\alpha \in \pi_1^{-1}(\Lambda)} \int_{t \in \mathcal{T}} I_A(a)\mu\left(\pi_1^{-1}(X)\right)\eta(\alpha)(dt)\delta(d\alpha) \\
&= \int_{\alpha \in \pi_1^{-1}(\Lambda)} \mu_{\eta(\alpha)}\left(A, \pi_1^{-1}(X)\right)\ \delta(d\alpha) \\
&= \delta\left(\pi_1^{-1}\left(C_{\Lambda A X}\right)\right)
\end{aligned}
$$

where the first equality is obtained by rewriting the function $f(\alpha)$, the second one by applying the definition of combined transition induced by a scheduler, and, finally, the last one by applying the definition of measure of a cone. This is what we had to prove for the inductive step.

Since we can define a measure for all basic sets, by applying Proposition 8.13, we get that $\eta$ is measurable. Also, since $\pi_1(\delta)$ and $\delta_1$ agree on all basic sets, which form a semi-ring, they agree on the whole $\sigma$-algebra $\mathcal{F}_{Exec}$ generated by them (Theorem 2.10). This completes the proof. □

Theorem 8.17 shows that the action of a scheduler on $S$ can be derived from the action of the corresponding schedulers on each component since the properties of an execution can be derived from the properties of its components. This allows us to analyse systems in a compositional way.

**Remark 8.2.** The definition of the projection of a scheduler (Equation 8.12) and the proof of Theorem 8.17 both respect our intuition and are not surprising. However, they are not completely straightforward as one has to take into account the issues that arise when considering continuous state spaces. As a consequence, the proof strictly depends on the condition that state spaces are analytic and it does not hold on general measure spaces.

**Remark 8.3.** Theorem 8.17 extends the analogue result for the discrete case [Seg95b]. In particular, Equation (8.12) can be rewritten in a more familiar form as:

$$
\eta_1(\alpha_1)(t) = \sum_{\alpha \in \pi_1^{-1}(\alpha_1)} \left( \delta(\alpha \mid \pi_1^{-1}(\alpha_1))\ \eta(\alpha)(\pi_1^{-1}(t)) \right)
$$

In the discrete case, we can define the probability for a single transition $t$. The equation above shows the intuition behind the definition of $\eta_1$: each transition is assigned the weighted probability of its inverse image under projection after each execution in the parallel composition, conditioned on being in an execution whose projection is $\alpha_1$.

## 8.5 Discussion

The main issue analysed in this chapter was the interaction between non-determinism and continuous state spaces and distributions. We showed that arbitrary schedulers can lead to intractable executions and defined the class of measurable schedulers that identify all and only "good" executions, on which it is possible to define a probability measure.

This problem was considered for Labelled Markov Processes [DGP02] for systems without non-determinism. However, we take a different approach and do not impose restrictions on the transition relation (this would not be trivial, if possible at all, with the addition of non-determinism), but rather restrict the possible executions of a system by limiting the power of schedulers.

# Chapter 9

# Equivalences for Stochastic Transition Systems

## 9.1 Introduction

In this chapter we extend the traditional trace and bisimulation equivalences of labelled transition systems to stochastic transition system.

The trace equivalence that we define is the natural extension to the continuous case of the corresponding notion for systems with discrete probabilities [Seg95b]: in this setting, the notion of trace defined for labelled transition systems and CSP is replaced by a probability distribution over the set of traces. In the case of stochastic transition systems, a trace distribution can be defined only under measurability conditions; therefore, the results and restrictions of the previous chapter are needed. As in the non-probabilistic case, trace semantics can be used for the specification and preservation of probabilistic safety properties.

We also define the notions of strong and weak bisimulation: initially defined in the context of the process algebra CCS [Mil89], bisimulation relations are arguably the most widely accepted notion of equivalence for concurrent processes, as it records the branching behaviour of a system, and, in its weak version, it can abstract from internal computation. Bisimulation has been adapted to discrete probabilistic systems: the first definition was found in [LS89a], where transitions are considered equivalent if their targets are equivalent probability distributions, that is, if the distributions assign the same probability to each equivalence class. Other variants based on this idea can be found: for example, [SL95] defines bisimulations for probabilistic automata under randomised schedulers showing it preserves the logic PCTL, and [HJ90] defines it for the alternating model. Several algorithms for the decision of many of the bisimulation relations for probabilistic systems have been presented, e.g. [PLS00, CS02, AW04]. Versions of strong bisimulation have been defined for systems with continuous state spaces [DEP98, D'A99, BD04, Bra02]; [Bra02] also defines a notion of weak bisimulation, even in a setting different from ours, as it uses a different definition of weak transitions.

When defining bisimulation, the use of continuous state spaces and distributions introduces new issues, again involving the problem of measurability of executions. In particular, we show that extending bisimulation in the natural way, that is, by matching the transitions enabled from bisimilar states, does not meet the usual soundness requirement that bisimulation implies trace equivalence. This is due to the fact that bisimulation is defined point-wise and therefore global properties like measurability are ignored. We propose an alternative notion of bisimulation based on hyper-transitions [Sto02], that is, transitions

defined from distributions over states to distributions over states and labels. This new notion of bisimulation is powerful enough to preserve trace equivalence and manages to distinguish systems that are not well behaved from well behaved systems.

We also consider a possible logical characterisation of bisimulation, discussing, in particular, how the results of [DEP98] for Labelled Markov Processes do not hold when non-determinism is introduced.

### Chapter Outline

In Section 9.2, we give stochastic transition systems a trace semantics in terms of trace distributions. In Section 9.3 we extend the notions of bisimulation to the continuous setting, thus obtaining local strong and weak bisimulation, and we show that this definition is inadequate. In Section 9.4 we define the notions of bisimulations based on hyper-transitions and also discuss how to deal with non-determinism when giving bisimulation a logical characterisation. Section 9.5 presents the main result of this chapter, relating bisimulation to trace equivalence. Finally, in Section 9.6, we discuss the semantic relations proposed.

## 9.2 Trace Semantics

The results of the previous chapter give us the mathematical tools needed in order to correctly give stochastic transition systems a trace semantics: in the probabilistic setting, a trace is a distribution over the set of possible traces [Seg95a]. Each way of resolving non-determinism results in a different trace distribution; the trace semantics of a system is given by the set of all its possible trace distributions.

Since in the continuous setting a single trace usually has probability zero, we define trace distributions on the $\sigma$-algebra of traces $\mathcal{F}_{Traces}$. According to the result of Proposition 8.13, each measurable probabilistic execution induces a measure over traces via the measure on cones. Therefore, also trace semantics is defined in terms of measurable schedulers and not all possible ways to resolve non-determinism give a trace distribution.

**Definition 9.1 (Trace Distribution).** Given a stochastic transition system $S$ and a probability measure $tr$ on $(traces^*, \mathcal{F}_{traces^*})$, we say that $tr$ is a trace distribution for $S$ if there exists a measurable scheduler $\eta$ such that

$$tr = trace(\delta_{\eta,\bar{q}}) \tag{9.1}$$

Finally, we can define the trace semantics for a stochastic transition system.

**Definition 9.2 (Trace Semantics).** The trace semantics for a stochastic transition system $S$ is defined as:

$$TraceDistr(S) = \{tr \mid \exists \eta \in \mathcal{A}_{meas} \text{ such that } tr = trace(\delta_{\eta,\bar{q}})\} \tag{9.2}$$

**Definition 9.3 (Trace Equivalence).** Two stochastic transition systems $S_1$ and $S_2$ are trace equivalent (denoted by $S_1 \equiv_T S_2$) if

$$TraceDistr(S_1) = TraceDistr(S_2) \tag{9.3}$$

**Remark 9.1.** The trace semantics for stochastic transition systems of Definition 9.2 coincides with that of probabilistic automata when we restrict to discrete $\sigma$-algebras, since in this case all schedulers are implicitly measurable. This semantics also coincides with the traditional CSP-style trace semantics for labelled transitions systems if we restrict to Dirac measures.

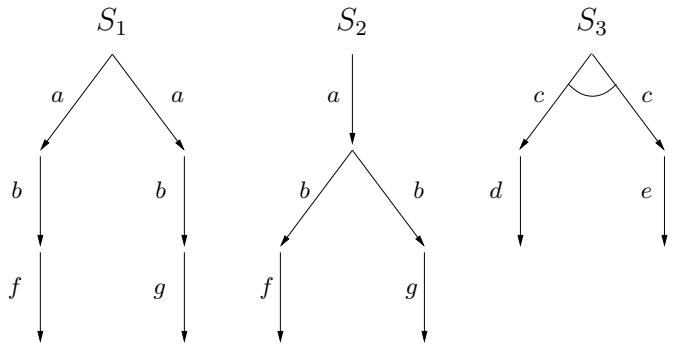



Figure 9.1: Trace semantics is not preserved by parallel composition.

It is known that trace distribution semantics is not a congruence with respect to parallel composition. This was shown for discrete probabilistic systems in [Seg95a], and the result carries over to our setting. The following example [Seg95a] explains this result.

**Example 9.1.** Consider the systems of Figure 9.1: $S_1$ and $S_2$ are trace equivalent, but their compositions with $S_3$ are not. This is because trace equivalence does not consider branching information, so in $S_1$ the execution of an $a$ action determines which of $f$ and $g$ is going to executed, while in $S_2$ this decision is postponed until $b$ is executed. Therefore, if the probabilistic choice of $S_3$ happens after the $a$ action, $S_2 \parallel S_3$ can decide which $b$ action of $S_2$ to perform.

For discrete probabilistic systems this problem is solved by introducing the notion of trace pre-congruence; however, for our purposes, we will focus on the relationship between the notion of trace semantics we have given and bisimulation, and on measurability issues.

## 9.3 Local Bisimulation

We study bisimulation relations under the control of randomised schedulers. The relations of this section are an extension of existing definitions for discrete models [LS89a, SL95, DGP02] to continuous and non-deterministic stochastic systems. We describe such notions of bisimulation as *local*, in order to highlight that they only consider at the point-wise behaviour of a system and to distinguish them from the definition of bisimulation that we give later. We show that such notions are inadequate since they do not preserve linear properties like trace semantics.

Given a stochastic transition system $S$ and an equivalence relation $\mathcal{R}$ on $(Q, \mathcal{F}_Q)$, we lift $\mathcal{R}$ to define an equivalence relation on the set of probability measures on $(Q, \mathcal{F}_Q)$ and on the set of probability measures on $(L \times Q, \mathcal{F}_{L\times Q})$. Define $X \in \mathcal{F}_Q$ to be $\mathcal{R}$-closed if it is the union of equivalence classes. Two probability measures $\mu_1$ and $\mu_2$ on $Q$ are $\mathcal{R}$-equivalent ($\mu_1 \mathcal{R} \mu_2$) if $\mu_1(X) = \mu_2(X)$ for all $\mathcal{R}$-closed $X \in \mathcal{F}_Q$, while two probability measures $\mu_1$ and $\mu_2$ on $Q \times L$ are $\mathcal{R}$-equivalent if $\mu_1(A, X) = \mu_2(A, X)$ for all $A \in \mathcal{F}_L$ and $\mathcal{R}$-closed $X \in \mathcal{F}_Q$.

The natural way to extend strong bisimulation to stochastic transition systems, similar to that of [BD04], requires that every pair of equivalent states enables equivalent transitions.

**Definition 9.4 (Local strong bisimulation).** Two stochastic transition systems $S_1$ and $S_2$ are *locally strongly bisimilar* (denoted by $S_1 \sim_l S_2$) if there exists an equivalence relation $\sim_l$ on the union of their sets of states such that

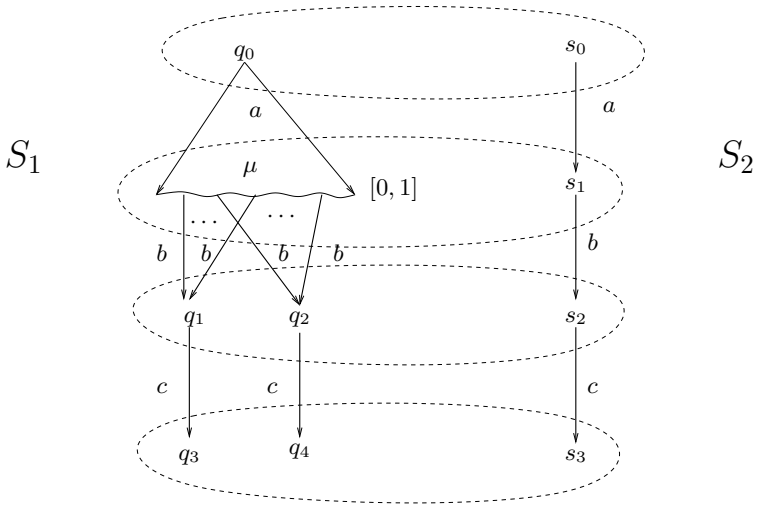



Figure 9.2: $S_1$ and $S_2$ are not trace equivalent but locally bisimilar.

1. $\overline{q}_1 \sim_l \overline{q}_2$ and
2. whenever $q_1 \sim_l q_2$ and $q_1 \to \mu_1$, there exists a combined transition $q_2 \to \mu_2$ such that $\mu_1 \sim_l \mu_2$.

We extend the notion of bisimulation to that of weak bisimulation, relating states only if they enable the same visible behaviour, that is, executions whose sequence of actions is of the form $\tau^* a \tau^*$.

**Definition 9.5 (Local weak bisimulation).** Two stochastic transition systems $S_1$ and $S_2$ are *locally weakly bisimilar* (denoted by $S_1 \approx_l S_2$) if there exists an equivalence relation $\approx_l$ on the union of their sets of states such that

1. $\overline{q}_1 \approx_l \overline{q}_2$ and
2. whenever $q_1 \approx_l q_2$ and $q_1 \to \mu_1$, there exists a weak transition $q_2 \Rightarrow \mu_2$ such that $\mu_1 \approx_l \mu_2$.

The definitions of local strong and weak bisimulation trivially respect the following basic property relating bisimulation relations.

**Theorem 9.6.** *Given two stochastic transition systems $S_1$ and $S_2$, if $S_1 \sim_l S_2$, then $S_1 \approx_l S_2$.*

*Proof.* The result comes from the fact that a strong transition is a special type of weak transitions. □

### 9.3.1 Local Bisimulation and Trace Semantics

The notions of bisimulation given above do not take into account measurability issues and are therefore inadequate for our purposes as they do not preserve linear properties of systems. This is best explained by the following example.

**Example 9.2.** Consider the systems of Figure 9.2: from the initial state of $S_1$, a measure on $[0,1]$ is enabled. Then, from a non-measurable subset of $[0,1]$, a Dirac transition on

$q_1$ labelled by $b$ is enabled; from its complement, a Dirac transition on $q_2$ labelled by $b$ is enabled. Finally, a Dirac transition on $q_3$ is enabled from $q_1$ and a Dirac transition on $q_4$ from $q_2$, both labelled by $c$. $S_2$ is very simple, and all of its transitions are Dirac. The dotted lines show the bisimulation relation (according to the definition above) between $S_1$ and $S_2$. Though the two systems are equivalent according to our intuition (every possible execution is labelled by $a$, $b$ and $c$), they are *not* trace equivalent since it is not possible to define a measure on executions (hence on traces) on $S_1$ because of its non-measurable behaviour after the first step.

The example above uses a system whose behaviour is somehow pathological, a result of our too "liberal" definition of transition relations for stochastic transition systems, and it might be argued that such systems should be disallowed and therefore not considered. We take a different position and allow for systems like that of the example above, but, instead, we define a notion of bisimulation that successfully distinguishes well behaved systems from badly behaved ones like that used above. Local bisimulation, though adequate for existing models, is not strong enough for this purpose; we need to define a notion of bisimulation that somehow takes into account the global behaviour. This is done by using the idea of hyper-transitions [Sto02].

## 9.4 Bisimulation

We define notions of bisimulation for stochastic transition systems that preserve measurability issues and trace semantics. We sometimes call them *global* strong and weak bisimulations to distinguish them from the definitions of the previous section and to stress their ability to consider the behaviour over sets of states, thus preserving measurability, which can be seen as a global property.

### 9.4.1 Strong Bisimulation

We extend the notion of transition and use the concept of *hyper-transition* [Sto02], adapting it to our continuous setting.

**Definition 9.7 (Hyper-transitions).** Let $\mu$ be a probability measure on $(Q, \mathcal{F}_Q)$ and for each $q \in Q$ let $q \to \mu_q$ be a combined transition. Define the measure $\mu'$ on states and labels as

$$\mu'(A, X) = \int_Q \mu_q(A, X)\mu(dq) \tag{9.4}$$

if the integral is well-defined for all $A \in \mathcal{F}_L$ and $X \in \mathcal{F}_Q$. Then we say that $\mu \to \mu'$ is a hyper-transition.

A hyper-transition is a transition that begins with a distribution over states and ends with a distribution over labels, and combines the transitions $q \to \mu_q$ according to the initial distribution $\mu$. Since $\mu'$ is defined only when the integral is well-defined, a hyper-transition is possible only under measurability conditions. For this reason, the definition of hyper-transitions is closely linked with the notion of measurable schedulers and measurable probabilistic executions: the integral of Equation 9.4 is well-defined if $\mu_q(A, X)$ is a measurable function from the set of states to $[0, 1]$. Informally, if the family of transitions $\{q \to \mu\}_{q \in Q}$ is chosen by a scheduler, the integral is well defined whenever the scheduler is measurable. We can now define our variant of strong bisimulation for stochastic transition systems.

**Definition 9.8 (Strong bisimulation).** Let $S_1$ and $S_2$ be two STSs with the same space of labels. An equivalence relation $\sim$ on the union of the set of states is a *strong bisimulation* between $S_1$ and $S_2$ if:

1. $\overline{q}_1 \sim \overline{q}_2$ and
2. for all $\mu_1$ and $\mu_2$ $\sim$-equivalent measures on either $Q_1$ or $Q_2$, whenever there is a hyper-transition $\mu_1 \to \mu_1'$, there exists a hyper-transition $\mu_2 \to \mu_2'$ such that $\mu_1' \sim \mu_2'$.

We say that two stochastic transitions systems $S_1$ and $S_2$ are strongly bisimilar if there exists a strong bisimulation between them, and denote it by $S_1 \sim S_2$.

**Theorem 9.9.** *Given two stochastic transitions systems $S_1$ and $S_2$, if $S_1 \sim S_2$ then $S_1 \sim_l S_2$.*

*Proof.* This result holds because a normal transition $q \to \mu$ can be seen as the special case of hyper-transition $dirac(q) \to \mu$. Whenever $q_1 \sim q_2$, since $dirac(q_1) \sim dirac(q_2)$, by hypothesis there exist two hyper-transitions $dirac(q_1) \to \mu_1$ and $dirac(q_2) \to \mu_2$ such that $\mu_1 \sim \mu_2$. Such hyper-transitions correspond to the transitions $q_1 \to \mu_1$ and $q_2 \to \mu_2$. Therefore $q_1$ and $q_2$ enable equivalent transitions, thus proving that $S_1 \sim_l S_2$. □

Consider again the two systems of Example 9.2: they are not strongly bisimilar according to our definition as we cannot define a hyper-transition where $a$ has probability 1, and therefore we cannot simulate the corresponding transition in $S_2$. We have thus succeeded in distinguishing such automata by imposing the preservation of measurability of paths.

**Remark 9.2.** Although our definition of bisimulation is stronger than local bisimulation, the two definitions coincide if we restrict our analysis to discrete systems. If we consider unlabelled Markov processes, whose transition relation is a Markov kernel, it is always possible to integrate over the transitions enabled from a measurable set of states, showing that a local bisimulation is a bisimulation as well. If we compare to Labelled Markov Processes, then the two definitions of bisimulation coincide when we restrict to the discrete $\sigma$-field over actions. Indeed, no $\sigma$-field on actions is defined for LMPs, and all the use of actions that is made is compatible with the discrete $\sigma$-field over actions.

### 9.4.2 Weak Bisimulation

We aim to derive a notion of weak bisimulation that relates states only if they enable the same visible behaviour, that is, executions whose sequence of actions is of the form $\tau^* a \tau^*$, where $\tau$ denotes internal computation [Mil89]. Weak bisimulation has been extended to probabilistic systems [BH97, PLS00, SL95]. In order to define weak bisimulation, we extend the notion of hyper-transitions to *weak hyper-transitions*, by considering the combination of weak transitions.

**Definition 9.10 (Weak hyper-transitions).** Let $\mu$ be a probability measure on $(Q, \mathcal{F}_Q)$ and for each $q \in Q$ let $q \Rightarrow \mu_q$ be a *weak* transition. Define the measure $\mu'$ on states and labels as

$$\mu'(A, X) = \int_Q \mu_q(A, X)\mu(dq) \tag{9.5}$$

if the integral is well-defined for all $A \in \mathcal{F}_L$ and $X \in \mathcal{F}_Q$. Then we say that $\mu \Rightarrow \mu'$ is a weak hyper-transition.

We can now define weak bisimulation for stochastic transition systems.

**Definition 9.11 (Weak bisimulation).** Let $S_1$ and $S_2$ be two STSs with the same space of labels. An equivalence relation $\approx$ on the union of the set of states is a weak bisimulation between $S_1$ and $S_2$ if:

1. $\overline{q}_1 \approx \overline{q}_2$ and
2. for all $\mu_1$ and $\mu_2$ $\approx$-equivalent measures on either $Q_1$ or $Q_2$, whenever there is a hyper-transition $\mu_1 \to \mu_1'$, there exists a weak hyper-transition $\mu_2 \Rightarrow \mu_2'$ such that $\mu_1' \approx \mu_2'$.

We say that two stochastic transitions systems $S_1$ and $S_2$ are weakly bisimilar if there exists a weak bisimulation between them, and denote it by $S_1 \approx S_2$.

**Theorem 9.12.** *Given two stochastic transitions systems $S_1$ and $S_2$, if $S_1 \approx S_2$ then $S_1 \approx_l S_2$.*

*Proof.* The proof follows the same arguments as the proof of Theorem 9.9. □

We now show that weak bisimulation is preserved through the parallel operator.

**Theorem 9.13.** *Let $S_1, S_2$ be STSs compatible with $S_3$. If $S_1 \approx S_2$ and the regular conditional probabilities with respect to the partitioning induced by $\approx$ exist, then we have $S_1 \,||\, S_3 \approx S_2 \,||\, S_3$.*

*Proof.* The proof follows standard arguments: consider the systems $S_1, S_2, S_3$, such that $S_1 \approx S_2$, and define $S_1' = S_1 \,||\, S_3$ and $S_2' = S_2 \,||\, S_3$. Define an equivalence relation $\approx$ on the union of the states of $S_1'$ and $S_2'$, denoted by $Q'$, as follows: $(q_1, q_1') \approx (q_2, q_2')$ if $q_1 \approx q_2$ and $q_1' = q_2'$. We need to check that $\approx$ is a bisimulation between $S_1'$ and $S_2'$.

- $\overline{q}_1' \approx \overline{q}_1'$, since $\overline{q}_1' = (\overline{q}_1, \overline{q}_3)$, $\overline{q}_2' = (\overline{q}_2, \overline{q}_3)$ and $\overline{q}_1 \approx \overline{q}_2$ by hypothesis.
- consider $\mu$, $\nu$ distributions on the union $Q'$ such that $\mu \approx \nu$. Consider a hyper-transition $\mu \to \mu'$. Then, for each state $(q_1, q_3)$ in $Q_1'$, there exists a transition $(q_1, q_3) \to \mu_{(q_1,q_3)}$ such that, for all $A \in \mathcal{F}_L$ and $Y \in \mathcal{F}_{Q'}$:

$$\begin{aligned}\mu'(A, Y) &= \int_{(q_1,q_3)} \mu_{(q_1,q_3)}(A, Y)\mu_1(d(q_1, q_3)) \\ &= \int_{X \in \mathcal{F}_{Q'}/\approx} \int_{(q_1,q_3)\in X} \mu_{(q_1,q_3)}(A, Y)\mu_1(d(q_1, q_3) \mid X)\mu_1(dX)\end{aligned}$$

  where the decomposition is possible thanks to the assumption that the regular conditional probabilities with respect to the quotient space exist, and to Theorem 2.14.

  Fix an equivalence class of states $X$; by definition of $\approx$, all the states in $X$ have the same component $q_3 \in S_3$. Also assume that $\mu$ and $\nu$ are measures on $Q_1'$ and $Q_2'$, respectively. Define a projection function $\pi_1$ that, given a transition $(q_1, q_3) \to \mu$, returns the left projection, that is, the transition $q_1 \to \pi_1(\mu)$, where the projection of the measure $\mu$ on $L \times Q_1'$ is defined in the standard way.

Given the equivalence class $X$, define the combined transition $q_3 \to \mu_{q_3,X}$ as the combination of all the right components leaving the states in $X$, for which, we recall, the right component is the same:

$$\mu_{q_3,X}(A,Y) = \int_{(q_1,q_3)\in X} \pi_3(\mu_{(q_1,q_3)})(A,\pi_3(Y))\mu_1(d(q_1,q_3) \mid X)$$

where $\pi_3$ denotes the right projection of a probability measure, returning a measure on $Q_3$. Observe that the probability measures on the parallel are defined by the product of their left and right projections, that is, for example:

$$\mu'(A,Y) = \pi_1(\mu')(A,\pi_1(Y))\pi_3(\mu')(A,\pi_3(Y))$$

Given an equivalence class $X_1$ of $Q_1 \cup Q_2/\approx$ (of the bisimulation between $S_1$ and $S_2$), consider the hyper-transition $\mu(\cdot \mid X_1) \to \mu'_X$, defined as:

$$\mu'_{X_1}(A,Y_1) = \int_{q\in X_1\cap X} \mu_{(q,q_3)}(A,\pi_1^{-1}(Y_1))\pi_1(\mu)(dq \mid X_1)$$

This measure describes the combination of the transitions leaving the states in $X_1$, after restricting to $X$. By bisimulation hypothesis, from each $q_2 \in X_1$, state of $Q_2$, there exists an equivalent transition $dirac(q_2) \to \nu_{q_2}$. From each state $(q_2,q_3) \in X \cap Q'_2$, define the transition $\nu_{(q_2,q_3)}$ as the combination of $\nu_{q_2}$ and $\mu_{q_3,X}$. It follows that $\nu_{(q_2,q_3)}(A,Y) = \nu_{q_2}(A,\pi_1(Y))\mu_{q_3,X}(A,\pi_3(Y))$. Let us analyse the hyper-transition obtained by combining the transitions $dirac(q_2,q_3) \to \nu_{(q_2,q_3)}$:

$$\nu'(A,Y) = \int_{(q_2,q_3)\in Q'_2} \nu_{(q_2,q_3)}(A,Y)\nu(d(q_3,q_3))$$

By using the hypothesis that regular conditional probabilities exist, we can apply Theorem 2.14 and obtain:

$$= \int_{X\in\mathcal{F}_{Q'}/\approx} \int_{(q_2,q_3)\in X} \nu_{(q_2,q_3)}(A,Y)\nu(d(q_2,q_3) \mid X)\,\nu(dX)$$

By applying the definition of $\nu_{(q_2,q_3)}$, and since $\mu_{q_3,X}$ is constant in a given equivalence class $X$, we get:

$$= \int_{X\in\mathcal{F}_{Q'}/\approx} \mu_{q_3,X}(A,\pi_3(Y)) \int_{(q_2,q_3)\in X} \nu_{q_2}(A,\pi_1(Y))\nu(d(q_2,q_3) \mid X)\nu(dX)$$

We can further divide the state space by considering the equivalence classes between $S_1$ and $S_2$ within $X$ and obtain:

$$= \int_{X\in\mathcal{F}_{Q'}/\approx} \mu_{q_3,X}(A,\pi_3(Y)) \int_{(X_1\in\mathcal{F}_Q/\approx)\cap X} \int_{(q_2,q_3)\in X_1} \nu_{q_2}(A,\pi_1(Y))\nu(d(q_2,q_3) \mid X \mid X_1)\nu(dX_1 \mid X)\nu(dX)$$

$\nu_{q_2}$ is constant inside an equivalence class $X_1$; moreover, $\mu$ and $\nu$ assign the same probability to equivalence classes, by hypothesis:

$$= \int_{X \in \mathcal{F}_{Q'}/\approx} \mu_{q_3,X}(A, \pi_3(Y)) \int_{(X_1 \in \mathcal{F}_Q/\approx) \cap X} \mu_{X_1}(A, \pi_1(Y)) \mu(dX_1 \mid X) \mu(dX)$$

By applying the definition of $\mu_{X_1}$, we can rewrite the inner integral in terms of the elements $(q_1, q_3)$ of $Q'_1$:

$$= \int_{X \in \mathcal{F}_{Q'}/\approx} \mu_{q_3,X}(A, \pi_3(Y)) \int_{(q_1,q_3) \in X} \mu_{(q_1,q_3)}(A, Y) \mu(d(q_1, q_3) \mid X) \mu(dX)$$

$$= \int_{X \in \mathcal{F}_{Q'}/\approx} \int_{(q_1,q_3) \in X} \mu_{q_3,X}(A, \pi_3(Y)) \pi_1(\mu_{(q_1,q_3)})(A, \pi_1(Y)) \mu(d(q_1, q_3) \mid X) \mu(dX)$$

By finally applying Theorem 2.14 again, we obtain:

$$= \int_{(q_1,q_3) \in Q'} \mu_{(q_1,q_3)}(A, Y) \mu(d(q_1, q_3))$$

$$= \mu'(A, Y)$$

This proves that $\nu'$ is $\approx$-equivalent to $\mu'$, which is what we had to show.

□

**Corollary 9.14.** *Let $S_1, S_2$ be STSs compatible with $S_3$. If $S_1 \approx S_2$ and the regular conditional probabilities with respect to the partitioning induced by $\approx$ exist, then we have $S_1 \,||\, S_3 \sim S_2 \,||\, S_3$.*

### 9.4.3 Discussion: a Logical Characterisation of Bisimulation?

In [DEP98], a Hennessy-Milner-style modal logic characterising bisimulation was proposed for Labelled Markov Processes (LMPs). Surprisingly, despite the fact that LMPs model continuous state spaces and arbitrary probability measures, the logic was extremely simple and, in particular, contained neither negation nor infinite conjunction.

We considered whether it is possible to have such a simple logic characterising (strong) bisimulation for stochastic transition systems. The answer is negative. This is because we model "pure" non-determinism, as opposed to the "probabilistic" non-determinism of LMPs (in our terminology, LMPs present no non-determinism at all). The need for both negation and infinite conjunctions is shown by using the same examples for standard labelled transition systems with infinite branching. Consider the two systems of Figure 9.3: they cannot be distinguished without the use of negation and a characterising formula is $\langle a \rangle \neg \langle b \rangle$, which is satisfied by $S_1$, but not by $S_2$. Figure 9.4 explains the need for infinite conjunction, where the notation $a^i$ denotes $i$ successive $a$ transitions: the two systems can be distinguished by the formula $\langle a \rangle \bigwedge_i \langle a \rangle^i \top$, but not by any formula with finitely many conjunctions.

The logic of [DEP98] works for the examples above since non-deterministic choice is replaced by probabilistic choice. For example, the systems of Figure 9.4 can be distinguished

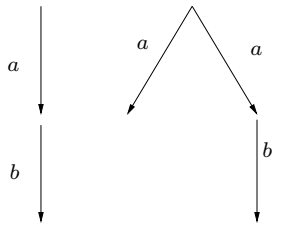
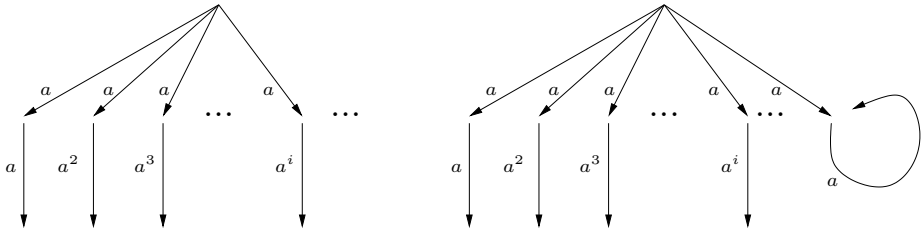



Figure 9.3: Two systems that cannot be distinguished without negation.


a
a
a

requirement.

The proof of this results follows a similar proof in [Seg95b], where probabilistic forward bisimulation was shown to preserve trace pre-congruence. We adapt the ideas to deal with our more general setting (including measurability issues) and with the fact the we use bisimulation. First, we introduce the auxiliary notions of fringes and execution correspondence structures.

### 9.5.1 Fringes

Informally, a *fringe* is a line that cuts a probabilistic execution in two parts, one before the fringe, and one after. A fringe is defined as a probability measure on finite execution (the executions defining the fringe) that agrees with the probability measure on cones induced by the probabilistic execution.

Given a set of finite executions $X_1 A_2 \ldots A_n X_n$, we say that an execution $\alpha$ extends the set of executions $X_1 A_2 \ldots A_n X_n$, denoted by $\alpha \geq X_1 \ldots X_n$ if there exists $\alpha' \in X_1 \ldots X_n$ such that $\alpha \geq \alpha'$.

**Definition 9.15 (Fringe).** A fringe for a probabilistic execution $P_\eta$ is a probability distribution $fr$ on $(Exec^*, \mathcal{F}_{Exec^*})$ such that for each sequence of measurable sets $X_1 \ldots X_i$ of $Exec^*$:

$$\int_{\alpha \geq X_1 A_2 \ldots A_i X_i} fr(d\alpha) \leq \delta_\eta(C_{X_1 A_2 \ldots A_i X_i})$$

We say that two fringes are in a $\leq$ relation, that is, $fr_1 \leq fr_2$ if

$$\int_{\alpha \geq X_1 \ldots X_i} fr_1(d\alpha) \leq \int_{\alpha \geq X_1 \ldots X_i} fr_2(d\alpha)$$

which informally means that the fringe $fr_1$ cuts the probabilistic execution before $fr_2$.

In particular, we define $fr(i, P_\eta)$ to be the fringe whose support is given by the executions of length $i$ and such that $fr(i, P_\eta)(X_1 A_2 \ldots A_i X_i) = \delta_\eta(C_{X_1 A_2 \ldots A_i X_i})$.

### 9.5.2 Execution Correspondence Structure

We need to define a structure that relates two probabilistic executions of bisimilar systems by relating the fringes at each level of one execution to the corresponding fringes of the other. The existence of this mapping is the key to proving that bisimulation implies trace equivalence.

Given an equivalence relation $\approx$ on states, we lift it to executions as follows: two distributions $\alpha_1$ and $\alpha_2$ are $\approx$ equivalent if $trace(\alpha_1) = trace(\alpha_2)$ and $lstate(\alpha_1) \approx lstate(\alpha_2)$. This also induces a relation $\approx$ on fringes: two fringes $fr_1$ and $fr_2$ are $\approx$-related if, for all $\approx$-closed sets of states and $C$ measurable sets of finite traces, we have that $fr_1(\{\alpha \in C \mid lstate(\alpha) \in X\}) = fr_2(\{\alpha \in C \mid lstate(\alpha) \in X\})$.

**Definition 9.16 (Execution Correspondence Structure).** Given a weak bisimulation relation $\approx$, an execution correspondence structure is a tuple $(P_1, P_2, m)$, where $P_1$ and $P_2$ are probabilistic executions and $m$ is a mapping from natural numbers to fringes of $P_2$, such that:

1. for each $i$, $m(i) \leq m(i+1)$;
2. for each $X_1 \ldots X_n$, $\lim_{i \to \infty} \int_{\alpha \geq X_1 A_2 \ldots A_n X_n} m(i)(dx) = \delta_2(C_{X_1 A_2 \ldots A_n X_n})$;

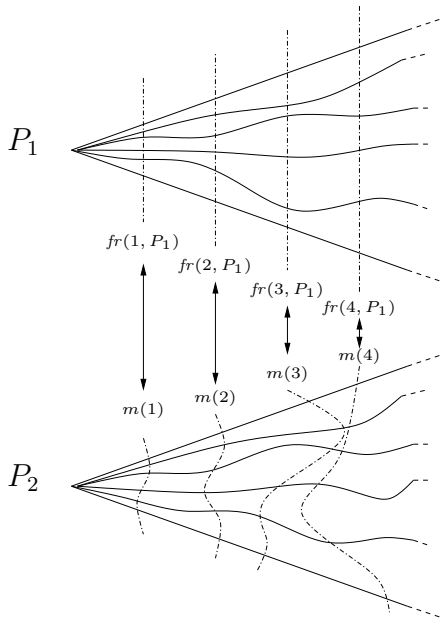



Figure 9.5: An execution correspondence structure.

3. $fr(i) \approx m(i)$.

The definition above requires the existence of a succession of fringes of $P_2$ that are in $\leq$ relation (Condition 1), that eventually capture each cone (Condition 2), and that are $\approx$ related to the fringes of $P_1$ at each level (Condition 3).

### 9.5.3 The Main Result

We can finally prove our main result. We start by proving that if two systems are weakly bisimilar, then we can find an execution correspondence structure between them.

Define $[\cdot]_\approx$ to be the function that, given an execution $\alpha$, it returns the equivalence class with respect to $\approx$ to which $\alpha$ belongs. In order to prove this result, we assume that the regular conditional probabilities for $\delta_i$, $i = 1, 2$, measures on cones induces by the probabilistic execution, with respect to $[\cdot]_\approx$ exist.

**Proposition 9.17.** *Given two stochastic transition systems $S_1$ and $S_2$, a weak bisimulation $\approx$ between them, and a probabilistic execution $P_1 = (Exec^*, \mathcal{F}_{Exec^*}, \mu_1)$ of $S_1$, if the regular conditional probabilities for $\delta_i$, $i = 1, 2$, with respect to $[\cdot]_\approx$ exist, then there exist a probabilistic execution $P_2 = (Exec^*, \mathcal{F}_{Exec^*}, \mu_2)$ and a mapping $m$ from natural numbers to fringes of $P_2$ such that $(P_1, P_2, m)$ is an execution correspondence structure.*

*Proof outline.* Given $P_1$, we build $P_2$ by matching each level of $P_1$ with the corresponding hyper-transitions.

- The first level of $P_1$ is defined by a transition leaving the start state of $S_1$, $\overline{q}_1$. This is matched by a weak transition leaving $\overline{q}_2$; this transition also defines the first fringe $m(i)$ for $P_2$; $m(1) \approx fr(1, P_1)$ by bisimulation hypothesis.
- Assume we have defined $P_2$ up to the $i$-th fringe $m(i)$. Such fringe is equivalent to $fr(i, P_1)$ and therefore they induce equivalent distributions on states. The hyper-transition leaving $tr(i, P_1)$, whose target is the fringe $fr(i + 1, P_1)$, can be matched

by an equivalent weak hyper-transition leaving $m(i)$ and defining $m(i+1)$. Since the systems are bisimilar, these hyper-transitions match and it follows that $m(i+1) \approx fr(i+1, P_1)$.

Figure 9.5 explains the idea behind this proof. Conditions 1 and 3 of Definition 9.16 are easy to check, while the fact that every cone of $P_2$ is eventually captured comes from the fact that we are building $P_2$ and therefore we only define its reachable part by including it in each successive fringe. □

*Proof.* The proof proceeds as follows:

1. We define an extended structure, easier to deal with, that we use to define the probabilistic execution $P_2$. This structure uses an extra state $\mathsf{S}$ that denotes that the systems had stopped at this state for a certain fringe. These states are used to define the fringes and denote the points from which to extend the probabilistic execution while constructing the next fringe. This structure is then mapped to a probabilistic execution of $S_2$ in a way that preserves fringes and the measure on executions.

2. We build $P_2$ and the fringes on $P_2$ inductively by matching the transitions of $P_1$ with weak transitions in $P_2$. While we define each fringe $m(i)$, we also prove it to be equivalent to $fr(i, P_1)$, that is, $m(i) \approx fr(i, P_1)$.

3. We check that each $m(i)$ that we have defined is indeed a fringe for $P_2$.

4. We check that each fringe $m(i)$ comes before the fringe $m(i+1)$, that is, $m(i) \leq m(i+1)$.

5. Finally, we check that each cone is eventually captured, that is, Condition 2 of Definition 9.16.

This completes the proof. Let us show each step in detail.

1. Firstly, we extend the state space of $S_2$ with an extra state $\mathsf{S}$. We use this state to make explicit reference to the fact that a probabilistic execution has stopped with some non-zero probability. We build a probability execution $\mathcal{P}_2$ on this extended state space and define a mapping $\pi_{\mathsf{S}}$ that returns a probabilistic execution $\pi_{\mathsf{S}}(\mathcal{P}_2)$ of $S_2$ by ignoring this extra state. We assume that the new state $\mathsf{S}$ is always preceded by some action not in $L$, so that it does not interfere with the normal behaviour of the probabilistic execution.; since this action is not relevant for this proof, we always omit it. Extend the function *lstate* on finite executions of $P_2$ so that it returns the last non-$\mathsf{S}$ state.

   The projection function $\pi_{\mathsf{S}}$ takes an execution of $P_2$ and returns and execution of $\pi_{\mathsf{S}}(\mathcal{P}_2)$. $\pi_{\mathsf{S}}$ is a measurable function: consider a measurable set of executions $X_1 \ldots X_n$ in $\mathcal{F}_{Exec^*}$; its counter-image is given by ${\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n) = X_1\mathsf{S}^* \ldots X_n\mathsf{S}^*$, where the operator $^*$ denotes the countable union of concatenations: $X_1\mathsf{S}^* \ldots X_n\mathsf{S}^* = \cup_{i_1} \cdots \cup_{i_n} X\mathsf{S}^{i_1} \ldots X_n\mathsf{S}^{i_n}$. Given the transition function $\omega_2$ of $\mathcal{P}_2$, we build the transition function $\pi_{\mathsf{S}}(\omega_2)$ of $\pi_{\mathsf{S}}(P_2)$ as follows:

$$\pi_{\mathsf{S}}(\omega)(\alpha)(X) = \int_{\alpha' \in Exec^*{}_{\mathsf{S}}} \omega(\alpha')(X) Q(\alpha, d\alpha')$$

where $Exec^*{}_{\mathsf{S}}$ denotes the finite execution of the extended state space including $\mathsf{S}$, and where $Q(\alpha, d\alpha')$ is the conditional probability function for $\delta_2$, measure on the cones of $P_2$ induced by $\omega_2$ with respect to $\pi_{\mathsf{S}}$, whose existence is guaranteed by Theorem 2.12. We prove that the measure on cones $\pi_{\mathsf{S}}(\delta_2)$ induced by $\pi_{\mathsf{S}}(\omega_2)$ is the same as the projection of the measure on cones $\delta_2$ induced by $\omega_2$, that is, $\delta_2({\pi_{\mathsf{S}}}^{-1}(X)) = \pi_{\mathsf{S}}(\delta_2)(X)$ for all measurable sets of executions $X$. We use the same technique of the proof of Theorem 8.17 and we prove the result by induction. The base case is:

$$\begin{aligned}
\pi_{\mathsf{S}}(\delta_2)(X) &= \begin{cases} 1 & \text{if } \overline{q}_2 \in X \\ 0 & \text{otherwise} \end{cases} \\
\delta_2(\pi^{-1}(X)) &= \delta_2(\cup_{i\geq 0} X\mathsf{S}^i) \\
&\geq \delta_2(X) \\
&= \begin{cases} 1 & \text{if } \overline{q}_2 \in X \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Inductive case:

$$\begin{aligned}
\pi_{\mathsf{S}}(\delta)(X_1 \ldots X_n) &= \int_{\alpha \in X_1..X_{n-1}} \pi_{\mathsf{S}}(\mu)(X_n)\pi_{\mathsf{S}}(\delta)(d\alpha) \\
&= \int_{\alpha \in X_1..X_{n-1}} \int_{\alpha' \in {\pi_{\mathsf{S}}}^{-1}(\alpha)} \mu(X_n) Q(\alpha, d\alpha')\pi_{\mathsf{S}}(\delta)(d\alpha) \\
&= \int_{\alpha \in {\pi_{\mathsf{S}}}^{-1}(X_1..X_{n-1})} \mu(X_n)\delta(d\alpha) \\
&= \delta({\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n))
\end{aligned}$$

by Theorem2.14. The projection function $\pi_{\mathsf{S}}$ also preserves fringes, that is, if $fr$ is a fringe for $\mathcal{P}_2$, then $\pi_{\mathsf{S}}(fr)$ is a fringe for ${\pi_{\mathsf{S}}}'$. This is shown as follows:

$$\begin{aligned}
\pi_{\mathsf{S}}(\delta_2)(C_{X_1 \ldots X_n}) &= \delta_2({\pi_{\mathsf{S}}}^{-1}(C_{X_1 \ldots X_n})) \\
&\geq \int_{\alpha \geq {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n)} fr(d\alpha) \\
&= fr(\{\alpha \geq {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n)\})
\end{aligned}$$

Note that $\{\alpha \geq {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_1)\} = {\pi_{\mathsf{S}}}^{-1}(\{\alpha \geq X_1 \ldots X_n\})$: consider an execution $\alpha$ in the first set, then there exists $\alpha' \in {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n)$ and $\alpha \geq \alpha'$. This implies that $\pi_{\mathsf{S}}(\alpha) \geq \pi_{\mathsf{S}}(\alpha') \in X_1 \ldots X_n$. It follows that $\alpha \in {\pi_{\mathsf{S}}}^{-1}(\alpha') \subseteq {\pi_{\mathsf{S}}}^{-1}(\{\alpha \geq X_1 \ldots X_n\})$. For the opposite inclusion, consider an execution $\alpha \in {\pi_{\mathsf{S}}}^{-1}(\{\alpha \geq X_1 \ldots X_n\})$; there must exist an execution $\alpha' \in X_1 \ldots X_n$ such that $\pi_{\mathsf{S}}(\alpha) \geq \alpha'$. $\alpha \geq {\pi_{\mathsf{S}}}^{-1}(\alpha')$ follows from the fact that $\alpha'$ itself is in ${\pi_{\mathsf{S}}}^{-1}(\alpha')$. Therefore, $\alpha \in \{\alpha \geq {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_1)\}$.

We finally get the following:

$$\begin{aligned}
fr(\{\alpha \geq {\pi_{\mathsf{S}}}^{-1}(X_1 \ldots X_n)\}) &= fr({\pi_{\mathsf{S}}}^{-1}(\{\alpha \geq X_1 \ldots X_n\})) \\
&= \pi_{\mathsf{S}}(fr)(\{\alpha \geq X_1 \ldots X_n\}) \\
&= \int_{\alpha \geq X_1 \ldots X_n} \pi_{\mathsf{S}}(fr)(d\alpha'')
\end{aligned}$$

which proves that $\pi_{\mathsf{S}}(fr)$ is a fringe for ${\mathcal{P}_2}'$.

We use this modified structure for two reasons: firstly, it makes the construction by induction of the probabilistic execution $\mathcal{P}_2$, easier, since each state either stops with probability one or does not stop. In this way, we can append new trees from terminating executions in a straightforward manner, without having to combine them with existing transitions leaving such states. Secondly, it keeps track of where the system had stopped at previous levels, thus making it easier to define the measure induced by new fringes in terms of the previous fringe.

2. We now build the fringes $m(i)$ of $\mathcal{P}_2$. We denote the fringes $\mathit{fringe}(i, P_1)$ by $\mathit{fringe}(i)$ since it is not going to create confusion. We define $m(i)$ by induction, by constructing a probabilistic execution $\mathcal{P}_2^i = (\mathit{Exec}^*{}_{\mathsf{S}}, \mathcal{F}_{\mathit{Exec}^*{}_{\mathsf{S}}}, \omega_2^i)$ for each step of $P_1$. $P_2$ is defined as the limit of all the $\mathcal{P}_2^i$'s.

   - Base case: we match the transition leaving the starting state of $P_1$ with a weak transition of $S_2$. Such weak transition corresponds to a probabilistic execution that defines $\mathcal{P}_2^1$.

     Let $\mathit{dirac}(\overline{q}_1) \rightarrow \mu$ be the hyper-transition enabled from the start state of $S_1$. Since $S_1$ and $S_2$ are weakly bisimilar, there exists a weak hyper-transition $\mathit{dirac}(\overline{q}_2) \Rightarrow \mu'$ such that $\mu \approx \mu'$. Let ${\mathcal{P}_2^1}' = (\mathit{Exec}^*, \mathcal{F}_{\mathit{Exec}^*}, {\omega_2^1}')$ be the probabilistic execution generating the weak transition leaving $\overline{q}_2$; this defines the first level of $\mathcal{P}_2$: $\omega_2^1(\alpha, AX) = {\omega_2^1}'(\alpha, AX)$ for all executions $\alpha$ and measurable sets of states and labels $AX$; in addition, whenever ${\omega_2^1}'(\alpha, LQ) < 1$, we define $\omega_2^1(\alpha, \mathsf{S}) = 1 - {\omega_2^1}'(\alpha, LQ)$. Denote the measure on cones of executions by $\delta_2^1$; it is clear that $\delta_2^1(\mathit{Exec}^*) = 1$, since it is generated by the probabilistic execution corresponding to a weak transition. Define $m(1)$ as follows:

     $$m(1)(X_1 \ldots X_n \mathsf{S}) = \delta_2^1(C_{X_1 \ldots X_n \mathsf{S} \bot})$$

     Note that $\delta_2^1(X_1 \ldots X_n \bot) = 0$ if $S \notin X_n$, since we have defined $\omega_2^1$ so that it stops only after $\mathsf{S}$ states. The first fringe of $P_1$ coincides with the distribution leaving the initial state $\overline{q}_1$:

     $$\mathit{fringe}(1)(AX) = \delta_1(C_{AX}) = \mu_1(\overline{q}_1)(AX)$$

     We need to show that $\mathit{fringe}(1) \approx \pi(m(1))$, that is, for all $T \in \mathcal{F}_{\mathit{traces}^*}$ and $X \in \mathcal{F}_Q$:

     $$\begin{aligned} &\mathit{fringe}(1)(\{\alpha \mid \mathit{trace}(\alpha) \in T \ \wedge \ \mathit{lstate}(\alpha) \in X\}) \\ &= \ \pi_{\mathsf{S}}(m(1))(\{\alpha \mid \mathit{trace}(\alpha) \in T \ \wedge \ \mathit{lstate}(\alpha) \in X\}) \end{aligned}$$

     We only need to consider one visible action in the trace, as $\mathit{fringe}(1)$ is non zero only for cones of the type $AX$:

     $$\mathit{fringe}(1)(AX) = \mu_1(\overline{q}_1)(AX)$$

     By bisimulation hypothesis

     $$= \delta_2^1{}'(C_{Q(AX)^* X \bot} \cap \mathcal{W}_A)$$

This set can be rewritten as the countable union of basic sets by using the $^*$ operator:

$$= \delta_2^{1\prime}(C_{Q(L^\tau Q)^*(AQ)(L^\tau Q)^* L^\tau X \perp})$$

By considering the extended probabilistic execution, we get:

$$= \delta_2^1(C_{Q(L^\tau Q)^*(AQ)(L^\tau Q)^* L^\tau X \mathsf{S} \perp})$$

By definition of $m(1)$ this is equal to:

$$\begin{aligned} &= m(1)(Q(L^\tau Q)^*(AQ)(L^\tau Q)^* L^\tau X \mathsf{S} \perp) \\ &= \pi_{\mathsf{S}}(m(1))(Q(L^\tau Q)^*(AQ)(L^\tau Q)^* L^\tau X \perp) \\ &= \pi_{\mathsf{S}}(m(i))(\{\alpha \mid \mathit{trace}(\alpha) \in A \ \wedge \ \mathit{lstate}(\alpha) \in X\}) \end{aligned}$$

which is what we needed to show to prove that $\mathit{fringe}(1) \approx \pi_{\mathsf{S}}(m(1))$. A trivial consequence is that the two distributions restricted to the last state are $\approx$-equivalent.

- Inductive case: assume we have already defined the fringe corresponding to the $i$-th level: we have a fringe $m(i)$ of $P_2$, a transition function $\omega_2^i$ and the corresponding measure on cones $\delta_2^i$, such that $m(i)(X_i \dots X_n \mathsf{S}) = \delta_2^i(X_1 \dots X_n \mathsf{S} \perp)$.
  For each equivalence class $H$ of the equivalence relation $\approx$ lifted to executions (two executions are equivalent if their last states are $\approx$-equivalent and if they have the same trace), define a probability measure $\delta(\cdot \mid H)$ as the conditional probability on executions conditioned on being in $H$. This is true if, for example, $Q/\approx$ is analytic; without this assumption, we have no guarantee on the well-definedness of this conditional probability and we would not be able to construct $\mathcal{P}_2$.
  We build the $i$+1-th fringe starting from these objects. By induction hypothesis, we also have that $\pi(m(i)) \approx \mathit{fringe}(i)$. We extend each terminating execution of $\mathcal{P}_2^i$ with a weak transition equivalent to the combination of the transitions leaving the equivalent executions in $P_1$. Formally, for each $\alpha \in \mathit{support}(m(i))$, consider the set of executions $H_\alpha$ of $P_1$ defined as $H_\alpha = \{\alpha' \in \mathit{support}(\mathit{fringe}(i)) \mid \alpha' \approx \pi_{\mathsf{S}}(\alpha)\}$. It is clear that $\mathit{dirac}(\alpha) \approx \delta(\cdot \mid H_\alpha)$. Consider the hyper-transition $\delta(\cdot \mid H_\alpha) \to \mu_{H_\alpha}$, where $\mu_{H_\alpha}$ is defined as

$$\mu_{H_\alpha}(AX) = \int_{\alpha' \in Exec^*} \mu_1(\alpha')(AX)\delta(d\alpha' \mid H_\alpha) \tag{9.6}$$

  The integral above is well defined because we use measurable schedulers. This defines a hyper-transition from the $i$-th fringe to the $i+1$-th fringe of $P_1$.
  By bisimulation hypothesis, there exists a weak hyper-transition $\mathit{dirac}(\alpha) \Rightarrow \mu'_\alpha$ such that $\mu'_\alpha \, R \, \mu_{H_\alpha}$. Let $P_\alpha = (Exec^*, \mathcal{F}_{Exec^*}, \omega_\alpha)$ be the probabilistic execution inducing $\mu'_\alpha$.
  $\mathcal{P}_2^i$ is extended by "appending" the probabilistic executions $P_\alpha$ corresponding to each execution $\alpha$ from its last state: $\omega_2^{i+1}(\alpha\alpha')(AX) = \omega_\alpha(\alpha', AX)$ and $\omega_2^{i+1}(\alpha\alpha')(\mathsf{S}) = 1 - \mu_\alpha(\alpha', LQ)$, for all $\alpha \in \mathit{support}(m(i))$, and $\omega_2^{i+1}(\alpha) = \omega_2^i(\alpha)$ for all other executions. The use of the extra state $\mathsf{S}$ makes this definition much

simpler as we do not have to combine the new probabilistic executions leaving a state with the existing ones.

We have identified a hyper transition leaving the $i$-th fringe of $P_1$ and a weak hyper transition leaving the $i$-th transition of $P_2$. The former is defined as $\mathit{lstate}(\mathit{fringe}(i)) \to \mu$, with $\mu$ defined as:

$$\begin{aligned}\mu(AX) &= \int_{\alpha \in Exec^*} \mu_1(\alpha)(AX)\mathit{fringe}(i)(d\alpha) \\ &= \int_{H \in Exec^*/\approx} \int_{\alpha \in H} \mu_1(\alpha)(AX)\mathit{fringe}(i)(d\alpha \mid H)\mathit{fringe}(i)(dH) \\ &= \int_{H \in Exec^*/\approx} \mu_H(AX)\mathit{fringe}(i)(dH)\end{aligned}$$

where the first equality is obtained by applying Theorem 2.14 with respect to the regular conditional probability for the projection on the quotient space $Exec^*/\approx$, and the second equality follows from the definition of $\mu_H$ of Equation 9.6. The weak hyper-transition $\mathit{lstate}(m(i)) \Rightarrow \mu'$ is defined as follows, where the equalities are justified as above:

$$\begin{aligned}\mu'(AX) &= \int_{\alpha \in Exec^*} \mu'_\alpha(AX)m(i)(d\alpha) \\ &= \int_{H \in Exec^*/\approx} \int_{\alpha \in H} \mu'_\alpha(AX)m(i)(\alpha \mid H)m(i)(dH) \\ &= \int_{H \in Exec^*/\approx} \mu'_H(AX)m(i)(dH)\end{aligned}$$

Since $\mathit{fringe}(i) \approx m(i)$ by induction hypothesis, the two measures agree on all $H \in Exec^*/\approx$, and since we have defined $\mu'_H$ to be the target distribution equivalent to $\mu_H$, we have that $\mu$ and $\mu'$ agree on all $AX$, and therefore $\mu \approx \mu'$, that is, they are two equivalent steps in the bisimulation.

The support of $m(i+1)$ is defined as the set of executions that now end with state $\mathsf{S}$; $m(i+1)$ is defined as follows:

$$m(i+1)(X_1 \dots X_n) = \delta_2^{i+1}(C_{X_1 \dots X_n \perp})$$

Note that only sets of executions such that $\mathsf{S} \in X_n$ have non zero probability.

It is useful to express the probability of some set of executions in the $i+1$-th fringe of $P_2$ in terms of the probability of reaching the $i$-th fringe and the probability of the probabilistic executions corresponding to the weak transitions leaving each state in the $i$-th fringe. Formally, assume $\mathit{support}(m(i)) \leq X_1 \dots X_n$, that is, for every execution in the support of $m(i)$, there exists an execution in $X_1 \dots X_n$ extending it. Then we want to prove the following:

$$\delta_2^{i+1}(X_1 \dots X_n) = \int_{\alpha \in Exec^*} \delta'_{\mathit{lstate}(\alpha)}(X_1 \dots X_n \mid \alpha)m(i)(d\alpha) \tag{9.7}$$

where $\delta_{lstate(\alpha)}(X_1 \dots X_n \mid \alpha)$ denotes the probability of sets of execution in $X_1 \dots X_n$ that extend $\alpha$ (or, informally, the probability of $X_1 \dots X_n$ given $\alpha$, hence the notation). To prove this, we partition the set $X_1 \dots X_n$ according to which level the last $\mathsf{S}$ state was encountered. Since we consider sets of executions that extend the support of $m(i)$, there must be an $\mathsf{S}$ state in every execution; such $\mathsf{S}$ state denote the point where the computation had stopped at the previous level.

$$X_1 \dots X_n = \bigcup_{j=0}^{n} \left(X_1 \dots X_{j-1}\{\mathsf{S}\}X'_{j+1} \dots X'_n\right) \tag{9.8}$$

where $X'_i = X_i \backslash \{\mathsf{S}\}$. Let us consider the probability $\delta_2^{i+1}(X_1 \dots X_j\{\mathsf{S}\}X'_{j+1}X'_n)$ of each individual set: this can be rewritten as the probability of reaching the states at level $j$ and then ending with the state $\mathsf{S}$, product the probability of the set of executions that extend this point.

$$\begin{aligned}
\delta_2^{i+1}(C_{X_1 \dots X_j\{\mathsf{S}\}X'_{j+1}X'_n}) &= \int_{\alpha \in X_1 \dots X_j\{\mathsf{S}\}} \delta_{lstate(\alpha)}(C_{X'_{j+1} \dots X'_n})\delta_2^{i+1}(d\alpha) \\
&= \int_{\alpha \in X_1 \dots X_j\{\mathsf{S}\}} \delta_{lstate(\alpha)}(C_{X'_{j+1} \dots X'_n})\delta_2^{i}(d\alpha)
\end{aligned}$$

Where the last equality is justified by the fact that up to the last occurrence of $\mathsf{S}$, $\omega_2^i$ and $\omega_2^{i+1}$ agree and therefore induce the same measure on cones. By definition of $m(i)$, we get:

$$\delta_2^{i+1}(X_1 \dots X_j\{\mathsf{S}\}X'_{j+1}X'_n) = \int_{\alpha \in X_1 \dots X_{j-1}\{S\}} \delta_{lstate(\alpha)}(X_1 \dots X_n \mid \alpha)m(i)(d\alpha)$$

Since the decomposition of $X_1 \dots X_n$ of Equation 9.8 partitions the whole set, the probability of the $X_1 \dots X_n$ is given by the sum of the probability of each component, and the result above generalises to Equation 9.7.

We immediately use this result to show that this new probabilistic execution still terminates with probability 1 and therefore $m(i+1)$ defines a probability measure:

$$\begin{aligned}
\delta_2^{i+1}(Exec^*) &= \int_{\alpha \in support(m(i))} \delta_{lstate(\alpha)}(Exec^*)\delta_2^{i}(d\alpha) \\
&= \int_{\alpha \in support(m(i))} 1 \; \delta_2^{i}(d\alpha) = 1
\end{aligned}$$

The first integral is well defined under the assumption that the weak hyper-transition is well defined. Since $\delta_{lstate(\alpha)}$ is the measure on cones induced by the weak transition leaving $lstate(\alpha)$, it terminates with probability 1. The last

equality is justified by the fact that $\delta_2^i$ terminates with probability 1 on the support of $m(i)$.

We now show that the fringe $\pi_{\mathsf{S}}(m(i+1))$ is $\approx$-equivalent to $\mathit{fringe}(i+1)$ under the inductive assumption that $\pi_{\mathsf{S}}(m(i)) \approx \mathit{fringe}(i)$. Consider a measurable set of traces $T = T'A$ and a measurable set of states $X$; the probability of the set of executions whose trace is in $T$ and whose last state is in $X$ can be decomposed into the set of executions whose trace is in $T$ and that only perform internal actions after the previous stopping point (that is, after the fringe $m(i)$), and the set of traces that execute one external action in $A$ after $m(i)$:

$$\begin{aligned}
&\pi_{\mathsf{S}}(m(i+1))(\{\alpha \mid \mathit{trace}(\alpha) \in T \wedge \mathit{lstate}(\alpha) \in X\}) \\
&\quad = m(i+1)(\{\alpha \mid \mathit{trace}(\alpha) \in T \wedge \mathit{lstate}(\alpha) \in X\}) \\
&\quad = \int_{\alpha \in Exec^*} I_T(\alpha)\ \mu'_{lstate(\alpha)}(A^\tau X) m(i)(d\alpha) + \\
&\qquad + \int_{\alpha \in Exec^*} I_{T'}(\alpha)\ \mu'_{lstate(\alpha)}(AX) m(i)(d\alpha)
\end{aligned}$$

Let us consider each part separately:

$$\begin{aligned}
&\int_{\alpha \in Exec^*} I_{T'}(\alpha)\ \mu'_{lstate(\alpha)}(AX) m(i)(d\alpha) \\
&\quad = \int_{H \in \mathcal{F}_{Exec^*}/\approx} \left( \int_{\alpha \in H} I_{T'}(\alpha) \mu'_{lstate(\alpha)}(AX) \delta(d\alpha \mid H) \right) m(i)(dH)
\end{aligned}$$

by decomposing the state space according to Theorem 2.14. Given an equivalence relation $H$ of $\mathcal{F}_{Exec^*}/\approx$, $\mu_{lstate(\alpha)}$ can be rewritten as $\mu'_H$ as it is constant for all $\alpha \in H$, by construction of the weak transitions leaving each state in the support of $m(i)$. $I_{T'}(\alpha)$ is constant for all $\alpha \in H$, and therefore it can be rewritten as $I_{T'}(H)$. We can rewrite the term above as:

$$\begin{aligned}
&= \int_{H \in \mathcal{F}_{Exec^*}/\approx} I_{T'}(H) \mu'_H(AX) \left( \int_{\alpha \in H} \delta(d\alpha \mid H) \right) m(i)(dH) \\
&= \int_{H \in \mathcal{F}_{Exec^*}/\approx} I_{T'}(H) \mu'_H(AX) m(i)(dH)
\end{aligned}$$

$\mu_H = \mu'_H$ (where $\mu_H$ is defined in Equation 9.6) since they are target measures of equivalent transitions by construction. In addition $m(i)(H) = \mathit{fringe}(i)(H)$ by induction hypothesis:

$$= \int_{H \in \mathcal{F}_{Exec^*}/\approx} I_{T'}(H) \mu_H(AX) \mathit{fringe}(i)(dH)$$

by applying the definition of $\mu_H$ of Equation 9.6, we obtain:

$$= \int_{H \in \mathcal{F}_{Exec^*}/\approx} \left( I_{T'}(\alpha) \int_{\alpha \in H} \omega(\alpha, AX)\delta(d\alpha \mid H) \right) \mathit{fringe}(i)(dH)$$

and, by applying Theorem 2.14 again:

$$= \int_{\alpha \in Exec^*} I_{T'}(\alpha)\omega(\alpha, AX)\mathit{fringe}(i)(d\alpha)$$

The following equality is proved in the same way:

$$\int_{\alpha \in Exec^*} I_T(\alpha)\mu'(\alpha)(A^\tau X)m(i)(d\alpha) = \int_{\alpha \in Exec^*} I_T(\alpha)\mu(\alpha)(A^\tau X)\mathit{fringe}(i)(d\alpha)$$

We finally get the following:

$$\begin{aligned} &\int_{\alpha \in Exec^*} I_T(\alpha)\ \mu'_\alpha(A^\tau X)m(i)(d\alpha)\ + \int_{\alpha \in Exec^*} I_{T'}(\alpha)\ \mu'_\alpha(AX)m(i)(d\alpha) \\ &\quad = \int_{\alpha \in Exec^*} I_T(\alpha)\mu(\alpha)(A^\tau X)\mathit{fringe}(i)(d\alpha) \\ &\qquad + \int_{\alpha \in Exec^*} I_{T'}(\alpha)\mu(\alpha)(AX)\mathit{fringe}(i)(d\alpha) \end{aligned}$$

This last term denotes the sets of executions in $\mathit{fringe}(i)$ whose trace is given by $T$ and whose last state is in $X$. Again, we have partitioned these executions between those whose last action is silent and those whose last action is visible and in $A$. We finally get the following equality

$$\begin{aligned} &\pi_{\mathsf{S}}(m(i+1))(\{\alpha \mid \mathit{trace}(\alpha) \in T \ \wedge \mathit{lstate}(\alpha) \in X\}) = \\ &\quad = \mathit{fringe}(i+1)(\{\alpha \mid \mathit{trace}(\alpha) \in T \ \wedge \mathit{lstate}(\alpha) \in X\}) \end{aligned}$$

that proves that $\pi_{\mathsf{S}}(m(i+1)) \approx \mathit{fringe}(i)$ and completes the construction of the inductive step.

We have obtained a sequence of probabilistic executions $\mathcal{P}_2^i$, for all $i$, with transition relation $\omega_2^i$. Finally, define $\omega_2(\alpha, AX) = \lim_{i \to \infty} \omega_2^i(\alpha, AX)$. This identifies $P_2$.

3. While building the successive levels of $P_2$, we proved that each $m(i)$ is equivalent to the corresponding $\mathit{fringe}(i)$. This demonstrates condition 3 of Definition 9.16. We still have to verify the other two conditions, and also that all the $m(i)$'s defined above are indeed fringes of $P_2$.

   Let us start from this last point: we have to prove that for all $\int_{\alpha \geq X_1 \ldots X_n} m(i)(d\alpha) \leq X_1 \ldots X_n,\ \delta_2(C_{X_1 \ldots X_n})$. The set of executions that are contained in the support of

$m(i)$ and that extend the executions in $X_1 \dots X_n$ is given by $X_1 \dots X_n (LQ)^* \mathsf{S}\bot$, as all non terminating executions have probability zero.

$$\begin{aligned}\int_{\alpha \geq X_1 \dots X_n} m(i)(d\alpha) &= m(i)(X_1 \dots X_n Q^* \mathsf{S}\bot) \\ &= \delta_2^i(\cup_{i\geq 0} X_1 \dots X_n (LQ)^i \mathsf{S}\bot)\end{aligned}$$

By construction of the fringes $m(i)$ and of the measures $\delta_2^i$, only the finite executions ending in $\mathsf{S}$ have non zero probability. Therefore, the term above can be rewritten as:

$$= \delta_2^i(X_1 \dots X_n)$$

By applying the definition of measure of basic sets, we obtain:

$$\begin{aligned}&= \int_{\alpha \in X_1 \dots X_{n-1}} \omega_2^i(X_n)\ \delta_2^i(d\alpha) \\ &\leq \int_{\alpha \in X_1 \dots X_{n-1}} \omega_2(X_n)\ \delta_2(d\alpha) \\ &= \delta_2(X_1 \dots X_n)\end{aligned}$$

The inequality is justified by the fact that, for all $i$, $\omega_2^i \leq \omega_2$; this comes from the construction of the fringes, as $\omega_2^{i+1}$ is never less that $\omega_2^i$. As a consequence, $\delta_2^i \leq \delta_2$.

4. We use similar arguments to show that $m(i) \leq m(i+1)$: since $\omega_2^i \leq \omega_2^{i+1}$ for all $i$, we can prove that $m(i)(X_1 \dots X_n (LQ)^* \mathsf{S}\bot) = \int_{\alpha \geq X_1 \dots X_n} m(i)(d\alpha) \leq \int_{\alpha \geq X_1 \dots X_n} m(i+1)(d\alpha) = m(i+1)(X_1 \dots X_n (LQ)^* \mathsf{S}\bot)$, for all $X_1 \dots X_n$, that is, $m(i) \leq m(i+1)$, which is condition 2 of Definition 9.16.

5. Finally, we show that all cones are eventually captured, that is, for all $X_1 \dots X_n$,

$$\lim_{i \to \infty} \int_{\alpha \geq X_1 \dots X_n} m(i)(d\alpha) = \delta_2(C_{X_1 .. X_n})$$

By induction on the length of basic sets, we show that $\delta_2^i \to \delta_2$:

- The base case is trivial, as $\delta_2(C_X) = 1$ if $\overline{q}_2 \in X$, and 0 otherwise and ll the $\delta_2^i$ are defined in the same way.
- For the inductive step, assume that $\delta_2^i \to \delta_2$ for all basic sets of length $n-1$. Then, consider the basic set$X_1 \dots X_n$:

$$\delta_2^i(C_{X_1 \dots X_n}) = \int_{\alpha \in X_1 \dots X_{n-1}} \omega_2^i(\alpha)(X_n) \delta_2^i(d\alpha)$$

We have $\omega_2^i(\alpha)(X) \to \omega_2(\alpha)(X)$ by definition of $\omega_2$ and $\delta_2^i(d\alpha) \to \delta_2(d\alpha)$ by induction. It follows that $\delta_2^i(C_{X_1 \dots X_n}) \to \delta_2(C_{X_1 \dots X_n})$. Finally, $m(i)(X_1 \dots X_n) = \delta_2^i(X_1 \dots X_n) \to \delta_2(X_1 \dots X_n)$.

This completes the proof and shows that if two stochastic transition systems are weakly bisimilar, then there exists an execution correspondence structures between them for all probabilistic executions of each.

□

**Corollary 9.18.** *Given an execution correspondence structure, we have* $trace(m(i)) = trace(fringe(i))$*, for all* $i$*.*

*Proof.* $m(i) \approx fringe(i)$ means that both distributions give the same probability to all measurable sets of traces and $\approx$-closed sets of states. Clearly, it can be induced that the two measures coincide if lifted to measures on traces as

$$\begin{aligned} trace(m(i))(T) &= m(i)(T, Q) \\ &= fringe(i)(T, Q) \\ &= trace(fringe(i))(T) \end{aligned}$$

□

We can now prove the main result of this section.

**Theorem 9.19.** *Given two stochastic transition systems* $S_1$ *and* $S_2$*, if there exists a weak bisimulation relation* $\approx$ *such that* $S_1 \approx S_2$*, then they have the same trace distributions, under the same conditions of Proposition 9.17.*

*Proof.* Let $P_1 = (Exec^*, \mathcal{F}_{Exec^*}, \mu_1)$ be a probabilistic execution of $S_1$ leading to a trace distribution $tr_1$. By applying Proposition 9.17, there exists a probabilistic execution $P_2 = (Exec^*, \mathcal{F}_{Exec^*}, \mu_2)$ and a mapping $m$ from natural numbers to fringes of $P_2$ such that $fringe(i, P_1) \approx m(i)$, for all $i$, that is, $(P_1, P_2, m)$ is an execution correspondence structure for $\approx$. We need to show that $P_2$ induces the same trace distribution as $P_1$.

Consider a measurable set of traces $T$. The probability of $T$ in $P_1$ is given by the probability of the set of executions whose trace is in $T$:

$$\delta_1(T) = \int_{\alpha \in Exec | trace(\alpha) \in T} \delta_1(d\alpha)$$

This value can be rewritten as the limit of the set of executions of length $i$ whose trace extends $T$:

$$\delta_1(T) = \lim_{i \to \infty} \int_{\alpha \in Exec^* || \alpha | = i \wedge trace(\alpha) \geq T} \delta_1(d\alpha)$$

We can apply the notion of fringes for each length $i$ of the executions:

$$= \lim_{i \to \infty} \int_{\alpha \in Exec^* | trace(\alpha) \geq T} fringe(i, P_1)(d\alpha)$$

Let us considerer now the probability of $T$ under $P_2$:

$$\delta_2(T) = \int_{\alpha \in Exec^* | trace(\alpha) \in T} \delta_2(d\alpha)$$

Note that $T$ corresponds to a measurable set of executions (Theorem 8.7), and, as such, it can be described in terms of the generators of the $\sigma$-algebra on executions. This allows us to use the second condition of the definition of execution correspondence structure, which implies that this set is eventually captured by the fringes $m(i)$:

$$\delta_2(T) \;=\; \lim_{i\to\infty} \int\limits_{\alpha\in Exec^*|trace(\alpha)\geq T} m(i)(d\alpha)$$

We need to prove the following equality:

$$\int\limits_{\alpha\in Exec^*|trace(\alpha)\geq T} fringe(i,P_1)(d\alpha) = \int\limits_{\alpha\in Exec^*|trace(\alpha)\geq T} m(i)(d\alpha) \tag{9.9}$$

The set of executions $\{\alpha \in Exec^* \mid trace(\alpha) \geq T\}$ corresponds to the set $T'$ of traces that extend $T$. This set is measurable as it is the countable union of the sets of traces that extend $T$ with any sequence of actions. The fact that $trace(fringe(i,P_1))(T') = trace(m(i))(T')$, that is, that they are assigned the same probability by both the fringes $fringe(i,P_1)$ and $m(i)$, follows from Corollary 9.18 and proves Equation 9.9. □

The following result is an easy consequence of the previous theorem, since a strong bisimulation is also a weak bisimulation.

**Corollary 9.20.** *Given two stochastic transition systems $S_1$ and $S_2$, if there exists a strong bisimulation relation $\sim$ such that $S_1 \sim S_2$, then $S_1$ and $S_2$ induce the same set of trace distributions, under the same conditions of Proposition 9.17.*

### 9.5.4 Summary

In this section, we have extended the standard result relating bisimulation and traces semantics, showing that if two systems are bisimilar, then they induce the same trace distributions. In our continuous setting, though, the result depends on measurability conditions: firstly, we needed to extend the notion of bisimulation so that it would not only consider point-wise behaviour, but also what we called global behaviour. Secondly, we imposed the restriction that the quotient of the sets of states with respect to the bisimulation equivalence relations allows for regular conditional probabilities to exist; we do not know if this latter condition is strictly necessary in order to have the result, or if it is possible to prove the result in a different way without this assumption.

Note that in the discrete case both the conditions above hold implicitly, and therefore the result we have shown for stochastic transition systems extends to models like probabilistic automata that are encapsulated in our model.

The restrictions we have imposed were all concerned with ruling out behaviour that would result in intractable executions; this was caused by the fact that we wanted our model to be as general and permissive as possible, and we have placed restrictions along the way as needed. An alternative approach would be to place restriction at the source, by limiting the model, possibly by restricting the possible transition relations. It would be interesting to find the restrictions under which local and global bisimulations coincide, so that local bisimulation suffices to preserve trace equivalence.

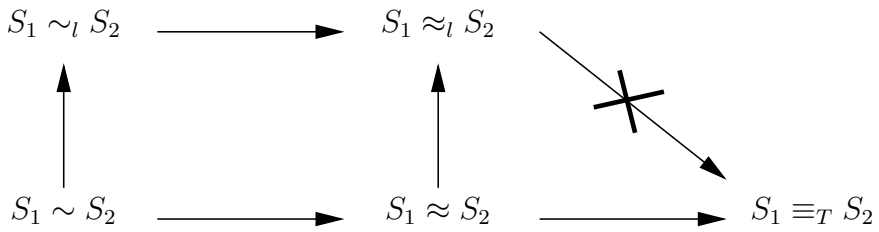



Figure 9.6: The relationship between the equivalences for stochastic transition systems

## 9.6 Discussion

In this chapter, we have introduced several notions of equivalence for stochastic transition systems. We have observed that the continuous nature of the state spaces must be taken into account in order to define relations that extend the equivalent notions for discrete systems and that preserve the usual requirement as expected. In particular, trace semantics can be defined only by restricting to measurable schedulers and the usual notion of bisimulation cannot be defined point-wise if we want it to preserve trace equivalence. Figure 9.6 shows how the equivalences defined in this chapter relate to one another.

# Chapter 10

# A Stochastic Process Algebra

## 10.1 Introduction

In this chapter, we introduce a process algebra for real-time systems with stochastic behaviour. It can be seen an extension of the process algebra of Part I, where clocks are not necessarily reset to zero, but according to arbitrary continuous distributions. We also allow for discrete probabilistic choice among different processes.

This process algebra, called Stochastic CSP, or SCSP, extends CCSP and also draws ideas from timed automata, using clocks to record the passage of time and influence the evolution of a system. SCSP is similar to other existing algebras; for example, the stochastic process algebra *Spades* [D'A99] also uses clocks, even if in a different way, since in this case their values decrease down to 0, at which point actions can be triggered. See the discussion of Chapter 7 for an overview of related models.

SCSP terms are mapped to stochastic transition systems via a set of operational rules, and they can represent both probabilistic and non-deterministic behaviour. Probabilistic behaviour is introduced by the discrete probabilistic choice operator and by the stochastic reset of clocks, while non-determinism is introduced by the choice operators and by the fact that a system can non-deterministically choose how long to let time elapse before performing an action (within the constraints of guards and invariants). This justifies the need for a model like stochastic transition systems capable of representing arbitrary probability distributions and uncountable non-determinism. We study this process algebra from a rather different angle than that for CCSP: we are not concerned with having decidable relations leading to model checking algorithms. Instead, we give it semantics in terms of the equivalence relations of Chapter 9, by extending the corresponding notions to SCSP. The usefulness of SCSP is as a specification formalism that can be evaluated through simulations. Not surprisingly, trace equivalence and refinement are not congruences for SCSP, making them unsuitable for compositional reasoning and thus prompting for further research in this direction. On the other hand, bisimulation equivalences are compositional under the restrictions of our language. We do not study more advanced aspects like axiomatisation or possible applications, but we highlight possible developments that could arise from this approach.

### Chapter Outline

In Section 10.2 we introduce the process algebra SCSP and, in Section 10.3, we give it an operational semantics in terms of stochastic transition systems. In Section 10.4, we define several equivalence relations for SCSP based on the results of the previous chapter.

$$
\begin{array}{lcl}
T & ::= & STOP \mid T \underset{A}{||} T \mid \phi \triangleright T \mid \mathsf{rec}\, Y.T \mid Y \\
 & & \bigsqcap_{i=1}^{N} P_i \mid \underset{A}{\Box}_{i=1}^{N} (a_i, \varphi_i) \to P_i \mid P \setminus A \\
P & ::= & \Delta(T) \mid P \oplus_q P \mid \{\!|X|\!\} P
\end{array}
$$

Table 10.1: Syntax of Stochastic CSP

## 10.2 The Language

We introduce the stochastic process algebra called Stochastic CSP (SCSP). This language is inspired by CSP, with internal and external choice and multi-way parallel operator. SCSP also extends Clocked CSP, since, like CCSP, it augments a traditional process algebra by the addition of clocks and operators that manipulate them, and it also provides for the description of probabilistic behaviour. This comes in two variants: discrete probabilistic choice among processes, and continuous distributions controlling the setting of the values of clocks.

### 10.2.1 Syntax

Fix a (countable) set of actions, denoted by $\Sigma$ (not including the special silent action $\tau$), and a finite set of clocks $\mathcal{C}$. For each clock $x$, let $\mu_x$ be the probability measure on the set of the reals equipped with the corresponding Borel $\sigma$-algebra, $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$, associated to $x$; every time $x$ is reset, its possible new value is distributed according to $\mu_x$. Let $\Sigma^\tau$ denote the set of actions including the silent one, that is, $\Sigma \cup \{\tau\}$.

The stochastic process algebra SCSP for the alphabet $\Sigma$ is described by the syntax of Table 10.1, where $q \in \mathbb{R}$, $0 \leq q \leq 1$, $\phi \in \mathcal{B}_c(\mathcal{C})$, $X \subseteq \mathcal{C}$, $a \in \Sigma$, and $A \subseteq \Sigma$. Denote the set of all possible terms generated by the syntax above by **SCSP**.

The $T$ process terms describe the non-probabilistic behaviour of a system, and we find many of the usual CSP operators here:

- $STOP$ denotes the deadlocked process, that cannot interact with other processes, but can let time elapse;
- $\phi \triangleright T$ denotes the invariant operator and forces process $T$ to perform an action while the invariant $\phi$ is true;
- $\bigsqcap_i P_i$ denotes internal choice, resolved instantly by the execution of a internal action;
- $\Box_i (a_i, \varphi_i) \to P_i$: this choice is resolved by the execution of an action $a_i$, possible only when the corresponding guard $\varphi_i$ is satisfied;
- $\mathsf{rec}\, Y.T$ denotes the recursive term, binding the process variable $Y$ to the term $T$.
- $P \setminus A$ behaves like process $T$, with the actions in $A$ hidden and transformed into $\tau$ actions;
- $T \underset{A}{||} T$ represents parallel composition; the two term must synchronise on the actions in $A$, while they can execute the remaining actions independently.

The $P$ process terms describe the probabilistic behaviour; this is described by three operators:

- discrete probabilistic choice ($\oplus_q$): the process $P_1 \oplus_q P_2$ behaves like $P_1$ with probability $q$, and like $P_2$ with probability $1 - q$;
- arbitrary probability distribution: the process $\{\!|X|\!\}P$ resets all the clocks in $X$ according to their probability distribution, while leaving all the others unchanged;
- $\Delta(T)$ represents the process that behaves like $T$ with probability 1.

The behaviour of a SCSP process is similar to that of a CCSP process or of a timed automaton with the addition of probabilities: a process is evaluated under some clock condition and all clocks increase at the same rate as real time. Time can elapse only while invariants are satisfied and transitions are guarded by conditions on clocks; the target of transitions is a probability distribution on processes and clock valuations. Stochastic transition systems are the concrete operational model for SCSP. This is made formal in Section 10.3.1.

#### Parallel Composition and Clocks

As for CCSP, we require that processes joined by the parallel operator have disjoint sets of clocks. This is imposed in order to achieve compositionality: if a process changes the value of some clocks, this action would affect the behaviour of the other components.

We denote the set of clocks used by a process $T$ by $\mathcal{C}(T)$; this corresponds to the idea of internal clocks of CCSP, and the rules of Table 4.3 can be used to define the set $\mathcal{C}(T)$ inductively. Since we do not aim at a compositional denotational semantics, it is not necessary to define the set of external clocks as in CCSP. When we give a process an operational semantics, its state space is given by the product of the set of possible process terms and the possible valuations of its clocks.

### 10.2.2 What We Model

Although we map SCSP terms directly onto infinite state stochastic transition systems, it can be useful to give them an informal graphical representation in terms of a timed automata-like structure. The non-probabilistic operators behave as in the non-probabilistic case, so let us analyse the effect of the probabilistic operators. Consider, for example, the process

$$P = (a, \phi) \to \{\!|X|\!\}(\Delta(T_1) \oplus_{1/2} (\{\!|X'|\!\}\Delta(T_2)))$$

Such a process can be graphically described by Figure 10.1: the target of the $a$ transition is the probability distribution that resets the clocks in $X$ and then with probability $1/2$ becomes the process $T_1$, and with probability $1/2$ resets the clocks in $X'$ and becomes $T_2$.

This model clearly resembles timed automata with the addition of discrete probabilistic choice on locations and stochastic clock resets. It is also similar to the model of stochastic automata [D'A99], with the addition of discrete probabilistic choice and with the difference that in our case the values of clocks advance rather than decrease to zero.

**Remark 10.1.** CCSP terms correspond to the subset of SCSP terms such that $\mu_x = \mathit{dirac}(0)$, that is, to the set of processes where all clocks are always reset to zero by means of a Dirac distribution on zero. Since CCSP processes are complete for timed automata

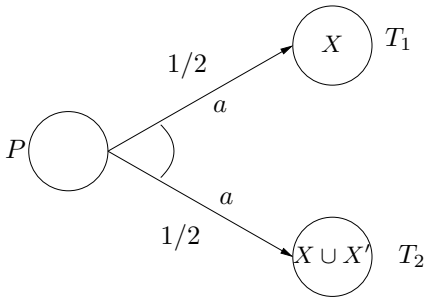



Figure 10.1: A graphical representation of a SCSP term.

(Theorem 4.5), timed automata can be modelled using such a deterministic subset of SCSP.

Standard CSP terms (with the modified external choice operator as in Table 10.1) are also a subset of SCSP and they correspond to the subset of processes with the empty set of clocks, and only the operator $\Delta(\cdot)$ among the probabilistic operators. In this case, the notion of time becomes irrelevant.

## 10.3 Operational Semantics

We give SCSP terms an operational semantics in terms of stochastic transition systems. The state space is generated by the cross product of the set of possible SCSP terms and the possible clock valuations (Definition 4.1). Assume $|\mathcal{C}(T)| = n$, then the state space of the operational model is given by $Q = \mathbf{SCSP} \times \mathbb{R}^n$. The first set, **SCSP**, is countable, being generated by the syntax of Table 10.1, while $\mathbb{R}^n$ is analytic. It follows that the whole state space is analytic, being the cross product of analytic spaces. Likewise, the alphabet $L$ of the operational model is analytic, since it is given by the cross product of the countable alphabet $\Sigma$ and the non-negative reals, representing the delay preceding the execution of an action. The set of internal actions $L^\tau$ is composed by the single element $(\tau, 0)$, since there must always be synchronisation on non-zero delays, even if followed by an internal action.

We overload the probabilistic operators of SCSP to work on probability measures. Assume an implicit numbering of the clocks, that is, $\mathcal{C} = \{x_1, \ldots, x_n\}$. Given a probability measure $\mu$ on $Q$, let $\mu(i)$ be the projected measure on the $i$-th clock induced by $\mu$, and $\mu(0)$ be the projected measure on the space of process terms. Note that $\mu(0)$ is a probability measure on a countable set, so it makes sense to talk about the probability of a single element. We define the reset of a probability measure $\mu$ for a subset $X$ of clocks as:

$$\{\!|X|\!\}\mu = \otimes_{i=0}^{n} \mu'(i) \tag{10.1}$$

where $\mu'(0) = \mu(0)$ and $\mu'(i) = \mu_{x_i}$ if $x_i \in X$, $\mu(i)$ otherwise. $\{\!|X|\!\}\mu$ is the distribution that agrees with $\mu$ on the distribution of all the clocks that are not in $X$; the clocks in $X$ are distributed according to their own distribution. We also extend the discrete probabilistic operator to probability measures. $\mu_1 \oplus_q \mu_2$ is defined as follows, for all $Y \in \mathcal{F}_Q$:

$$(\mu_1 \oplus_q \mu_2)(Y) = q\mu_1(Y) + (1-q)\mu_2(Y) \tag{10.2}$$

Finally, we define the parallel composition of two measures $\mu_1$ and $\mu_2$. When working with parallel processes, each component has a different state space, given by the valuations

| Probabilistic Operators |
|---|
| $\dfrac{}{(\Delta(T), \nu) \longmapsto dirac(T, \nu)}$ |
| $\dfrac{(P_1, \nu) \longmapsto \mu_1 \quad (P_2, \nu) \longmapsto \mu_2}{(P_1 \oplus_q P_2, \nu) \longmapsto \mu_1 \oplus_q \mu_2}$ |
| $\dfrac{(P, \nu) \longmapsto \mu}{(\{\!\lfloor X \rfloor\!\} P, \nu) \longmapsto \{\!\lfloor X \rfloor\!\} \mu}$ |

Table 10.2: Operational semantics rules for SCSP for probabilistic terms.

on the set of clocks of the component. Therefore, the composition of two measures results in a new measure whose state space is given by the set of processes and the set of valuations on the union of the sets of clocks. Given two measures $\mu_1$ and $\mu_2$ on $\mathbf{SCSP} \times \mathbb{R}^{n_1}$ and $\mathbf{SCSP} \times \mathbb{R}^{n_2}$, where $n_1$ and $n_2$ denote the number of clocks of each component, define the parallel composition $\mu_1 \underset{A}{||} \mu_2$ as the measure $\mu$ on $\mathbf{SCSP} \times \mathbb{R}^{n_1+n_2}$ defined by:

$$\mu(T_1 \underset{A}{||} T_2, Y_1 \times Y_2) = \mu_1(T_1, Y_1) \cdot \mu_2(T_2, Y_2) \tag{10.3}$$

for all $Y_1, Y_2$ measurable sets of $\mathbb{R}^{n_1}$ and $\mathbb{R}^{n_2}$, respectively. This definition exploits the fact that the sets of clocks are disjoint. Informally, $\mu_1 \underset{A}{||} \mu_2$ is the measure that resets a clock $x_i$ if either component has reset it, and that combines the probabilities on process terms by combining terms via the parallel operator.

### 10.3.1 SOS Rules

We define the operational rules for each SCSP operator, first considering the probabilistic operators, then the standard CSP operators and, finally, the parallel operator.

- Table 10.2 shows the rules for the probabilistic operators. Informally, a probabilistic term $P$ is mapped onto a probability measure on the space of processes and clock valuations by "collapsing" the different clock valuations. This defines a probabilistic transition relation, denoted by $\longmapsto$, that is used to define the behaviour of the other operators.

  Note that the $\oplus$ operator describes a probabilistic choice between processes, and therefore it defines the probability distribution on the **SCSP** part of the state space. The operator $\{\!\lfloor \cdot \rfloor\!\}$, on the other hand, defines the probabilistic distribution on the space of clock valuations.

- Table 10.3 shows the operational rules for all the operators, except for parallel composition. The rules are not surprising and are similar to the rules given for CCSP, with the difference that probability measures are now the target of transitions. Note, in particular, that the hiding operator only hides the action part of a transition, and not its delay part; for this reason, hiding does not always result in an internal action in the operational model. Actions of the type $(\tau, d)$, $d > 0$, cannot be considered internal in the usual way as systems cannot execute them independently, or else components lose synchronisation.

STANDARD OPERATORS

$$\frac{(T,\nu) \xrightarrow{a,d} \mu \quad \forall d' \leq d\ \nu + d' \models \phi}{\phi \triangleright T \xrightarrow{a,d} \mu}$$

$$\frac{(P_k,\nu) \longmapsto \mu}{(\bigsqcap_{i=1}^{N} P_i,\nu) \xrightarrow{\tau,0} \mu}\ k \in \{1..N\}$$

$$\frac{(P_k,\nu+d) \longmapsto \mu \quad \nu + d \models \phi_k}{(\Box_{i=1}^{N}(a_i,\phi_i) \to P_i,\nu) \xrightarrow{a_k,d} \mu}\ k \in \{1..N\}$$

$$\frac{(P,\nu) \xrightarrow{a,d} \mu}{(P \setminus A,\nu) \xrightarrow{a,d} \mu \setminus A}\ a \notin A \qquad \frac{(P,\nu) \xrightarrow{a,d} \mu}{(P \setminus A,\nu) \xrightarrow{\tau,d} \mu \setminus A}\ a \in A$$

$$\frac{T \xrightarrow{a,d} \mu}{\mathsf{rec}\, X.T \xrightarrow{a,d} \mu[X/\mathsf{rec}\, X.T]}$$

Table 10.3: Operational semantics rules for SCSP's standard operators.

- Finally, the behaviour of the parallel operator is shown in Table 10.4. This is dealt with separately as we have introduced a new type of transition $\overset{d}{\rightsquigarrow}$, denoting the willingness of a process to let time elapse for some amount of time. Such action is needed in order to make sure that processes in the parallel composition always agree on the clock valuations. For instance, if process $P_1$ wants to execute action $a$ after a delay $d$, (action $(a,d)$ in the operational model), it is necessary that $P_2$ allows a delay of length $d$ in order for their parallel composition $P_1 \underset{A}{||} P_2$ to be able to execute $(a,d)$. Given a valuation $\nu$ on the union of the clocks of two components $T_1$ and $T_2$, denote by $\nu(1)$ and $\nu(2)$ the restriction of $\nu$ on the clocks of $T_1$ and $T_2$, respectively.

  The parallel composition of two processes results in a stochastic transition system whose state space is different from the state spaces of the components: while the components only consider the valuations on their respective sets of clocks, the new state space considers the valuations on the union of the sets of clocks of the components.

We can now define the operational semantics of SCSP terms.

**Definition 10.1 (Operational Semantics).** Given a SCSP process $T$, its operational semantics is given by the stochastic transition system $S(T) = ((Q,\mathcal{F}_Q),\overline{q},(L,\mathcal{F}_L),\to)$, where:

- $Q = \mathbf{SCSP} \times \mathbb{R}^n$ and $\mathcal{F}_Q$ is the corresponding product $\sigma$-algebra, where $n = |\mathcal{C}(T)|$.
- $\overline{q} = (T,\nu_0)$.
- $L = (\Sigma \cup \{\tau\}) \times \mathbb{R}^{\geq 0}$, with the corresponding product $\sigma$-algebra. The set of internal actions $L^\tau$ contains the single element $(\tau,0)$.

PARALLEL OPERATOR

$$\frac{}{(\Box_i(a_i,\phi_i)\to P_i,\nu)\overset{d}{\rightsquigarrow}} \qquad \frac{\forall d'\le d\ \ \nu+d'\models\phi}{(\phi\triangleright T,\nu)\overset{d}{\rightsquigarrow}}$$

$$\frac{(T_1,\nu(1))\xrightarrow{a,d}\mu_1 \qquad (T_2,\nu(2))\overset{d}{\rightsquigarrow}}{(T_1\underset{A}{\mid\mid}T_2,\nu)\xrightarrow{a,d}\mu_1\underset{A}{\mid\mid}dirac(T_2,\nu+d)}\ a\notin A$$

$$\frac{(T_1,\nu(1))\overset{d}{\rightsquigarrow} \qquad (T_2,\nu(2))\xrightarrow{a,d}\mu_2}{(T_1\underset{A}{\mid\mid}T_2,\nu)\xrightarrow{a,d}dirac(T_1,\nu+d)\underset{A}{\mid\mid}\mu_2}\ a\notin A$$

$$\frac{(T_1,\nu(1))\xrightarrow{a,d}\mu_1 \qquad (T_2,\nu(1))\xrightarrow{a,d}\mu_2}{(T_1\underset{A}{\mid\mid}T_2,\nu)\xrightarrow{a,d}\mu_1\underset{A}{\mid\mid}\mu_2)}\ a\in A$$

Table 10.4: Operational semantics rules for the parallel operator.

- $\to$ is the transition relation generated by the rules of Tables 10.2, 10.3 and 10.4.

We prove that the parallel composition of two SCSP terms can result in the same operational model as the one obtained by composing the operational models of the components. This result serves both as a sanity check on the operational semantics, and also allows to import the compositionality result of Theorems 8.17 and 9.13.

**Theorem 10.2.** *The operational semantics is preserved by parallel composition.*

*Proof.* Firstly, assume we can distinguish the actions not in $A$: since the parallel composition of stochastic transition systems requires synchronisation on all common actions, we need to be able to distinguish which component executed which action not included in the interface alphabet. This can be easily achieved by renaming such actions. Next, consider the actions $\overset{d}{\rightsquigarrow}$: since they denote the willingness of a process to let time elapse, they can be replaced by Dirac transitions; in this way, such actions can be synchronised in the concrete model. Formally, whenever $(T,\nu)\overset{d}{\rightsquigarrow}$, we replace it by the set of transitions $(T,\nu)\xrightarrow{a,d}dirac(T,\nu+d)$, for all actions $a$ in the private alphabet of the other component. After this transformation, the two components $S(T_1)$ and $S(T_2)$ have the same alphabet, and each time an action not in $A$ is performed, one of the components performs a Dirac transition. It follows that the two systems $S(T_1)$ and $S(T_2)$ must synchronise on all the actions except for $(\tau,0)$.

We need to verify the following:

$$S(T_1\underset{A}{\mid\mid}T_2)\equiv S(T_1)\underset{(\Sigma\times\mathbb{R}^{\ge 0})\setminus\{(\tau,0)\}}{\mid\mid}S(T_2)$$

Note that the two systems have different state spaces, and they cannot be equal: the states of $S(T_1\underset{A}{\mid\mid}T_2)$ are pairs of the type $(T_1'\underset{A}{\mid\mid}T_2',\nu)$, where $T$ is a parallel process and $\nu$ is a valuation on the union of the sets of clocks; the states of $S(T_1)\underset{A\times\mathbb{R}^{\le 0}}{\mid\mid}S(T_2)$ are pairs of

pairs of the type $((T_1', \nu(1)), (T_2', \nu(2)))$. For this reason, we have to match transitions of one system with equivalent transition of the other system. Since the target of transitions are measures on different spaces, we define two measures $\mu$ and $\mu'$ of the two systems, respectively, as equivalent ($\mu \equiv \mu'$) if $\mu(T_1 \underset{A}{||} T_2, X_1 \times X_2) = \mu'((T_1, X_1), (T_2, X_2))$, where $X_1$ and $X_2$ are measurable sets of valuations on the clocks of $T_1$ and $T_2$, respectively. We say that the two systems of Equation 10.3.1 are equivalent if each transition $(T_1' \underset{A}{||} T_2', \nu) \xrightarrow{a,d} \mu$ of $S(T_1 \underset{A}{||} T_2)$ exists if and only if there exists a transition $((T_1, \nu(1)), (T_2, \nu(2))) \xrightarrow{a,d} \mu'$ of $S(T_1) \underset{(\Sigma \times \mathbb{R}^{\geq 0}) \setminus \{(\tau,0)\}}{||} S(T_2)$, such that $\mu \equiv \mu'$.

Assume there exists a transition $(T_1' \underset{A}{||} T_2', \nu) \xrightarrow{a,d} \mu$; then:

- If $a \in A$, according to the rules of Table 10.4, such a transition exists if there exist two transitions $(T_1', \nu(1)) \xrightarrow{a,d} \mu_1$ and $(T_2', \nu(2)) \xrightarrow{a,d} \mu_2$ such that $\mu = \mu_1 \underset{A}{||} \mu_2$. These are possible transitions of $S(T_1)$ and $S(T_2)$, respectively, and therefore can be synchronised into the action $((T_1, \nu(1)), (T_2, \nu(2))) \xrightarrow{a,d} \mu_1 \otimes \mu_2$. $\mu$ and $\mu_1 \otimes \mu_2$ are equivalent from the definition of $\underset{A}{||}$ for measures.

- If $a \notin A$, $(a, d) \neq (\tau, 0)$ and it is a private action of $T_1$, such transition exists if there exist transitions $(T_1, \nu) \xrightarrow{a,d} \mu_1$ and $(T_2, \nu) \xrightarrow{a,d} \mathit{dirac}(T_2, \nu)$ such that $\mu = \mu_1 \underset{A}{||} \mathit{dirac}(T_2, \nu)$. Since these are transitions of $S(T_1)$ and $S(T_2)$, respectively, we can apply the same arguments as above.

- If $(a, d) = (\tau, 0)$ then one component can execute such action independently. This case is proved as the previous one with one component not performing any action and $\mu$ being the composition between a distribution $\mu'$ and a Dirac transition.

The other direction is proved in the same way.

□

**Remark 10.2 (The Expansion Law).** In the development of stochastic process algebras, the expansion law has had a relevant position. This law allows to rewrite the parallel operator in terms of simple operators by distributing the parallel operator inside the choice operator. For example, the simple term $(a \to P_1) || (b \to P_2)$ can be rewritten as $a \to (P_1 || b \to P_2) \,\square\, b \to (a \to P_1 || P_2)$. Such a law, while desirable, is difficult to achieve in a stochastic setting because, if each the execution of an action depends on some stochastically distributed delay, the execution of the second action being performed depends on the delay of the first action. This is one of the main advantages of restricting to exponential distributions, that enjoy the memoryless property, and therefore the delay related to the second action is independent of the delay of the first action.

It is possible to have such a law in SCSP with respect to external choice, since we use a clocked approach similar to that of [D'A99] and parallel composition can be rewritten in terms of external choice, if both components of the parallel operator are external choice

processes. Let $T_1 = \Box_i (a_i, \varphi_i) \to P_i$ and $T_2 = \Box_i (b_i, \gamma_i) \to Q_i$, then:

$$\begin{aligned} T_1 \underset{A}{||} T_2 \quad = \quad & \Box_{a_i \notin A} (a_i, \varphi_i) \to (P_i \underset{A}{||} \Delta(T_2)) \\ & \Box_{b_i \notin A} (b_i, \gamma_i) \to (\Delta(T_1) \underset{A}{||} Q_i) \\ & \Box_{a_i = b_i \in A} (a_i, \varphi_i \wedge \gamma_i) \to (P_i \underset{A}{||} Q_i) \end{aligned}$$

where $\underset{A}{||}$ is appropriately lifted to probabilistic terms. Since it is not our purpose to study the equational laws of SCSP, we will not go into details of how the equality above is proved; however, it is clear that actions not in $A$ can be executed independently, while those in the interface can be executed in common and only if both the guards of each components are satisfied. If both components perform an action, then the target is a distribution obtained by combining the distributions achieved by the components; if, otherwise, an action is performed independently by one component, the other component does not change, hence the Dirac distribution.

## 10.4 Equivalences for SCSP

In this section we define different equivalence relations, based on bisimulation and trace semantics, by extending the relation of the previous chapter. We also briefly analyse how operators behave with respect to such equivalence, and highlight the fact that the problems that exist in discrete models carry over to this setting.

### 10.4.1 Trace Semantics

We define an equivalence relation based on the trace semantics for stochastic transition systems of Section 9.2. The traces of a system are given by sequences of pairs $(a, d)$, where $a$ is an action in $\Sigma^\tau$ and $d$ is a non-negative delay (the only pair that is not possible is $(\tau, 0)$ as it is considered internal in the operational model). This model corresponds to that of timed traces for timed automata or Timed CSP; we chose not to adopt this model in Part I as it would lead to undecidable verification. In the case of SCSP we are not interested in decidability as our focus is now on how to define a trace semantics in the continuous setting, studying the resulting measurability issues.

**Definition 10.3.** Let $T$ be a SCSP term and $S(T)$ the stochastic transition system generated by the operational semantics. The trace semantics of $T$ is defined as the trace semantics of $S(T)$, that is,

$$\mathit{trace}(T) = \mathit{trace}(S(T)) \tag{10.4}$$

**Definition 10.4 (Trace Equivalence).** Let $T_1$ and $T_2$ be two SCSP terms. We say that $T_1$ and $T_2$ are *trace equivalent*, $T_1 \equiv_T T_2$, if $\mathit{trace}(T_1) = \mathit{trace}(T_2)$.

It can be easily observed that the trace semantics defined in this way is not compositional with respect to the operators of the language: the parallel composition operator does not preserve trace semantics and the example of Section 9.2 can be easily extended to SCSP terms.

Let us consider the issue of abstracting from internal actions. According to the semantics given above, the only internal action is the pair $(\tau, 0)$. This means that all $\tau$

actions with a non-zero delay associated to it are considered visible. This does not meet our intuition as we would like to consider two processes as equivalent if they can perform the traces $(\tau,1)(\tau,2)(a,1)$ and $(a,4)$, respectively: in both cases the visible action $a$ is performed after letting 4 time units elapse. This example also shows that it is not possible to ignore all $\tau$ actions, whatever the associated delay is, as we would then lose the information on the total amount of time elapsed. We therefore introduce a function that abstracts from internal computation while still preserving the timing information.

### Hiding $\tau$ Actions

We generalise the trace semantics of Definition 10.4 in order to ignore $\tau$ actions by collapsing several silent steps followed by a visible step into a single visible step whose delay is given by the sum of all the delays.

We say that a trace $(a_1,d_1)(a_2,b_2)\dots$ is $\tau$-free if $a_i \neq \tau$ for all $i \in \mathbb{N}$. The set of $\tau$-free traces is denoted by $L_f$ and we define a $\sigma$-algebra $\mathcal{F}_{L_f}$ in the usual way, as the $\sigma$-algebra generated by basic set. In this case, the basic sets are described by the sequences $(a_1,I_1)(a_2,I_2)\dots$, for all $a_i \in \Sigma^\tau$ and $I$ closed interval of $\mathbb{R}$, since we have countably many actions and the closed intervals are the generators for the Borel $\sigma$-algebra on the reals. Define a function $\tau{-}hide$, that, given a trace $(a_1,d_1)(a_2,d_2)\dots$, returns the $\tau$-free trace $(b_1,d_1)(b_1,d_2),\dots$ such that there exists a mapping $f:\mathbb{N}\to\mathbb{N}$ with $b_i = a_{f(i)}$, for all $j$ such that $f(i-1)<j<f(i)$, $a_j=\tau$, and $d_i = \sum_{j=f(i-1)+1}^{f(i)} d_j$.

**Proposition 10.5.** *The function $\tau{-}hide$ is measurable from $(L,\mathcal{F}_L)$ to $(L_f,F_{L_f})$.*

*Proof.* We need to prove that for all measurable sets of $\tau$-free traces, their counter-image under the function $\tau{-}hide$ is a measurable set of traces. The generators of $\mathcal{F}_{L_f}$ are the sets of traces $(a_1,I_1)(a_2,I_2)\dots$, $a_i \in \Sigma^\tau$ and $I_i$ being a closed interval of the real line. We show that, for each element $(a_i,I_i)$ of the generator, its counter-image is a measurable set. It follows that the counter-image of the whole set if given by concatenation of the measurable sets corresponding to each element.

Let us focus then on a generic pair $(a,I)$, where $I=[l,u]$. Its counter-image is given by the sequences of $\tau$-actions of any finite length, such that the sum of their delays is an element of $I$. The elements of length one are given exactly by $(a,I)$.

Let us consider the set of pairs of real numbers such that their sum is an element of $I$. Let $\delta=(u-l)/2$ and $\epsilon$ be arbitrarily small. Construct the following pairs of intervals:

$$\begin{aligned} U^2_{l,u,\epsilon} =& (([0,\epsilon],[l-\epsilon,u]),\\ & ([\epsilon,2\epsilon],[l-2\epsilon,u-\epsilon]),\\ & ([2\epsilon,3\epsilon],[l-3\epsilon,u-2\epsilon]),\\ & \dots\\ & ([u-\epsilon,u],[0,\epsilon])) \end{aligned}$$

(the intervals on the right are implicitly intersected with the set of non-negative numbers). This defined the set $U^2_{l,u,\epsilon}$, containing the pairs of numbers whose sum is included in $[a-\epsilon,b+\epsilon]$. The number of pairs described above is clearly finite for $\epsilon>0$. Consider $\epsilon_0 \leq 1/2$ and $\epsilon_{i+1}=\epsilon_i^2$. Then, the set of pairs of numbers whose sum is in $[l,u]$ is given by $U^2_{l,u} = \bigcap_i U^2_{l,u,\epsilon_i}$, which is measurable since it is the countable intersection of measurable sets. To prove this equality, consider a pair $(m_1,m_2)$ in $U^2_{l,u}$: since $\epsilon_i \to 0$ when $i\to\infty$, then the sum of its components must be in $[l,u]$. This shows that the set of pairs of numbers whose sum is in a given interval is measurable.

The construction to show that the same holds for the set of tuples of size $i+1$ of numbers whose sum is in $[l,u]$ is done by induction and is similar:

$$\begin{aligned} U^{i+1}_{l,u,\epsilon} =& (([0,\epsilon], U^{i}_{l-\epsilon,u}),\\ & ([\epsilon, 2\epsilon], U^{i}_{l-2\epsilon,u-\epsilon}),\\ & \dots\\ & ([u-\epsilon, u], U^{2}_{0,\epsilon})) \end{aligned}$$

This defines the tuples of reals whose sum is $[l-\epsilon, u+\epsilon]$. By the construction above, we can define $U^{i+1}_{l,u}$ as the countable intersection of measurable sets.

Therefore, the counter-image of $(a, I)$ under the function $\tau{-}hide$ is the countable union of measurable sets of finite sequences of pairs such that the sum of the delays is in $I$ and all actions but the last one is $a$. □

This result allows us to define a more general trace equivalence that abstracts from internal computations, while still taking into account the delays associated with each action. Since $\tau{-}hide$ is measurable, it induces a measure on the set of $\tau$-free traces for each measure on the set of traces.

**Definition 10.6 ($\tau$-abstract Trace Semantics).** The $\tau$-abstract trace semantics is defined as

$$trace_f(T) = \{tr_f \in sub\mathcal{D}(L_f, \mathcal{F}_{L_f}) \mid \exists fr.\ tr_f = \tau{-}hide(fr) \ \wedge\ fr \in trace(T)\} \tag{10.5}$$

**Definition 10.7 ($\tau$-abstract Trace Equivalence).** Let $T_1$ and $T_2$ be two SCSP terms. We say that $T_1$ and $T_2$ are *$\tau$-abstract trace equivalent*, $T_1 \equiv_{T_f} T_2$, if $trace_f(T_1) = trace_f(T_2)$.

It is easy to observe the following.

**Theorem 10.8.** *Given two SCSP processes $T_1$ and $T_2$, if $T_1 \equiv_T T_2$, then $T_1 \equiv_{T_f} T_2$.*

This definition of trace semantics is still not a congruence. However, it reflects our intuition as it views processes whose visible actions are executed at the same time as trace equivalent.

### Refinement Relations

The trace semantics defined in this section can be easily extended to refinement: like for standard CSP and CCSP, refinement is defined as inverse inclusion of behaviours.

**Definition 10.9 (Trace refinement).** Let $T_1$ and $T_2$ be two SCSP terms. We say that $T_2$ refines $T_1$ with respect to trace semantics if the set of traces of $T_2$ is contained in the set of trace of $T_1$:

$$T_1 \sqsubseteq_T T_2 \text{ iff } trace(T_2) \subseteq trace(T_1). \tag{10.6}$$

$\tau$-abstract trace refinement is defined similarly by considering the $\tau$-abstract trace semantics. Refinement relations for SCSP suffer from all the usual limitations of trace equivalence, in particular they are not compositional and therefore not suitable for compositional reasoning.

### 10.4.2 Bisimulation Equivalences

Again, we define bisimulation equivalences by relying on the underlying operational model of stochastic transition systems.

**Definition 10.10 (Strong Bisimulation).** Let $T_1$ and $T_2$ be two SCSP; we say that $T_1$ and $T_2$ are *strongly bisimilar*, $T_1 \sim T_2$, if there exists a strong bisimulation between $S(T_1)$ and $S(T_2)$.

We show that strong bisimulation is compositional with respect to the operators of the process algebra.

**Theorem 10.11.** *Strong bisimulation is a congruence with respect to the operators of Table 10.1.*

*Proof.* We need to prove the result both for the probabilistic terms $P$ and for the standard operators $T$. The result for the standard operators is straightforward; assume $P_1 \sim P_2$ (where $\sim$ is lifted to measures in the usual way), or $T_1 \sim T_2$, then:

- $P \sqcap P_1 \sim P \sqcap P_2$ is straightforward from the definition of bisimulation;
- $(a, \phi) \to P \,\square\, (b, \phi) \to P_1$ and $(a, \phi) \to P \,\square\, (b, \phi) \to P_2$ are bisimilar as they both enable transitions labelled $(b, d)$ (under the condition that the guards are satisfied) that have $\sim$-equivalent distributions as target, by bisimulation hypothesis; such transitions result in the same (up to bisimulation) set of combined transitions.
- recursion, hiding and timed invariant are proved by a simple application of the definition of bisimulation to the corresponding operational rules;
- the congruence with respect to the parallel operator is a consequence of Theorems 9.13 and 10.2.

Let us now consider the probabilistic operators:

- $\Delta(T_1) \sim \Delta(T_2)$ holds trivially;
- $P \oplus_q P_1$ and $P \oplus_q P_2$ are bisimilar since, for all $\sim$-closed measurable sets $Y$ and valuations $\nu$:

$$\begin{aligned}(\mu \oplus_q \mu_1)(Y) &= q\mu(Y) + (1-q)\mu_1(Y) \\ &= q\mu(Y) + (1-q)\mu_2(Y) \\ &= (\mu \oplus_q \mu_2)(Y)\end{aligned}$$

  where $(P_1, \nu) \longmapsto \mu_1$, $(P_2, \nu) \longmapsto \mu_2$ and $(P, \nu) \longmapsto \mu$;
- $\{\!|X|\!\}P_1$ and $\{\!|X|\!\}P_2$ are bisimilar since, for all $\sim$-closed measurable sets $Y$ and validations $\nu$, $\{\!|X|\!\}\mu_1$ and $\{\!|X|\!\}\mu_2$ agree on the values of clocks in $X$ because they both reset them, and agree on the other components by bisimulation hypothesis.

This completes the proof and shows that strong bisimulation is a congruence with respect to the operators of the language. □

**Definition 10.12 (Weak Bisimulation).** Let $T_1$ and $T_2$ be two SCSP terms; we say that $T_1$ and $T_2$ are *weakly bisimilar*, $T_1 \approx T_2$, if there exists a weak bisimulation between $S(T_1)$ and $S(T_2)$.

One would expect that the negative result for standard process algebras about weak bisimulation not being a congruence would extend to our case. This is not the case: consider the classical example of processes $a \rightarrow STOP$ and $a \rightarrow \tau \rightarrow STOP$. Although these processes are weakly bisimilar, their compositions with the process $b \rightarrow STOP$ by means of the choice operator $\Box$ are not bisimilar. This is not a valid example for SCSP because the process $P = a \rightarrow \tau \rightarrow STOP \,\Box\, b \rightarrow STOP$ cannot be generated by our process algebra. Since we have imposed the restriction that external choice can only be guarded by visible actions, it is not possible to have a $\tau$ action immediately after the choice operator. A graph isomorphic to that corresponding to $P$ can be obtained by applying a hiding operator at some outer level, but in this case it would be possible to distinguish the processes at the time the choice operator is introduced. We get the following result.

**Theorem 10.13.** *Weak bisimulation is a congruence with respect to the operators of Table 10.1.*

*Proof.* The proof follows the same steps as the proof of Theorem 10.11; in particular, the proof for external choice works under the assumption that all the actions $a_i$ of the choice operators are visible and therefore can be matched directly. □

**Remark 10.3.** It would be interesting to extend the approach of Section 10.4.1 to weak bisimulation in order to hide internal computation while still preserving the information about time in weak transitions. Such an approach, however, does not work with the definition of weak transitions and weak bisimulation that we have given: in a weak transition, we match a single $a$ action with a sequence $\tau^* \, a \, \tau^*$ of actions. In our setting, each action has a delay associated to it. By applying the function defined above, we still would have to consider the delays associated to the $\tau$ actions after the occurrence of $a$. Such delays cannot be ignored (or else we would match a transition $a$ taking some amount of time with a weak transition that takes longer) nor added to the delay corresponding to $a$. It follows that the idea of Section 10.4.1 is not compatible with weak transitions as we have defined them and it requires a different formulation of time-abstract weak transitions.

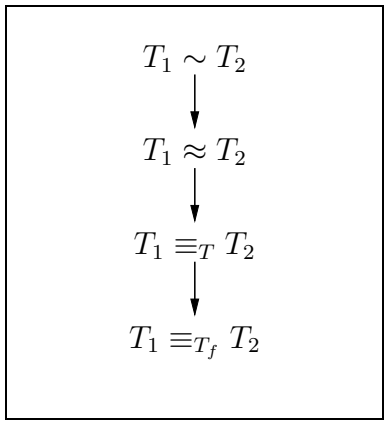



Figure 10.2: Relationship between the different equivalence relations for SCSP.

## 10.5 Discussion and Summary

In this chapter, we have introduced a new stochastic process algebra, SCSP. We have defined equivalences based on trace semantics and bisimulation, showing that the former

is not a congruence, while both strong and weak bisimulations are. These equivalences are based on those defined for stochastic transition systems and therefore the results of this chapter depend on those of the previous chapters. In particular, trace semantics is defined only under measurable schedulers, and we have used the notions of global bisimulations so that bisimilarity is a sufficient condition for trace equivalence. Figure 10.2 shows the relation between the different equivalence notions for SCSP processes.

The main purpose of this chapter was to give a motivating example for stochastic transition systems, highlighting in particular how the restrictions we imposed on the model are reasonable. However, the development of SCSP is far from being as complete as those of other process algebras for general distributions. Spades [D'A99] is probably the most similar approach to SCSP as a modelling language: both algebras model stochastic extensions to timed automata and use similar ideas, like clocks. In Spades, however, the values of clocks decrease with the passage of time and actions are triggered when their values reach zero; in SCSP, the values of clocks increase, obtaining a model which is closer to timed automata. The two algebras are then given different equivalence relations: Spades terms are related by several different notions of bisimulation, while SCSP is given a CSP-like trace semantics, with the corresponding equivalence and refinement relations. More different is the semantic model of [Bra02] to that of SCSP: in this case the stochastic process algebra is modelled on Interactive GSMPs and the corresponding operational model is quite different from that of stochastic transition systems, with a different notion of weak transitions. These two alternative process algebras present a more complete treatment of the resulting equivalence relations, studying aspects like axiomatisation.

We only went as far as defining a trace semantics for SCSP, leaving extensions such as failure semantics out; this would require a deeper analysis of both the algebra and the operational model of stochastic transition systems; existing proposals for discrete probabilistic extensions to CSP [Sei92, Nor97, MMSS95] are possible starting points for the definition of failure semantics.

# Chapter 11

# Conclusions

The main issue analysed in the second part of this thesis was the interaction between non-deterministic and probabilistic behaviour in systems with continuous state spaces, arbitrary probabilistic executions and uncountable non-determinism.

Driven by the aim of defining a stochastic process algebra equipped with a trace-based semantics, we realised the importance of the measure-theoretic issues that arise when studying the behaviour of a system across several steps of computation. For this reason, we focused on the analysis of these problems at the operational level, by defining a new model for non-deterministic continuous state space probabilistic systems, called stochastic transition systems.

In Chapter 8 we showed that it is necessary to impose restrictions on how non-determinism is resolved by schedulers: arbitrary decisions can result in objects that are intractable from a mathematical point of view. The main result of Chapter 8 was the identification of the class of measurable schedulers, that is, those schedulers that generate all and only tractable executions. This allows for the definition of measures on executions, thus enabling us to study the behaviour of a run of a system. The notion of measurable schedulers was shown to be compositional with respect to parallel composition, yielding a fundamental property that enables compositional reasoning.

Chapter 9 studied equivalences for stochastic transition systems. Firstly, we introduced the notion of trace semantics, an extension to the continuous case of trace distributions for models with discrete probabilities. We also defined strong and weak bisimulations. It was interesting to discover that bisimulation relations defined point-wise (local bisimulations) do not preserve linear-time properties like trace semantics. This is again due to measure-theoretic issues and reinforces the case for the importance of measurability when dealing with non-determinism. As a solution to this problem, we defined a stronger version of bisimulation that preserves trace semantics by relating probability measures on states rather than individual states themselves.

The results above finally set the scene for defining the stochastic process algebra SCSP in Chapter 10. We defined several equivalences for SCSP by mapping process terms to stochastic transition systems and by inheriting the equivalences of Chapter 9. This process algebra serves as the motivation for the work above; as such, the contribution is limited, since we have only shown that the negative result for compositionality shown for the discrete case [Seg95a, Low93b] extends, unsurprisingly, to a timed stochastic setting. We have also discussed how to abstract from internal computation while still considering the information about timing.

The thesis has achieved a better understanding of the problems concerned with introducing non-determinism in continuous systems, forming a basis for more research on the

topic.

## 11.1 Future Work

The results of this part lay the foundations for an approach to the study of systems presenting non-deterministic and probabilistic behaviours in conjunction with continuous state spaces that takes into account the issues of measure theory in order to study the behaviour of the executions of such systems.

This prompts for several possible directions of future work, both from the operational point of view, through stochastic transitions systems, and from the process algebraic point of view.

### 11.1.1 Stochastic Transition Systems

The core results for a model for non-deterministic and probabilistic systems call for further research in several directions. Firstly, we think it would be interesting to further investigate the problems of measurability that the definition of local bisimulation highlighted, studying whether it is possible to define a condition on the transition relation under which global and local bisimulation coincide. Such condition would restrict to a class of "good" systems that would rule out pathological examples like those we used.

The logical characterisation of (local) bisimulation we gave in Chapter 9 was aimed at pointing out the difficulties that uncountable non-determinism introduces with respect to the purely probabilistic model of LMPs. It would be desirable to have a better logical characterisation, maybe given in terms of a variant of PCTL.

From the point of view of trace semantics, it would be interesting to define simulation relations that characterise trace pre-congruence, thus extending the results of [LSV03] to the continuous domain.

Finally, several other directions are possible, aimed at extending the results for LMPs or other models embedded in stochastic transition systems to our setting.

### 11.1.2 Stochastic Process Algebra

The process algebra of Chapter 10 needs more research to make it into a comprehensive approach. A complete axiomatisation with respect to a trace semantics is hampered by the compositionality issues discussed above and many desirable laws are likely to fail. Working with trace pre-congruence and simulation relations can possibly offer the right framework for a trace and refinement based approach to process algebra.

# Bibliography


[ABB+01] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Möller, P. Pettersson, C. Weise, and W. Yi. UPPAAL - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan, editors, *Modelling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science Tutorial, pages 100–125. Springer–Verlag, 2001.

[ABBL03] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1-3):411–475, 2003.

[ABL98] L. Aceto, A. Burgueño, and K. G. Larsen. Model checking via reachability testing for timed automata. In *Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 263–280. Springer-Verlag, 1998.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):1–34, 1993.

[AD91] R. Alur and D. Dill. The theory of timed automata. In de Bakker et al. [dBHRR91], pages 45–73.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AD96] R. Alur and D. L. Dill. Automata-theoretic verification of real-time systems. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, Trends in Software, pages 58–82. John Wiley & Sons Ltd, 1996.

[AFH99] R. Alur, L. Fix, and T.A. Henzinger. Event clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[AFM+02] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for modelling and implementation of embedded systems. In Katoen and Stevens [KS02], pages 460–464.

[AL03] R. M. Amadio and D. Lugiez, editors. *Proceedings of CONCUR 2003, Marseille, France*, volume 2761 of *Lecture Notes in Computer Science*, 2003.

[Alu97] R. Alur. Timed automata. In Grumberg [Gru97], pages 8–22.

[AMP98] E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In CONCUR 98 [CON98], pages 470–484.

[Ash72] Robert B. Ash. *Real Analysis and Probability.* Academic Press, 1972.

[ASSB96] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[AW04] S. Andova and T.A.C. Willemse. Equivalences for silent transitions in probabilistic systems. In *Proceedings of EXPRESS 2004*, 2004.

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[BD00] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[BD04] M. Bravetti and P.R. D'Argenio. Tutte le algebre insieme: Concepts, discussions and relations of stochastic process algebras with general distributions. In C. Baier, B. Haverkort, H. Hermanns, J-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*. Springer, 2004.

[BDFP00] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In Emerson and Sistla [ES00], pages 464–479.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BDG94] M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proceedings of the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, November 1998.

[BG02] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[BH97] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130, 1997.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[Bil95] Patrick Billingsley. *Probability and Measure*. Wiley, New York, 3rd edition, 1995.

[BIM95] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.

[BK90] J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In Baeten and Mauw [BM99], pages 146–161.

[BKHW03] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf. Comparative branching-time semantics for Markov chains. In Amadio and Lugiez [AL03], pages 482–497.

[BM99] J. Baeten and S. Mauw, editors. *Proceedings of CONCUR 99,* Eindhoven, The Netherlands, volume 1664 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Bošn99] D. Bošnački. Digitization of timed automata. In *Proc. of the Fourth International Workshop on Formal Methods for Industrial Critical Systems FMICS '99, Trento, Italy*, pages 283–302, 1999.

[BR84] S.D. Brookes and A.W. Roscoe. An improved failures model for communicating processes. In Brookes et al. [BRW84], pages 281–305.

[Bra02] M. Bravetti. *Specification and Analysis of Stochastic Real-Time Systems*. PhD thesis, Università di Bologna, Padova, Venezia, 2002.

[Bro93] S. D. Brookes. Full abstraction for a shared variable parallel language. *Information and Computation*, 127:145–163, 1993.

[BRW84] S.D. Brookes, A.W. Roscoe, and G. Winskel, editors. *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[BS01] E. Bandini and R. Segala. Axiomatizations for probabilistic bisimulation. In Orejas et al. [OSvL01], pages 370–381.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[Čer93] K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In *Proceedings of the Fourth International Workshop on Computer Aided Verification*, pages 302–315. Springer-Verlag, 1993.

[CK03] S. Cattani and M. Kwiatkowska. CSP + Clocks: a process algebra for timed automata. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proc. 3rd Workshop on Automated Verification of Critical Systems (AVoCS'03)*, Technical Report DSSE-TR-2003-2, University of Southampton, pages 50–63, April 2003.

[CK05] S. Cattani and M. Kwiatkowska. A refinement-based process algebra for timed automata. *Formal Aspects of Computing*, 2005. To appear.

[Cle92] W.R. Cleaveland, editor. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[CON98] *Proceedings of CONCUR 98,* Nice, France, volume 1466 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[CS02] S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In L. Brim, P. Janar, M. Ketinsky, and A. Kuera, editors, *Proc. Proc. 14th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[CSKN05] S. Cattani, R. Segala, M. Kwiatkowska, and G. Norman. Stochastic transition systems for continuous state spaces and non-determinism. In V. Sassone, editor, *Proc. Foundations of Software Science and Computation Structures (FOSSACS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 125–139. Springer Verlag, 2005.

[CVWY92] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In Larsen and Skou [LS92b], pages 399–409.

[D'A97] P.R. D'Argenio. Regular processes and timed automata. In M. Bertran and T. Rus, editors, *Proceedings of the 4th AMAST Workshop on Real-Time Systems, Concurrent, and Distributed Software, ARTS'97,* Mallorca, Spain, volume 1231 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 1997.

[D'A99] P. R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, Department of Computer Science, University of Twente, November 1999.

[DB96] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, 1996.

[dBHRR91] J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[dBRR89] J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors. *Proceedings of the REX Workshop "Stepwise Refinement of Distributed Systems"*, volume 430 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[DDP04] V. Danos, J. Desharnais, and P. Panangaden. Bisimulation and cocongruence for probabilistic systems. *Information and Computation*, 2004.

[DEP98] J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for Labeled Markov Processes. In *Logic in Computer Science*, pages 478–487, 1998.

[DGJP99] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled markov systems. In Baeten and Mauw [BM99], pages 258–273.

[DGJP00] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating continuous Markov processes. In *Proceedings* $15^{th}$ *Annual Symposium on Logic in Computer Science,* Santa Barbara, California, pages 95–106. IEEE Computer Society Press, 2000.

[DGP02] J. Desharnais, V. Gupta, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179:163–193, 2002.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Dil90] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 197–212. Springer-Verlag New York, Inc., 1990.

[DJR+92] J. Davies, D. Jackson, G. Reed, J. Reed, A. Roscoe, and S. Schneider. Timed CSP: Theory and practice. In *Proceedings of REX Workshop, Nijmegen, LNCS 600, Springer-Verlag*, 1992.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer-Verlag, 1996.

[Dud89] R. M. Dudley. *Real Analysis and Probability.* Wadsworth and Brookes/Cole, 1989.

[ES00] E. Allen Emerson and A. Prasad Sistla, editors. *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000.

[For93] Formal Systems (Europe) Ltd. Failures Divergence Refinement — User manual and tutorial, 1993. Version 1.3.

[FZ88] M. Fischer and L. Zuck. Reasoning about uncertainty in fault tolerant distributed systems. In M. Joseph, editor, *Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 142–158. Springer-Verlag, 1988.

[GHR92] N. Götz, U. Herzog, and M. Rettelbach. TIPP – a language for timed processes and performance evaluation. Technical Report 4/92, IMMD7, University of Erlangen-Nürnberg, Germany, 1992.

[Gir81] M. Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, number 915 in Lecture Notes in Mathematics, pages 68–85. Springer-Verlag, 1981.

[GJS90] A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.

[Gru97] O. Grumberg, editor. *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[GSS95] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[GSST90] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In LICS 90 [LIC90], pages 130–141.

[Han91] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Science, Uppsala University, 1991.

[Han94] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems*. Elsevier, 1994.

[Her02] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[Hil94] J. Hillston. The nature of synchronization. In U. Herzog and M. Rettelbach, editors, *Proceedings of the 2nd Workshop on Process Algebra and Performance Modeling*, pages 51–70, 1994.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HJ89] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In RTSS 89 [RTS89].

[HJ90] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In RTSS 90 [RTS90].

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Automata, Languages and Programming*, pages 545–558, 1992.

[HNSY92] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In LICS 92 [LIC92], pages 394–406.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, 1985.

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[HRSV02] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52:183–220, 2002.

[Jac92] D. M. Jackson. *Logical Verification of Reactive Software Systems.* PhD thesis, University of Oxford, 1992.

[Jon89] B. Jonsson. On decomposing and refining specifications of distributed systems. In de Bakker et al. [dBRR89], pages 361–387.

[JP89] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In LICS 89 [LIC89], pages 186–195.

[KD01] J.P. Katoen and P.R. D'Argenio. General distributions in process algebra. In E. Brinksma, H. Hermanns, and J.P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis. First EEF/Euro Summer School on Trends in Computer Science.* Berg en Dal, The Netherlands, July 3-7, 2000., volume 2090 of *Lecture Notes in Computer Science*, pages 375–430. Springer-Verlag, 2001.

[Kel76] R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.

[KNSS00] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic real-time graphs. In C. Palamidessi, editor, *Proceedings of CONCUR 2000,* University Park, PA, USA, volume 1877 of *Lecture Notes in Computer Science*, pages 132–137. Springer, 2000.

[KNSS02] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, June 2002.

[KS02] J.-P. Katoen and P. Stevens, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Grenoble, France*, volume 2280 of *Lecture Notes in Computer Science.* Springer-Verlag, 2002.

[LIC89] *Proceedings* $4^{th}$ *Annual Symposium on Logic in Computer Science,* Asilomar, California. IEEE Computer Society Press, 1989.

[LIC90] *Proceedings* $5^{th}$ *Annual Symposium on Logic in Computer Science,* Philadelphia, USA. IEEE Computer Society Press, 1990.

[LIC92] *Proceedings* $7^{th}$ *Annual Symposium on Logic in Computer Science,* Santa Cruz, California. IEEE Computer Society Press, 1992.

[LIC03] *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada.* IEEE Computer Society, 2003.

[LIC04] *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland*. IEEE Computer Society, 2004.

[Low93a] G Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, Oxford University Computing Laboratory, 1993.

[Low93b] G. Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report PRG-TR-11-93, Oxford University Computing Laboratory, 1993.

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 56(3):131–133, 10 November 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LS89a] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In POPL 89 [POP89], pages 344–352.

[LS89b] N.A. Lynch and E.W. Stark. A proof of the Kahn principle for Input/Output automata. *Information and Computation*, 82(1):81–92, 1989.

[LS91a] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[LS91b] S. Lee and S. Sluizer. An executable language for modeling simple behavior. *IEEE Transactions on Software Engineering*, 17(6):527–543, June 1991.

[LS92a] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [Cle92], pages 456–471.

[LS92b] K.G. Larsen and A. Skou, editors. *Proceedings of the Third Workshop on Computer Aided Verification,* Aalborg, Denmark, July 1991, volume 575 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[LS95a] I. Lee and S.A. Smolka, editors. *Proceedings of CONCUR 95, Philadelphia, PA, USA*, volume 962 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[LS95b] N.A. Lynch and R. Segala. A comparison of simulation techniques and algebraic techniques for verifying concurrent systems. *Journal of Formal Aspects of Computing Science*, 7(3):231–265, 1995.

[LSV03] N. A. Lynch, R. Segala, and F. W. Vaandrager. Compositionality for probabilistic automata. In Amadio and Lugiez [AL03], pages 204–222.

[LV91] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations for timing-based systems. In de Bakker et al. [dBHRR91], pages 397–446.

[LV96] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[Mis00] M. W. Mislove. Nondeterminism and probabilistic choice: Obeying the laws. In Palamidessi [Pal00], pages 350–364.

[MMSS95] C. Morgan, A. Mclver, K. Seidel, and J.W. Sanders. Argument duplication in probabilistic CSP. Technical Report PRG-TR-95, Oxford University Computing Laboratory, 1995.

[MS96] U. Montanari and V. Sassone, editors. *Proceedings of CONCUR 95,* Pisa, Italy, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [BK90], pages 401–415.

[Nor97] G. Norman. *Metric Semantics for Reactive Probabilistic Processes*. PhD thesis, School of Computer Science, University of Birmingham, 1997.

[NS92] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In Larsen and Skou [LS92b], pages 376–398.

[NS94] X. Nicollin and J. Sifakis. The algebra of timed processes, atp: theory and application. *Information and Computation*, 114(1):131–178, 1994.

[OMW03] J. Ouaknine, M. Mislove, and J. Worrell. Axioms for probability and nondeterminism. In *Proceedings of EXPRESS 03, Marseille, France*, volume 91(3) of *Electronic Notes in Theoretical Computer Science*, 2003.

[OSvL01] F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors. *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

[Oua01] J. Ouaknine. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems*. PhD thesis, Oxford University, 2001.

[Oua02] Joël Ouaknine. Digitisation and full abstraction for dense-time model checking. In Katoen and Stevens [KS02], pages 37–51.

[OW02] J. Ouaknine and J. B. Worrell. Timed CSP = Closed Timed Automata. In *Proceedings of EXPRESS 02*, volume 68(2) of *Electronic Notes in Theoretical Computer Science*, 2002.

[OW03a] J. Ouaknine and J. B. Worrell. Revisiting digitization, robustness, and decidability for timed automata. In LICS 03 [LIC03], pages 198–207.

[OW03b] J. Ouaknine and J. B. Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of HSCC 03*, volume 2623 of *Lecture Notes in Computer Science*, pages 375–388. Springer-Verlag, 2003.

[OW04] J. Ouaknine and J. B. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In LICS 04 [LIC04], pages 54–63.

[Pal00] C. Palamidessi, editor. *Proceedings of CONCUR 2000,* University Park, PA, USA, volume 1877 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[Pan97] P. Panangaden. Stochastic techniques in concurrency, 1997. Notes from the EATCS Summer School, Udine.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer science Department, Aarhus University, 1981.

[PLS00] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In Palamidessi [Pal00], pages 334–349.

[POP89] *Conference Record of the* $16^{th}$ *ACM Symposium on Principles of Programming Languages,* Austin, Texas, 1989.

[Ree88] G. Reed. *A Uniform Mathematical Theory for Real-Time Distributed Computing*. PhD thesis, Oxford University, 1988.

[Ros94] A.W. Roscoe. Model-checking CSP. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 353–378. Prentice Hall, 1994.

[Ros98] A.W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.

[RR88] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.

[RTS89] *Proceedings of the* $10^{\text{th}}$ *IEEE Symposium on Real-Time Systems,* Santa Monica, Ca., 1989.

[RTS90] *Proceedings of the* $11^{\text{th}}$ *IEEE Symposium on Real-Time Systems,* Orlando, Fl., 1990.

[Sch95] S. Schneider. An operational semantics for timed CSP. *Information and Computation*, 116(2):193–213, 1995.

[Sch97] S. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28(1):43–90, 1997.

[Sch99] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley and Sons, 1999.

[Seg95a] R. Segala. A compositional trace-based semantics for probabilistic automata. In Lee and Smolka [LS95a], pages 234–248.

[Seg95b] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.

[Sei92] K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.

[SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[Sto02] M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, the Netherlands, April 2002.

[Str93] B. Strulo. *Process algebra for discrete event simulation*. PhD thesis, Department of Computing, Imperial College. London, 1993.

[SV03] M. Stoelinga and F. Vaandrager. A testing scenario for probabilistic automata. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP'03)* Eindhoven, the Netherlands, June 2003, volume 2719 of *Lecture Notes in Computer Science*, pages 464–477. Springer-Verlag, 2003. Also, Technical Rapport NIII-R0307, Nijmegen Institute of Computing and Information Sciences, University of Nijmegen, 2003.

[TAKB96] Serdar Tasiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In Montanari and Sassone [MS96], pages 546–562.

[Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.

[vBMOW04] F. van Breugel, M. Mislove, J. Ouaknine, and J. Worrell. Testing for simulation and bisimulation in labelled markov processes. *Theoretical Computer Science*, 2004.

[Whi80] W. Whitt. Continuity of generalized semi-Markov processes. *Mathematics of Operations Research*, 5:494–501, 1980.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [BK90], pages 502–520.

[Yov96] S. Yovine. Model checking timed automata. In G. Rozenberg and F. Vaandrager, editors, *European Educational Forum: School on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152, 1996.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2), 1997.

[YPD94] W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In D. Hogrefe and S. Leue, editors, *Proc. of the 7th International Conference on Formal Description Techniques*, pages 223–238, 1994.

# Index


$\sigma$-algebra, 15
  Borel, 16

Analytic space, 18

Bisimulation, 10, 113, 114
  for CCSP, 45
  for SCSP, 147
  for stochastic transition systems, 110, 113, 114
    strong, 113
    weak, 114
  for timed automata, 37
  strong, 11
  weak, 11

CCSP, 38
  clock resets, 42
  external clocks, 41
  internal clocks, 41
  invariants, 42
  normal form, 45
  operational semantics, 42
  region automaton, 44
  syntax, 38
  transitions, 43
Clock valuation, 32
Combined transition, 90
CSP, 14
  operational semantics, 15
  syntax, 14

Execution
  for labelled transition systems, 10
  for stochastic transition systems, 91

Failure semantics, 12

Hyper-transition, 112
  strong, 112

Labelled Transition Systems, 9
Local bisimulation
  strong, 110
  weak, 110

Measurable function, 16
Measure, 16
  extension, 17
  probability, 16
  product, 17

Parallel composition
  for stochastic transition systems, 100
  for labelled transition systems, 10
Probabilistic execution, 94

Region, 35
Region automaton, 36
Region failures, 56, 57
  equivalence, 57
  refinement, 57
  rules, 57
Region traces, 48–51
  equivalence, 50
  refinement, 50
  rules, 51
Regular conditional probability, 17

Scheduler, 93
  measurable, 94
SCSP, 136, 138, 140–142
  Operational semantics, 138, 140–142
  Syntax, 136
Semi-ring, 17

Stochastic transition system, 89

Timed automaton, 32
Trace, 10
    for stochastic transition systems, 92
Trace distribution, 108
Trace semantics, 11
    for SCSP, 144
    for stochastic transition systems, 108, 109
Transition, 9
    for stochastic transition systems, 89
    for timed automata, 33

Weak transition, 10
    for stochastic transition systems, 99