Georgiana Caltais

Coalgebraic Tools for Bisimilarity and Decorated Trace Semantics

Coalgebraic Tools for Bisimilarity and Decorated Trace Semantics

een wetenschappelijke proeve op het gebied
van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. mr. S.C.J.J. Kortmann,
volgens besluit van het college van decanen
in het openbaar te verdedigen op maandag 16 december 2013
om 12.30 uur precies door

Georgiana Caltais

geboren op 20 april 1984
te Suceava, Roemenië

Promotoren:

Prof. dr. Jan Rutten	
Prof. dr. Anna Ingólfsdóttir	(Háskólinn í Reykjavík, IJsland)

Copromotoren:

Dr. Alexandra Silva	
Dr. Marcello Bonsangue	(Universiteit Leiden, Nederland)

Manuscriptcommissie:

Prof. dr. Luca Aceto	(Háskólinn í Reykjavík, IJsland)
Prof. dr. Herman Geuvers	
Dr. Bas Luttik	(Technische Universiteit Eindhoven, Nederland)
Prof. Ugo Montanari	(Universitá di Pisa, Italië)
Dr. Erik de Vink	(Technische Universiteit Eindhoven, Nederland)

ISBN:978-94-6191-968-7

Contents

Acknowledgements

First of all I want to thank, in alphabetical order, Alexandra Silva, Anna Ingólfsdóttir, Jan Rutten, Luca Aceto and Marcello Bonsangue who supervised my research during the last three years, inspired me in my work and provided me with the support I needed in order to complete this thesis. I truly appreciate their flexibility, understanding and willingness to initiate the collaboration between the two research groups at Radboud University and Reykjavik University, making my PhD journey more interesting and prolific.
Many thanks to my collaborators that opened the doors for more applicability in my work: Filippo Bonchi and Damien Pous, co-authors of the results in Chapter 5, and Eugen-Ioan Goriac and Dorel Lucanu, co-authors of the contributions in Chapter 3. I am more than grateful to the members of the reading committee Bas Luttik, Erik de Vink, Herman Geuvers, Luca Aceto and Ugo Montanari for their comments and suggestions on this thesis. Moreover, I want to thank Wan Fokkink for planting the seeds of my interest in modelling and studying the semantics of concurrent reactive systems, during our rather fortuitous meeting in Reykjavik, in 2011.
I am very grateful to my friends in Reykjavik, for making my Icelandic experience brighter, warmer and more comfortable. Special credits go to the best Vikings ever Agnes, Ali, Andrea, Angelo, Claudio, Dario, Eugen, Gabriel, Ghiuseppe, Hamid, Hogni, Mădălina, Maria, Marijke, Matteo, Niccolo, Nils, Paulo, Pradipta, Robert, Rolanda, Stephan, Ute, Victor, Viky and all the Mjolnir crew, and to my dear Icelandic parents Valla and Skarpi. I want to express my appreciation to Andrei Manolescu and Marjan Sirjani for their support at the university. Whenever I traveled to the Netherlands I enjoyed the excellent company of my colleagues and friends Adam, Afrodita, Alexandra, Antonis, Darya, Fernando, Filippo, Francesco, Giannicola, Helle, Jonce, Jos, Joost, Neko, Michiel, Rita, Robbert, Sarah and Wojtek; they did a great job in cheering up my stays. I thank my Romanian friends Amalia, Anca, Andrei, Ciprian, Cozmin, Liliana, Mădălina, Marian, Măriuca, Mihaela, Mihai, Radu, Romina and Roxana for still remembering me, and making me feel welcome back home. Moreover, I want to thank my professors Gheorghe Marchitan, Cătălin Ţigăeru, and Adriana and Ovidiu Gheorghieş for awakening my passion for Mathematics and Computer Science, and for providing me with important advice when most needed.
I wish to express my gratitude to Goran Kristófer for his constant encouragements, care, and uplifting discussions, that invaluably contributed to the successful writing of this thesis, and to my personal growth. Finally, I acknowledge my parents Floarea and Constantin for dedicating their lives to my support, nurturing and guidance, and for being next to me in my good and bad times. Everything I am, I owe them.

The work in this thesis was supported by the project “Meta-theory of Algebraic Process Theories” (no. 100014021) of the Icelandic Research Fund, Radboud University and a CWI Internship.

Chapter 1

Introduction

One of the research areas of great importance in Computer Science is the study of the semantics of concurrent reactive systems [HP85]. These are systems that compute by interacting with their environment, and typically consist of several parallel components, which execute simultaneously and potentially communicate with each other. Examples of such systems range from rather simple devices such as calculators and vending machines, to programs controlling mechanical devices such as cars, subways or spaceships. In light of their widespread deployment and complexity, the application of rigorous methods for the specification, design and reasoning on the behaviour of reactive systems has always been a great challenge.

One possible approach to formally handle reactive systems is to use a "common language" for describing both the actual implementations and their specifications. When following this technique, checking whether an implementation and its specification describe the same behaviour reduces to proving some notion of equivalence/preorder between their corresponding descriptions over the chosen language. This procedure is also referred to as "equivalence checking".

Intuitively, we say that an implementation complies to its specification whenever the implementation displays only the behaviour allowed by the specification, and nothing more. However, it is important to notice that system verification can be performed at different levels of abstraction, with respect to non-determinism, for example, depending on the context of application. In this regard, we refer to a suite of semantics that are thoroughly studied throughout this thesis, namely: bisimilarity [MP81, Mil89] – the standard notion of behavioural equivalence in concurrency –, the spectrum of decorated trace semantics in van Glabbeek's work [vG01a], and must and may testing semantics [CH89, DH84, Hen88].

Along time, different mathematical frameworks have been exploited for modelling reactive systems and their behaviours, and for deriving efficient verification algorithms for their computer-aided analysis. In the sequel, we provide a short overview on two of such "dual" frameworks: algebra [BS12, Hen88] and coalgebra [JR97, Rut00].

1.1 Algebra

other way around

Algebraic process theories, or "process algebras", have been successfully used as prototype specification languages for reactive systems. Typically, the definition of process algebras consists in providing a syntax and an operational semantics, usually given in terms of so-called Structural Operational Semantics (SOS) rules [Plo04]. Intuitively, SOS is a framework used for describing how programs compute step by step, by emphasising the corresponding state-transformations that occur after the execution of certain actions. Once a desired notion of behavioural equivalence or preorder over processes is fixed, a corresponding sound (and ideally complete) axiomatisation is given. This way, one can establish the conformance of an implementation with its specification in an equational style, without generating the state space of processes, therefore potentially combatting the state explosion problem. We hint, for example, to the works in [ABV94] and [BdV04], where sound and complete axiomatisations for bisimilarity of systems complying to the GSOS [BIM95] format and GSOS with termination, respectively, are provided.

direct equational redeneren los van semantiek

Unfortunately, this approach has low flexibility as regards language modifications. A Axiomatisations are usually shown sound and complete by means of proof techniques that take into account the combinators of the language under consideration; hence new syntactical constructs frequently impose new proofs (from scratch). Consider, for instance, the work in [ACEII11] extending the results in [ABV94] to the case of GSOS with predicates such as termination, divergence and convergence. Even though syntactically trivial, the extension required the construction of a new axiomatisation that had to be proven sound and complete (which is often not a trivial task). → work on formats

However, as soon as an axiomatisation is identified, the implementation of a verification tool based on equational reasoning is almost straightforward. We refer, for example, to the automated tool in [ACGI11] which can be used for reasoning on bisimilarity of systems complying to the extended GSOS format in [ACEII11].

Moreover, in the algebraic setting, SOS rules can be used not only for specifying the behaviour of systems in an intuitive fashion, but also for imposing a series of (syntactic) constraints to guarantee that a certain notion of behavioural equivalence (or preorder) for systems satisfying the aforementioned restrictions is also a (pre)congruence. Semantics which are also (pre)congruences are important from the practical perspective as well. Intuitively, whenever a subcomponent of a system is replaced, showing the equivalence between the new "upgraded" system and the initial one, with respect to a notion of (pre)congruence, reduces to showing the equivalence between the two subsystems that have been interchanged. This way, the complexity of the verification procedure is obviously reduced. In this respect, we refer, for instance, to the GSOS [BIM95] format which guarantees that bisimilarity is a congruence. In related work [BFvG04], precongruence formats for decorated trace semantics [vG01a] were established via modal characterisations of the corresponding preorders.

1.2 Coalgebra

A possible representation of implementations and their specifications is in terms of state machines. These allow for a uniform manipulation of systems such as: streams [Rut05], (non)deterministic and probabilistic automata [RS59, Rab80], Moore [Moo56] and Mealy [Mea55] machines, and labelled transition systems [Kel76].

1.2. Coalgebra

where is the algebra in co-algebra?

Coalgebra [JR97, Rut00] is a recent unifying theory combining ideas from the mathematical theory of dynamical systems and from the theory of state-based computation, and has been successfully applied as a mathematical framework for the study of state-based systems. Intuitively, from the coalgebraic perspective, systems with (possibly) infinite behaviour are represented as *black-box* machines analyzed only according to their observable behaviour. Mathematically, one can describe such a machine in terms of a coalgebra $(X, \delta \colon X \to \mathscr{F}(X))$ consisting of a set (of states) X, and a map δ encapsulating the corresponding behaviour based on a functor $\mathscr{F}$. This map represents the set of observers, or destructors, allowing one to "break" (infinite) system behaviour into analyzable fragments.

Coalgebraic analysis on the behaviours of systems can be performed as follows. First, identify the appropriate functor associated with the class of systems under analysis. Then, reason on the corresponding notion of behavioural equivalence by coinduction [SR11], a proof technique based on bisimulation, already implemented in automated tools [BP13, CGK$^+$13, CPS93a, GLMS11, RL09].

All the systems mentioned above can be coalgebraically modelled in a uniform way, by simply varying the behaviour functor $\mathscr{F}$. For instance, for the case of streams (*i.e.*, infinite words) over an alphabet A, the functor $\mathscr{F}(X) = A \times X$ provides the head of the stream, which is an element of A, and its tail, which is again a stream. Labelled transition systems are intuitively defined by the functor $\mathscr{F}(X) = (\mathscr{P}X)^A$, which for an action labelled in A returns the set of states that can be (non-deterministically) reached after executing that action. More interestingly, note that each functor induces a notion of behavioural equivalence [Rut00]. For streams, for example, this coincides with stream equality, whereas for deterministic automata and labelled transition systems, the corresponding notions of behavioural equivalence are language equivalence and bisimilarity [MP81, Mil89], respectively.

As already stated, verification of systems can be performed at different levels of abstraction, depending on the context of application. The work in this thesis is closely related to the results in [SBBR10]. There it is shown how the generality and modularity of coalgebras can be exploited (via a coalgebraic subset construction) in order to uniformly reason about the behaviour of labelled transition systems in terms of trace, ready or failure equivalence [vG01a], rather than bisimilarity. Moreover, reasoning on the aforementioned equivalences follows "for free" by coinduction, and can be performed in a fully automated fashion using the tool in [BP13].

Even though the coalgebraic setting abstracts from the syntax in process description languages, its generality and uniformity enables also the interplay with syntax-based characterisations of systems. For example, we refer to the works in [Kli09, TP97], where bialgebraic frameworks for deriving congruence rule formats and proving compositionality of various kinds of semantics (such as bisimilarity and decorated trace semantics [vG01a]) were provided based on the so-called *distributive laws* of syntax over behaviour. From a simpler perspective, note that the dynamics of transition systems for process algebras can be coalgebraically characterised (in terms of states and transitions between states) according to the SOS rules expressing their behaviours.

1.3 Aim and approach

Along the research lines mentioned so far, the *aim* of our work is to exploit the strengths of the (co)algebraic framework in modelling reactive systems and reasoning on several types of associated semantics, in a uniform fashion. In particular, we are interested in handling notions of behavioural equivalence/preorder ranging from bisimilarity for systems that can be represented as non-deterministic coalgebras [SBR10], to decorated trace semantics for labelled transition systems and probabilistic systems, and testing semantics for labelled transition systems with internal behaviour. Moreover, we aim at deriving a suite of corresponding verification algorithms suitable for implementation in automated tools.

The *approach* we adopt is based on the following steps.

- First, we focus on the results in [BRS09] introducing a language of expressions for specifying a large class of systems that can be modelled as non-deterministic coalgebras, and a sound and complete axiomatisation for bisimilarity of such systems. The latter include, for example, streams, (non)deterministic automata, Mealy, Moore and labelled transition systems. In [BRS09], systems which are coalgebras of non-deterministic functors are described in a rather algebraic fashion, in terms of a language of expressions derived according to the functor of interest. Then, expressions are shown to have a coalgebraic structure, hence further enabling reasoning on their equivalence by coinduction.

  In our approach, we exploit a combination of algebra and coalgebra, based on interplays such as constructors – destructors, induction – coinduction (both as definition and as proof principles), and congruence – bisimilarity [JR97]. Building on these associations and on the strength of coalgebras in deriving algorithms and tools for the automatic verification of systems, we construct a decision procedure for the bisimilarity of generalised regular expressions [BRS09] (and therefore, of their corresponding non-deterministic systems). This is achieved by providing an algebraic specification for the coalgebra of expressions, and reducing coinduction to an entailment relation between this specification and a suitable set of equations.

  The theory was implemented in CIRC [GLR00, RL09] – an automated theorem prover based on coinduction, successfully used for reasoning on properties of infinite data structures such as streams –, and can be tested online at: `http://goriac.info/tools/functorizer/`.

- Although bisimilarity [MP81, Mil89] is the standard notion of behavioural equivalence in concurrency theory, considerable amount of work has been dedicated to the treatment of decorated trace semantics [vG01a, JS90], and may and must testing semantics [CH89, DH84, Hen88], for instance.

  Studying semantics other than bisimilarity is not only an interesting research subject per se, but is also important from the applicability perspective.

  For example, bisimilarity, which belongs to the class of the so-called "branching time" semantics, can be sometimes too fine for system verification. Therefore, coarser semantics such as the "linear time" semantics might be more appropriate. In this respect, we refer to the work in [vG01b] for a survey on the aforementioned semantic equivalences (and preorders), and for a study on their context of application and advantages.

Semantics coarser than bisimilarity, for example, that are also (pre)congruences, can play an important role in system reduction as well. Consider a scenario in which the correctness of concurrent systems is established according to a property expressed by a set of logical formulae. It would be desirable to use a (pre)congruence preserving such a property for deriving a smaller (reduced) labelled transition system whose components are eventually checked for the aforementioned property. Hence, the coarser the (pre)congruence, the coarser the refinement of the original system.

We refer, for example, to the work in [Val95], where it is shown that trace equivalence [vG01a] is the weakest congruence preserving the property "P may ever execute action a", whereas the so-called "stable failure equivalence" is the weakest deadlock-preserving congruence with respect to any set of Basic Lotos [BB87] operators containing parallel composition.

We also hint to the work in [SBBR10], where trace, failure and readiness semantics [vG01a] were recovered in a coalgebraic setting by applying the generalised powerset construction [SBBR13], which is reminiscent of the determinisation of non-deterministic automata.

Also of interest in concurrency, are must and may testing semantics [DH84, Hen88]. Unlike weak bisimilarity, must testing distinguishes between livelock and deadlock, for instance. This can be useful in practice as, even though internal behaviour of systems do not provide any information to an external observer, it can be desirable to set apart infinite internal computations from the impossibility of performing any further move. In [CH89], an alternative characterisation of may and must testing semantics is based on sequences of observable actions processes can execute. Hence, it is of interest studying a possible connection with the approach in [SBBR10], for a coalgebraic modelling of these semantics.

Motivated by these results and observations, as a second step we provide a uniform coalgebraic modelling of decorated trace, may and must testing semantics via the generalised powerset construction.

- Last, but not least, we exploit the coalgebraic modelling of decorated trace and must testing semantics (which is more interesting than may testing semantics, as it is sensitive to the non-determinism of processes), and devise algorithms for reasoning on the corresponding equivalences and preorders.

  Existing algorithms for the automated checking of these behavioural semantics over finite-state systems rely on the following idea. First, non-deterministic systems are transformed into the so-called (deterministic) "acceptance graphs", by applying a technique which is reminiscent of the determinisation of non-deterministic automata [RS59]. Then, reasoning on the aforementioned semantics on the original non-deterministic systems is reduced to the equivalent problem of reasoning on bisimilarity of the associated acceptance graphs. We refer to [CPS93b, CS96, CDLT08] for examples of automated tools implementing such algorithms.

  In our work, however, the coalgebraic setting enables the construction of verification algorithms which are not available for bisimilarity. More precisely, we build an algorithm based on (Moore-) bisimulations (up-to) [BP13, SR11, San98], which follows as a consequence of the determinisation procedure previously mentioned.

Moreover, we provide a variation of Brzozowski's algorithm [Brz62], by exploiting the abstract coalgebraic theory in [BBRS12].

Our approach is uniform and modular: once the "recipe" for handling failure semantics is established, the almost straightforward extension to non-deterministic systems with internal behaviour enabled shifting to must testing semantics. This is also a consequence of the fact that failure semantics coincides with must testing in the absence of divergence [CH89, Nic87]. Furthermore, the algorithms for reasoning on failure semantics can be easily adapted also for other decorated trace semantics studied in this thesis.

Both the bisimulation-based and Brzozowski's minimisation techniques were implemented in an automated tool, and can be tested online at:
`http://perso.ens-lyon.fr/damien.pous/brz/`.

1.4 Thesis outline

We summarise the content and the main contributions of the thesis.

Chapter 2 provides the basic definitions from coalgebra and recalls the generalised powerset construction, which we will use in our work.

Chapter 3 presents an algorithm to decide whether two (generalised regular) expressions defining systems that can be modelled as non-deterministic coalgebras are bisimilar or not. The aforementioned expressions and an analogue of Kleene's theorem and Kleene algebra, were recently proposed by Silva, Bonsangue and Rutten in [SBR10]. Examples of systems we handle include infinite streams, deterministic automata, Mealy machines and labelled transition systems. The procedure is implemented in the automatic theorem prover CIRC, by reducing coinduction to an entailment relation between an algebraic specification and an appropriate set of equations.
The main contributions are summarised in the table below.

A decision procedure for bisimilarity	
Algebraic modelling of expressions	Figure 3.2
Algebraic encoding of bisimilarity	Corollary 3.3.4
Soundness	Theorem 3.4.2
Decision procedure	Theorem 3.4.3

This chapter is based on the following papers:
[BCG$^+$11] Marcello M. Bonsangue, Georgiana Caltais, Eugen-Ioan Goriac, Dorel Lucanu, Jan J. M. M. Rutten, Alexandra Silva. A decision procedure for bisimilarity of generalised regular expressions. Proc. 13'th Brazilian Symposium on Formal Methods, 2011:226–241.

[BCG$^+$13] Marcello M. Bonsangue, Georgiana Caltais, Eugen-Ioan Goriac, Dorel Lucanu, Jan J. M. M. Rutten, Alexandra Silva. Automatic equivalence proofs for non-deterministic coalgebras. Science of Computer Programming, 2013:1324–1345.

Chapter 4 provides the coalgebraic handling of a series of semantics on transition systems in a uniform modular fashion, by employing the generalised powerset construction introduced by Silva, Bonchi, Bonsangue and Rutten in [SBBR13]. As we shall see, this construction yields a notion of minimal representatives for (i) decorated trace equivalences for labelled transition systems (LTS's) [Kel76] and generative probabilistic systems (GPS's) [vG01a, JS90] and, (ii) must and may testing semantics for non-deterministic systems with internal behaviour [CH89, DH84, Hen88]. As a consequence, reasoning on the aforementioned notions of behavioural equivalence/preorder can be performed in terms of (Moore-) bisimulations. Moreover, we show how the spectrum of decorated trace semantics can be recovered from the coalgebraic modelling.
The main contributions are listed in the following table.

Decorated traces and testing semantics coalgebraically	
Correctness of the coalgebraic modelling of:	
Ready & failure semantics for LTS's	Theorem 4.1.3
(Complete) trace semantics for LTS's	Theorem 4.1.9
Possible-futures semantics for LTS's	Theorem 4.1.12
Ready & failure trace semantics for LTS's	Theorem 4.1.16
Ready & (maximal) failure semantics for GPS's	Theorem 4.2.5
(Maximal) trace semantics for GPS's	Theorem 4.2.7
May testing semantics	Theorem 4.6.2
Must testing semantics	Theorem 4.6.7
Recovering the spectrum	Lemma 4.5.1 Lemma 4.5.2

This chapter is based on the papers:
[BBC+12] Filippo Bonchi, Marcello Bonsangue, Georgiana Caltais, Jan Rutten, Alexandra Silva. Final semantics for decorated traces. Electronic Notes in Theoretical Computer Science, 2012:73–86. Proc. Mathematical Foundations of Programming Semantics 2012.

Chapter 5 focuses on checking language equivalence (or inclusion) of finite automata. This is a classical problem in computer science, which has recently received a renewed interest and found novel and more effective solutions, such as the approaches based on antichains [ACH+10, WDHR06] or bisimulations up-to [BP13, RBR13, SR11, San98]. Several notions of equivalence (or preorder) have been proposed for the analysis of concurrent systems. Some approaches reduce the problem of checking these equivalences to the problem of checking bisimilarity. In this chapter, we tackle this challenge differently, and propose to "adapt" algorithms for language semantics. More precisely, we introduce an analogue of Brzozowski's algorithm and HKC – an optimisation of Hopcroft and Karp's algorithm [HK71] based on bisimulations up-to –, for checking must testing equivalence and preorder as well as failure equivalence. To achieve this transfer of technology (from language to must/failure semantics), we take a coalgebraic look at the problem at hand. The table below summarises the main contributions of this chapter.

Algorithms for decorated trace and must testing semantics	
HKC for failure semantics	Sections 5.2.1, 5.2.3
Brzozowski for failure semantics	Sections 5.2.4, 5.2.5
HKC for must testing semantics	Sections 5.2.2, 5.2.3
Brzozowski for must testing semantics	Sections 5.2.6, 5.2.7

This work is based on the paper:
Filippo Bonchi, Georgiana Caltais, Damien Pous, Alexandra Silva. Brzozowski's and Up-to Algorithms for Must Testing. To appear in volume 8301 of the Lecture Notes in Computer Science series.

1.5 Related work

The contributions of the thesis stem from the underlying idea of formally specifying and verifying concurrent reactive systems in a uniform fashion, both in theory and practice, by exploiting the (co)algebraic framework.

On the one hand, we build our work based on previous results originating from the correspondence between regular expressions and finite deterministic automata (DFA's) – two of the most basic structures in Computer Science –. Kleene's theorem [Kle56] gives a fundamental correspondence between these two structures: each regular expression denotes a language that can be recognised by a DFA and, conversely, the language accepted by a DFA can be specified by a regular expression. A sound and complete axiomatisation (later refined by Kozen in [Koz91, Koz01]) for proving the equivalence of regular expressions was introduced by Salomaa [Sal66], and an extension for the case of LTS's modulo bisimilarity was derived by Milner in [Mil84].
For coalgebras of a large class of functors, a language of regular expressions, a corresponding generalisation of Kleene's theorem, and a sound and complete axiomatisation for the associated notion of behavioural equivalence were introduced in [SBR10]. Both the language of expressions and their axiomatisation were derived, in a modular fashion, from the functor defining the type of the system.
One of the contributions of the thesis consists in a decision procedure for bisimilarity of generalised regular expressions in [SBR10], implemented in the coinductive theorem prover CIRC [GLR00, RL09]. More explicitly, we derived an encoding of generalised regular expressions and their coalgebraic structure into CIRC-compatible constructs, and implemented a tool allowing this translation automatically, hence enabling the automated reasoning on bisimilarity of non-deterministic coalgebras.
We further mention some of the existing coalgebraic based tools for proving bisimilarity and the main differences with our tool. CoCasl [HMS05] and CCSL [RTJ01] are tools that can generate proof obligations for theorem provers from coalgebraic specifications. In [HMS05] several tactics for interactive and automatic bisimulation building are implemented in Isabelle/HOL and are used to derive bisimilarities for translated specifications from CoCasl. The main difference between our tool and CoCasl or CCSL is that, given a functor, the tool derives a specification language for which equivalence is decidable (that is, it is automatic and not interactive). CIRC [GLR00, RL09], on top of which the current tool is built, is based on hidden logic [Ros00] and uses a partial decision procedure for proving bisimilarities via implicit construction of bisimulations. Our tool can be seen as an extension of CIRC to a fully automatic theorem prover for the class of non-deterministic coalgebras. We stress the fact that the focus of our work is on a language for which equivalence is decidable. Tools such as CoCasl, CCSL or CIRC have a more expressive language, where one can, for instance, specify streams, which in our language could not be specified (intuitively, the streams we can specify in our language are eventually periodic). In those tools decidability of equivalence can, however, not be guaranteed.

On the other hand, we exploit the coalgebraic framework in order to provide a uniform handling of a suite of semantics, other than bisimilarity. More explicitly, we are interested in deriving coalgebraic characterisations and algorithms suitable for implementation for: decorated trace semantics in the context of LTS's and GPS's as introduced in [vG01a, JS90], and testing semantics for LTS's with internal behaviour as given in [CH89].
In the recent past, some of the decorated trace semantics in van Glabbeek's spectrum

have been cast in the coalgebraic framework. Notably, trace semantics of LTS's was widely studied [HJS07, LPW00, SBBR10] and, more recently, (complete) trace, ready and failure semantics were recovered in [SBBR13] via a coalgebraic generalisation of the classical powerset construction [CHL03, Len99, SBBR10]. A coalgebraic characterisation of the spectrum was also attempted in [Mon08]
Since the introduction of process calculi, a lot of research has also been devoted to the analysis of testing semantics [DH84]. Intuitively, with respect to a fixed set of tests, two systems are deemed to be equivalent if they pass exactly the same tests.
In [CH89], a trace-based alternative characterisation of may and must testing was given. Based on this approach, we provide a coalgebraic modelling of the aforementioned semantics via the generalised powerset construction. Another coalgebraic outlook on must testing is presented in [BG06] which introduces a fully abstract coalgebraic semantics for CSP. The main difference with our work consists in the fact that [BG06] builds a coalgebra from the syntactic terms of CSP, while here we build a coalgebra starting from LTS's. As a further coalgebraic approach to testing, it is worth mentioning test-suites [Kli04], which tackle the semantics in van Glabbeek's spectrum [vG01a], but not must testing.
The problem of automatically reasoning on decorated trace and testing semantics of LTS's is an interesting research topic per se. One possible approach, which is reminiscent of the determinisation of non-deterministic automata, consists in deriving deterministic-like systems for which checking bisimilarity coincides with reasoning on the aforementioned semantics in the original LTS's. Several bisimulation-based algorithms are implemented in tools such as the ones in [CPS93b, CS96, CDLT08]. We also refer to the more recent work in [BP13], where the determinised automata are related based on bisimulations up-to [SR11, San98]. The advantage of this procedure is that, in most cases, building the bisimulations up-to requires visiting only portions of the automata. The partial exploration is also the key feature of the antichain algorithm [WDHR06] for reasoning on language equivalence of non-deterministic finite automata.
The best-known algorithm for minimising LTS's with respect to bisimilarity is the so-called partition refinement [KS83, PT87], which is analogous to Hopcroft's minimisation algorithm [Hop71] for deterministic automata with respect to language equivalence. Last, but not least, we refer to Brzozowski's minimisation algorithm [Brz62], which has been provided with a coalgebraic understanding in [BBRS12].
Along this line of research, in Chapter 5 we introduce an analogue of Brzozowski's algorithm and an algorithm based on bisimulations up-to for failure and must testing semantics.

Chapter 2

Preliminaries

In this chapter we recall the basic definitions for sets and coalgebras that are needed in the rest of the thesis. We also introduce the coalgebraic modelling of the (generalised) powerset construction. We assume the reader is familiar with basic notions from category theory. We refer the interested reader to [Rut00] and [Awo10] for more information on coalgebras and category theory, respectively.

2.1 Sets

Let **Set** denote the category of sets (represented by capital letters $X, Y, \ldots$) and functions (represented by lower case letters $f, g, \ldots$). We write Y^X for the family of functions from X to Y and $\mathcal{P}_\omega(X)$ for the collection of finite subsets of a set X. The product of two sets X, Y is written as $X \times Y$ and has the projections functions π_1 and π_2: $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$. We define $X \oplus Y = X \uplus Y \uplus \{\bot, \top\}$ where $\uplus$ is the disjoint union of sets, with injections $X \xrightarrow{\kappa_1} X \uplus Y \xleftarrow{\kappa_2} Y$. Note that the set $X \oplus Y$ is different from the classical coproduct of X and Y (which we shall denote by $X + Y$), because of the two extra elements $\bot$ and $\top$. These extra elements are used to represent, respectively, underspecification and inconsistency in the specification of some systems.

For each of the operations defined above on sets, there is an analogous one on functions. For the sake of brevity, we first introduce the notation $i \in \overline{1,n}$ as a shorthand for $i \in \{1, \ldots, n\}$. Let $f : X \to Y$, $f_1 : X \to Y$ and $f_2 : Z \to W$. We define the following operations:

$$\begin{array}{ll}
f_1 \times f_2 : X \times Z \to Y \times W & f_1 \oplus f_2 : X \oplus Z \to Y \oplus W \\
(f_1 \times f_2)(x, z) = \langle f_1(x), f_2(z) \rangle & (f_1 \oplus f_2)(c) = c,\ c \in \{\bot, \top\} \\
 & (f_1 \oplus f_2)(\kappa_i(x)) = \kappa_i(f_i(x)),\ i \in \overline{1,2} \\
f^A : X^A \to Y^A & \mathcal{P}_\omega(f) : \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(Y) \\
f^A(g) = f \circ g & \mathcal{P}_\omega(f)(X_1) = \{ y \in Y \mid f(x) = y, x \in X_1 \}
\end{array}$$

Note that in the definition above we are using the same symbols introduced for the operations on sets. It will always be clear from the context which operation is being used.

2.2 Coalgebras

The examples handled throughout this thesis live in the standard setting of sets and functions. We therefore define our formal frameworks for modelling and reasoning on behavioural equivalence of systems based on coalgebras of functors on **Set**.

2.2.1 Definition (Coalgebra). A *coalgebra* is a pair $(S, f : S \to \mathscr{F}(S))$, where S is a set of states and $\mathscr{F} : \mathbf{Set} \to \mathbf{Set}$ is a functor. ♣

The functor $\mathscr{F}$, together with the function f, determines the *transition structure* (or dynamics) of the coalgebra [Rut00], also referred to as *$\mathscr{F}$-coalgebra*.
A coalgebra (S, f) is *finite* if S is a finite set.

2.2.2 Definition (Coalgebra homomorphism). A *homomorphism* $h\colon (S, f) \to (T, g)$ from an $\mathscr{F}$-coalgebra (S, f) to an $\mathscr{F}$-coalgebra (T, g), is a function $h\colon S \to T$ making the following diagram commute:

$$\begin{tikzcd}
S \arrow[r, "h"] \arrow[d, "f"'] & T \arrow[d, "g"] \\
\mathscr{F}(S) \arrow[r, "\mathscr{F}(h)"'] & \mathscr{F}(T)
\end{tikzcd}
\qquad g \circ h = \mathscr{F}(h) \circ f$$

♣

2.2.3 Definition (Coalgebra isomorphism). A coalgebra homomorphism $i\colon S \to T$ is a *coalgebra isomorphism* if there exists a coalgebra homomorphism $j\colon T \to S$ such that $i \circ j = id_T$ and $j \circ i = id_S$.

2.2.4 Definition (Final coalgebra). An $\mathscr{F}$-coalgebra (Ω, ω) is *final* if for any $\mathscr{F}$-coalgebra (S, f) there exists a unique $\mathscr{F}$-coalgebra homomorphism

$$[\![-]\!] : (S, f) \to (\Omega, \omega):$$

$$\begin{tikzcd}
S \arrow[r, dashed, "{[\![-]\!]}"] \arrow[d, "f"'] & \Omega \arrow[d, "\omega"] \\
\mathscr{F}(S) \arrow[r, dashed, "{\mathscr{F}([\![-]\!])}"'] & \mathscr{F}(\Omega)
\end{tikzcd}
\qquad \omega \circ [\![-]\!] = \mathscr{F}([\![-]\!]) \circ f$$

♣

Note that not all functors admit final coalgebras. However, it was shown in [Rut00] that such coalgebras exist for the class of bounded functors [GS02]. A functor $\mathscr{F}$ is *bounded* if there are sets B and A and a surjective natural transformation from $B \times (-)^A$ to $\mathscr{F}$ (Theorem 4.7 in [GS02]). Moreover, final coalgebras, if they exist, are unique up to isomorphism.
Intuitively, a final $\mathscr{F}$-coalgebra (Ω, ω) represents the universe of all possible *behaviours* of $\mathscr{F}$-coalgebras (S, f). The unique homomorphism $[\![-]\!]$ maps each element of S to its behaviour. Using this mapping, behavioural equivalence can be defined as follows.

2.2.5 Definition (Behavioural equivalence). Let $\mathcal{F}$ be a functor that admits final coalgebras. For any two $\mathcal{F}$-coalgebras (S,f) and (T,g), $s \in S$ and $t \in T$ are *behaviourally equivalent*, written $s \sim_{\mathcal{F}} t$, if and only if they have the same behaviour, that is:

$$s \sim_{\mathcal{F}} t \text{ iff } [\![s]\!] = [\![t]\!]. \tag{2.1}$$

Coalgebras provide a useful technique for proving behavioural equivalence, namely, bisimulation [AM89].

2.2.6 Definition (Bisimulation). Let (S,f) and (T,g) be two $\mathcal{F}$-coalgebras. A relation $R \subseteq S \times T$ is a *bisimulation* if there exists a map $\alpha\colon R \to \mathcal{F}(R)$ such that the projections $\pi_1\colon R \to S$ and $\pi_2\colon R \to T$ are coalgebra homomorphisms, *i.e.*, they make the following diagram commute:

$$\begin{tikzcd}
S \arrow[d, "f"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\alpha"'] & T \arrow[d, "g"] \\
\mathcal{F}(S) & \mathcal{F}(R) \arrow[l, "\mathcal{F}(\pi_1)"] \arrow[r, "\mathcal{F}(\pi_2)"'] & \mathcal{F}(T)
\end{tikzcd}$$

♣

The following alternative definition of bisimulation, sometimes more appropriate for the proofs, was given in [HJ98]: a relation $R \subseteq S \times T$ is a bisimulation if and only if

$$(s,t) \in R \Rightarrow (f(s), g(t)) \in \overline{\mathcal{F}}(R)$$

where $\overline{\mathcal{F}}(R)$ is defined as

$$\overline{\mathcal{F}}(R) = \{(\mathcal{F}(\pi_1)(x), \mathcal{F}(\pi_2)(x)) \mid x \in \mathcal{F}(R)\} \tag{2.2}$$

If two states are bisimilar, and a final coalgebra exists, then they are behaviourally equivalent.
In [Rut00], it was shown that under certain conditions on $\mathcal{F}$ (which are met by all the functors considered in this thesis), bisimulations are a *sound and complete proof technique* for behavioural equivalence. Namely, by **coinduction** it holds that:

$$s \sim_{\mathcal{F}} t \text{ iff there exists a bisimulation } R \text{ such that } s\,R\,t. \tag{2.3}$$

For simplicity, we abuse the notation and write $s \sim_{\mathcal{F}} t$ whenever there exists a bisimulation relation containing (s,t), and we call $\sim_{\mathcal{F}}$ the bisimilarity relation.
Note that different functors $\mathcal{F}$ induce different notions of behavioural equivalence. For the case of streams, deterministic automata, and finite labelled transition systems, for example, behavioural equivalence corresponds to stream equality, language equivalence and the standard notion of bisimilarity by Milner and Park [MP81, Mil89], respectively.
For more insight on the coalgebraic framework introduced in this section, we further provide the coalgebraic modelling of deterministic and Moore automata (extensively used in Chapter 4 and Chapter 5).

2.2.7 Example. A *deterministic automaton* (DA) is a pair $(X, \langle o,t\rangle)$, where X is a (possibly infinite) set of states and $\langle o,t\rangle\colon X \to 2 \times X^A$ is a function with two components: o, the output function, determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each letter a in the input alphabet A the next state. Note that here 2 stands for the set with two elements $\{0,1\}$.

DA's are coalgebras for the functor $\mathcal{D}(X) = 2 \times X^A$. The final coalgebra of this functor is $(2^{A^*}, \langle \epsilon, (-)_a \rangle)$ where 2^{A^*} is the set of languages over A and $\langle \epsilon, (-)_a \rangle$, given a language L, determines whether or not the empty word ε is in the language ($\epsilon(L) = 1$ or $\epsilon(L) = 0$, respectively) and, for each input letter a, returns the *derivative* of L: $L_a = \{w \in A^* \mid aw \in L\}$.
From any DA, there is a unique map $[\![-]\!]$ into 2^{A^*} which assigns to each state its behaviour (that is, the language that the state recognises) [Rut98].

$$\begin{tikzcd}
X \arrow[r, dashed, "{[\![-]\!]}"] \arrow[d, "{\langle o,t\rangle}"'] & 2^{A^*} \arrow[d, "{\langle \epsilon,(-)_a\rangle}"] \\
2 \times X^A \arrow[r, dashed, "{id \times [\![-]\!]^A}"'] & 2 \times (2^{A^*})^A
\end{tikzcd}
\qquad
\begin{array}{l}
[\![x]\!](\varepsilon) = o(x) \\
[\![x]\!](aw) = [\![t(x)(a)]\!](w)
\end{array}$$

Behavioural equivalence for the functor $\mathcal{D}$ coincides with the classical language equivalence of automata: given a deterministic automaton $(S, \langle o, t \rangle)$, two states $x, y \in S$ are said to be *language equivalent* if and only if they accept the same language. ♠

We further provide the coalgebraic modelling of Moore automata – a generalisation of DA's – which, as we shall later see, will enable shifting from language equivalence to the context of decorated trace semantics.

2.2.8 Example. *Moore automata* with inputs in A and outputs in B are coalgebras for the functor $\mathcal{M}(X) = B \times X^A$, that is pairs $(X, \langle o, t \rangle)$ where X is a set, $t : X \to X^A$ is the transition function (like for DA) and $o : X \to B$ is the output function which maps every state to its output. Thus DA can be seen as a special case of Moore automata where $B = 2$.
The final coalgebra for $\mathcal{M}$ is $(B^{A^*}, \langle \epsilon, (-)_a \rangle)$ where B^{A^*} is the set of all functions $\varphi : A^* \to B$, $\epsilon : B^{A^*} \to B$ maps each φ into $\varphi(\varepsilon)$ and $(-)_a : B^{A^*} \to (B^{A^*})^A$ is defined for all $\varphi \in B^{A^*}$, $a \in A$ and $w \in A^*$ as $(\varphi)_a(w) = \varphi(aw)$.

$$\begin{tikzcd}
X \arrow[r, dashed, "{[\![-]\!]}"] \arrow[d, "{\langle o,t\rangle}"'] & B^{A^*} \arrow[d, "{\langle \epsilon,(-)_a\rangle}"] \\
B \times X^A \arrow[r, dashed, "{id \times [\![-]\!]^A}"'] & B \times (B^{A^*})^A
\end{tikzcd}
\qquad
\begin{array}{l}
[\![x]\!](\varepsilon) = o(x) \\
[\![x]\!](aw) = [\![t(x)(a)]\!](w)
\end{array}$$

Hence, reasoning on behavioural equivalence of Moore automata reduces to checking equality of functions. ♠

2.3 The generalised powerset construction

Sometimes, it is interesting to consider other equivalences than $\sim_{\mathcal{F}}$ for reasoning about $\mathcal{F}$-coalgebras. This is the case for non-deterministic automata (NDA's), for which language equivalence is often the intended semantics, instead of bisimilarity. NDA's are coalgebras for the functor $\mathcal{N}(X) = 2 \times (\mathcal{P}_\omega(X))^A$, where $\mathcal{P}_\omega$ stands for the finite powerset, and bisimilarity, which we denote by $\sim_{\mathcal{N}}$, is strictly included in language equivalence. This can be achieved by applying the classical *powerset construction* [RS59] for determinising non-deterministic automata, which can be briefly summarised as follows.
Consider an NDA, which is a coalgebra

$$(X, \langle o, t \rangle : X \to 2 \times (\mathcal{P}_\omega X)^A)$$

where (similarly to the case of DA's in Example 2.2.7): o is the output function and determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$), t is the transition function returning for each input letter a the set of next states, and 2 stands for the set with two elements $\{0, 1\}$.
The powerset construction derives a DA

$$(\mathscr{P}_\omega X, \langle o^\sharp, t^\sharp \rangle : \mathscr{P}_\omega X \to 2 \times (\mathscr{P}_\omega X)^A),$$

by associating to each state $x \in X$ of the NDA, a state $\{x\} \in \mathscr{P}_\omega X$. The new output and transition functions are:

$$\begin{array}{rcl} o^\sharp(Y) & = & \bigsqcup_{y \in Y} o(y) \\ t^\sharp(Y)(a) & = & \bigsqcup_{y \in Y} t(y)(a) \end{array} \tag{2.4}$$

where $\bigsqcup$ is used to represent both the "Boolean or" and the set union. Intuitively, $\bigsqcup$ stands for the join operation corresponding to the semilattice with carrier $\{0, 1\}$, and the one with carrier $\mathscr{P}_\omega S$, with $S \in \mathbf{Set}$, respectively. The final coalgebra of the DA is the set of languages 2^{A^*} over A, and the semantic map

$$[\![-]\!] : \mathscr{P}_\omega X \to 2^{A^*}$$

associates to each $\{x\}$ the language $[\![\{x\}]\!]$ accepted by x, and is defined as introduced in Example 2.2.7 in the previous section. Consequently, reasoning on language equivalence of two states x_1 and x_2 of an NDA reduces to identifying a bisimulation R relating $\{x_1\}$ and $\{x_2\}$ in the corresponding DA:

$$[\![\{x_1\}]\!] = [\![\{x_2\}]\!] \text{ iff } \{x_1\} \, R \, \{x_2\}. \tag{2.5}$$

Based on these observations, we refer to the *generalised powerset construction* [CHL03, Len99, SBBR10] for coalgebras $f : X \to \mathscr{F}T(X)$ for a functor $\mathscr{F}$ and a monad T. Intuitively, this construction applies to the context of NDA's by simply instantiating T with $\mathscr{P}_\omega$ and $\mathscr{F}$ with $2 \times (-)^A$ (more details are provided later on in this section, in Example 2.3.4). Monads are used to encompass computational effects such as non-determinism ($T(X) = \mathscr{P}_\omega(X)$) or partiality ($T(X) = 1 + X$, where $1 = \{*\}$ stands for termination). They come equipped with two operations: unit (η) and multiplication (μ). Intuitively, η enables the embedding of any value into the monad structure, whereas μ allows to collapse several levels of computational effects. For instance, the unit and multiplication of the powerset monad $T = (\mathscr{P}_\omega, \eta, \mu)$ are defined as follows:

$$\begin{array}{ll} \eta_X : X \to \mathscr{P}_\omega X & \mu_X : \mathscr{P}_\omega(\mathscr{P}_\omega X) \to \mathscr{P}_\omega X \\ \eta_X(x) = \{x\} & \mu_X(U) = \bigcup_{S \in U} S. \end{array} \tag{2.6}$$

We further provide an overview of the notions of a monad and algebras of a monad, and a series of intuitions for their integration into the context of the generalised powerset construction.
First recall that, given two functors $\mathscr{F}$ and $\mathscr{G}$ on **Set**, a *natural transformation* $\lambda : \mathscr{F} \Rightarrow \mathscr{G}$ is a family of functions $\lambda_X : \mathscr{F}(X) \to \mathscr{G}(X)$ such that, for all functions $f : X \to Y$, the following holds:

$$\lambda_Y \circ \mathscr{F}(f) = \mathscr{G}(f) \circ \lambda_X.$$

2.3.1 Definition (Monad). Let T be a functor on **Set**. A *monad* is a triple (T,η,μ) where $\eta\colon Id \Rightarrow T$ and $\mu\colon T^2 \Rightarrow T$ are two natural transformations, called *unit* and *multiplication*, respectively, such that the following diagrams commute:

$$\begin{tikzcd}
T \arrow[r, Rightarrow, "T\eta"] \arrow[dr, Rightarrow, "id"'] & T^2 \arrow[d, Rightarrow, "\mu"] & T \arrow[l, Rightarrow, "\eta T"'] \arrow[dl, Rightarrow, "id"] \\
 & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
T^3 \arrow[r, Rightarrow, "T\mu"] \arrow[d, Rightarrow, "\mu T"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\
T^2 \arrow[r, Rightarrow, "\mu"'] & T
\end{tikzcd}$$

♣

2.3.2 Definition (Algebra of a monad). An *algebra* of a monad (T,η,μ), or a T-*algebra*, is a pair $(X, h\colon T(X) \to X)$ satisfying the laws

$$h \circ \eta = id \quad h \circ \mu = h \circ Th.$$

♣

Intuitively, these laws show how to eliminate the computational effects by propagating the operation h throughout the monadic structure.
For the case of the powerset monad defined in (2.6), for example, T-algebras are semilattices (with bottom). Consider a join semilattice $(S, \bigsqcup)$ with 0 the least element. Showing that S carries an algebra structure consists in proving that there exists $h\colon \mathcal{P}_\omega(S) \to S$ satisfying the laws in Definition 2.3.2. It is easy to check that by taking

$$h(U) = \bigsqcup_{u\in U} u,$$

with $U \subseteq S$, we get the appropriate map.
The proof is as follows. Consider $u \in S$ and $\Psi \subseteq \mathcal{P}_\omega(\mathcal{P}_\omega S)$. Then:

$$\begin{aligned}
(h\circ\eta)(u) &= h(\{u\}) \\
&= u \\
(h\circ\mu)(\Psi) &= h(\mu(\Psi)) \\
&= h(\bigcup_{U_i\in\Psi} U_i) \\
&= \bigsqcup_{\substack{U_i\in\Psi \\ u_j\in U_i}} u_j \\
(h\circ\mathcal{P}_\omega h)(\Psi) &= h(\mathcal{P}_\omega h(\Psi)) \\
&= h(\{h(U_i) \mid U_i \in \Psi\}) \\
&= h(\{\bigsqcup_{u_j\in U_i} u_j \mid U_i\in\Psi\}) \\
&= \bigsqcup_{\substack{U_i\in\Psi \\ u_j\in U_i}} u_j
\end{aligned}$$

The first set of equalities is associated to the law $h \circ \eta = id$ in Definition 2.3.2 and, intuitively, states that eliminating the non-determinism from a singleton set $\{u\}$ consists in simply considering the value u. The last two sets of equalities correspond to the law $h \circ \mu = h \circ Th$. Intuitively, they show that eliminating two levels of non-determinism captured within a set $\Psi \subseteq \mathscr{P}_\omega(\mathscr{P}_\omega S)$ can be performed in two different ways: (a) first flatten Ψ and then return the join of the elements in the resulted set, or (b) first compute the joins $h(U_i)$ of the elements of sets $U_i \in \Psi$ and then return the join of all such $h(U_i)$'s.

2.3.3 Definition (Algebra homomorphism). Let (T, η, μ) be a monad. A function $f : X \to Y$ is a *homomorphism* between two T-algebras $(X, h\colon T(X) \to X)$ and $(Y, g\colon T(Y) \to Y)$ if it makes the following diagram commute:

$$\begin{tikzcd}
T(X) \arrow[r, "T(f)"] \arrow[d, "h"'] & T(Y) \arrow[d, "g"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

♣

These are the key ingredients exploited in [SBBR13] in order to derive the **generalised powerset construction** for coalgebras $f : X \to \mathscr{F}T(X)$ for a functor $\mathscr{F}$ and a monad T, with the proviso that $\mathscr{F}T(X)$ is a T-algebra, and $\mathscr{F}$ has a final coalgebra (Ω, ω), as summarised in the following commuting diagram:

$$\begin{tikzcd}
X \arrow[r, "\eta"] \arrow[d, "f"'] & T(X) \arrow[r, dashed, "\llbracket - \rrbracket"] \arrow[dl, "f^\sharp"] & \Omega \arrow[d, "\omega"] \\
\mathscr{F}T(X) \arrow[rr, dashed, "\mathscr{F}(\llbracket - \rrbracket)"'] & & \mathscr{F}(\Omega)
\end{tikzcd} \tag{2.7}$$

We refer the interested reader to [SBBR13] where all the technical details are explored and many instances of the construction are shown.
At an intuitive level, the coalgebra $f : X \to \mathscr{F}T(X)$ is extended to $f^\sharp\colon T(X) \to \mathscr{F}T(X)$ which, for two elements $x_1, x_2 \in X$, enables checking their "$\mathscr{F}$-equivalence with respect to the monad T" ($\eta(x_1) \sim_{\mathscr{F}} \eta(x_2)$) rather than checking their $\mathscr{F}T$-equivalence. Formally, assuming that $\mathscr{F}T(X)$ is a T-algebra, $f^\sharp$ is the unique algebra map between $(T(X), \mu)$ and $(\mathscr{F}T(X), h)$ (where h is a given algebra structure on $\mathscr{F}T(X)$) such that

$$f^\sharp = h \circ Tf.$$

Remark 1 *Based on (2.1) and (2.3), verifying $\mathscr{F}$-behavioural equivalence of two states x_1, x_2 in a coalgebra $(T(X), f^\sharp)$ consists in identifying a bisimulation R relating $\eta(x_1)$ and $\eta(x_2)$:*

$$\llbracket \eta(x_1) \rrbracket = \llbracket \eta(x_2) \rrbracket \text{ iff } \eta(x_1)\, R\, \eta(x_2). \tag{2.8}$$

2.3.4 Example. Consider again the case of NDA's which are coalgebras

$$(X, \langle o, t \rangle : X \to 2 \times (\mathscr{P}_\omega X)^A),$$

♠

as introduced in the beginning of this section. Observe that $\mathscr{P}(X)$ and $2 \cong \mathscr{P}(1)$ are (join) semilattices, which are algebras of the powerset monad (here 1 stands for the singleton set $\{*\}$). Moreover, product and exponentiation preserve the algebra structure, hence guaranteeing that $2 \times (\mathscr{P}_\omega(X))^A$ is an algebra for $\mathscr{P}_\omega$ as well.

At this point it is easy to see that the generalised powerset construction applies to the context of NDA's by simply instantiating T with $\mathscr{P}_\omega$ and $\mathscr{F}$ with $2 \times (-)^A$. It follows that the operation $\langle o, t \rangle$ of the NDA can be uniquely extended to $\langle o^\sharp, t^\sharp \rangle$, as in (2.4), in a deterministic setting. The language recognised by a non-deterministic state x can be defined by precomposing the unique morphism $[\![-]\!] : \mathscr{P}_\omega X \to 2^{A^*}$ with the unit η of $\mathscr{P}_\omega$. This enables reasoning on language equivalence of states of NDA's in terms of bisimulations, as in (2.5). Recall that for the case of deterministic LTS's, language equivalence and bisimilarity coincide [Eng85].
As a last aspect, note that the set of languages 2^{A^*} can be provided a join semilattice structure by considering the union of languages as the binary operation and the empty language as the least element. It can be easily shown (by induction on words $w \in A^*$) that the semantic map $[\![-]\!]$ is a join semilattice homomorphism (or, equivalently, a $\mathscr{P}_\omega$-algebra homomorphism).
More generally, the semantic map $[\![-]\!]$ in (2.7) is a T-algebra homomorphism whenever there exists a distributive law $T\mathscr{F} \Rightarrow \mathscr{F}T$, which guarantees that the carrier Ω of the final coalgebra is a T-algebra as well (see Proposition 4 in [JSS12]).

Chapter 3

Deciding bisimilarity

The results in this chapter are based on the work in [SBR10], where a language of regular expressions for specifying a large class of systems that can be modelled as non-deterministic coalgebras, and a sound and complete axiomatisation for the corresponding notions of behavioural equivalence were introduced.

Our contribution consists in a novel method for checking bisimilarity of generalised regular expressions using the coinductive theorem prover CIRC [GLR00, RL09]. The main novelty of the method lies in the generality of the systems it can handle; examples include streams of real numbers, Mealy machines and labelled transition systems. More precisely, our approach deals with systems that can be represented as locally finite coalgebras or, equivalently, coalgebras for which the smallest subcoalgebra generated by a state is finite [Rut00].

CIRC is a metalanguage application implemented in Maude [CDE$^+$07], and its target is to prove properties over infinite data structures. It has been successfully used for checking the equivalence of programs, and trace equivalence and strong bisimilarity of processes. The tool may be tested online and downloaded from:
`https://fmse.info.uaic.ro/tools/Circ/`.

Determining whether two expressions are equivalent is important in order to be able to compare behavioural specifications. In the presence of a sound and complete axiomatisation one can determine equivalence using algebraic reasoning. A coalgebraic perspective on regular expressions has however provided a more operational/algorithmic way of checking equivalence: one constructs a bisimulation relation containing both expressions. The advantage of the bisimulation approach is that it enables automation since the steps of the construction are fairly mechanic and require almost no ingenuity. We illustrate this with an example, to give the reader the feeling of the more algorithmic nature of bisimulation. We want to stress however that we are not underestimating the value of an algebraic treatment of regular expressions: on the contrary, as we will show later, the axiomatisation plays an important role in guaranteeing termination of the bisimulation construction and is therefore crucial for the main result of this chapter.

We show below a proof of the sliding rule: $a(ba)^* \equiv (ab)^*a$. The algebraic proof, using the rules and equations of Kleene algebra, needs to show the two containments

$$a(ba)^* \leq (ab)^*a \qquad \text{and} \qquad (ab)^*a \leq a(ba)^*$$

and it requires some ingenuity in the choice of the equation applied in each step. We show the proof for the first inequality, the other follow a similar proof pattern.

$$\begin{array}{lll}
 & a(ba)^* \leq (ab)^*a & \\
\Leftarrow & a+(ab)^*a(ba) \leq (ab)^*a & \text{right-star rule [Koz91]: } \begin{array}{l} b+xa \leq x \Rightarrow \\ ba^* \leq x \end{array} \\
\Longleftrightarrow & (1+(ab)^*ab)a \leq (ab)^*a & \text{associativity and distributivity} \\
\Longleftrightarrow & (ab)^*a \leq (ab)^*a & \text{right expansion rule: } 1+r^*r = r^*
\end{array}$$

For the coalgebraic proof, we build incrementally, and rather mechanically, a bisimulation relation containing the pair $(a(ba)^*, (ab)^*a)$. We start with the pair we want to prove equivalent and then we close the relation with respect to syntactic language derivatives, also known as *Brzozowski derivatives*. In the current example, the bisimulation relation would contain three pairs:

$$R = \{(a(ba)^*, (ab)^*a), ((ba)^*, b(ab)^*a+1), (0,0)\}$$

where 1 and 0 are, respectively, the regular expressions denoting the language containing only the empty word and the empty language. In constructing this relation, no decisions were made, and hence the suitability of bisimulation construction as an automatic technique to prove equivalence of regular expressions.

The main contributions of this chapter can be summarised as follows. We present a decision procedure to determine equivalence of generalised regular expressions, which specify behaviours of many types of transition systems, including Mealy machines, labelled transition systems and infinite streams. We illustrate the decision procedure we devised by applying it to several examples. As a vehicle of implementation, we choose CIRC, a coinductive theorem prover which has already been explored for the construction of bisimulations. To ease the implementation in CIRC, we present the algebraic specifications' counterpart of the coalgebraic framework of the generalised regular expressions mentioned above. This enables us to automatically derive algebraic specifications that model the language of expressions, and to define an appropriate equational entailment relation which mimics our decision procedure for checking behavioural equivalence of expressions. The implementation of both the algebraic specification and the entailment relation in CIRC allows for automatic reasoning on the equivalence of expressions.

Organisation of the chapter. Section 3.1 recalls the basic definitions of the language associated with a non-deterministic functor. Section 3.2 describes the decision procedure to check equivalence of regular expressions. Section 3.3 formulates the aforementioned language as an algebraic specification, which paves the way to implement in CIRC the procedure to decide equivalence of expressions. The implementation of the decision procedure and its soundness are described in Section 3.4. In Section 3.5 we show, by means of several examples, how one can check bisimilarity, using CIRC. In Section 3.6 we briefly wrap up the contributions of this chapter.

3.1 generalised regular expressions

In this section we briefly recall the basic definitions in [SBR10].

Non-deterministic functors are functors $\mathcal{G}\colon \mathbf{Set} \to \mathbf{Set}$ built inductively from the identity, and constants, using $\times$, $\oplus$, $(-)^A$ and $\mathcal{P}_\omega$:

$$NDF \ni \mathcal{G} ::= \mathsf{Id} \mid \mathsf{B} \mid \mathcal{G} \oplus \mathcal{G} \mid \mathcal{G} \times \mathcal{G} \mid \mathcal{G}^A \mid \mathcal{P}_\omega \mathcal{G} \tag{3.1}$$

where B is a finite join-semilattice and A is a finite set. Typical examples of such functors include $\mathcal{S} = \mathsf{B} \times \mathsf{Id}$, $\mathcal{M} = (\mathsf{B} \times \mathsf{Id})^A$, $\mathcal{D} = 2 \times \mathsf{Id}^A$, $\mathcal{Q} = (1 \oplus \mathsf{Id})^A$, $\mathcal{N} = 2 \times \mathcal{P}_\omega(\mathsf{Id})^A$ and $\mathcal{L} = 1 \oplus \mathcal{P}_\omega(\mathsf{Id})^A$. These functors represent, respectively, the type of streams, Mealy, deterministic, partial deterministic automata, non-deterministic automata and labelled transition systems with explicit termination. $\mathcal{S}$-bisimulation is stream equality, whereas $\mathcal{D}$-bisimulation coincides with language equivalence.
Next, we give the definition of the ingredient relation, which relates a non-deterministic functor $\mathcal{G}$ with its *ingredients*, *i.e.*, the functors used in its inductive construction. We shall use this relation later for typing our expressions.

3.1.1 Definition. Let $\lhd \subseteq NDF \times NDF$ be the least reflexive and transitive relation on non-deterministic functors such that

$$\begin{array}{lll} \mathcal{G}_1 \lhd \mathcal{G}_1 \times \mathcal{G}_2 & \mathcal{G}_2 \lhd \mathcal{G}_1 \times \mathcal{G}_2 & \mathcal{G}_1 \lhd \mathcal{G}_1 \oplus \mathcal{G}_2 \\ \mathcal{G}_2 \lhd \mathcal{G}_1 \oplus \mathcal{G}_2 & \mathcal{G} \lhd \mathcal{G}^A & \mathcal{G} \lhd \mathcal{P}_\omega \mathcal{G}. \end{array}$$ ♣

Throughout this chapter we use $\mathcal{F} \lhd \mathcal{G}$ as a shorthand for $(\mathcal{F}, \mathcal{G}) \in \lhd$. If $\mathcal{F} \lhd \mathcal{G}$, then $\mathcal{F}$ is said to be an *ingredient* of $\mathcal{G}$. For example, 2, Id, Id^A and $\mathcal{D}$ itself are all the ingredients of the deterministic automata functor $\mathcal{D}$.
A language of expressions $\mathsf{Exp}_\mathcal{G}$ is associated with each non-deterministic functor $\mathcal{G}$:

3.1.2 Definition (Expressions). Let A be a finite set, B a finite join-semilattice and X a set of fixed-point variables. The set Exp of all *(generalised regular) expressions* is given by the following grammar, where $a \in A$, $b \in \mathsf{B}$ and $x \in X$:

$$\varepsilon \quad ::= \quad x \mid \varepsilon \oplus \varepsilon \mid \gamma \tag{3.2}$$

where γ is a *guarded expression* given by: $\swarrow$ not algebra since variable binding

$$\gamma \quad ::= \quad \underline{\emptyset} \mid \gamma \oplus \gamma \mid \mu x.\gamma \mid b \mid l\langle\varepsilon\rangle \mid r\langle\varepsilon\rangle \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid \{\varepsilon\} \tag{3.3}$$

In the expression $\mu x.\gamma$, μ is a binder for all the free occurrences of x in γ. Variables that are not bound are free. A *closed expression* is an expression without free occurrences of fixed-point variables x. We denote the set of closed expressions by Exp^c.
The language of expressions for non-deterministic coalgebras is a generalisation of the classical notion of regular expressions: $\underline{\emptyset}$, $\varepsilon_1 \oplus \varepsilon_2$ and $\mu x.\gamma$ play similar roles to the regular expressions denoting empty language, the union of languages and the Kleene star. Moreover, note that, not unexpectedly, in [SBR10], $\oplus$ was axiomatised as an associative, commutative and idempotent operator, with $\underline{\emptyset}$ as a neutral element. The expressions $l\langle\varepsilon\rangle$, $r\langle\varepsilon\rangle$, $l[\varepsilon]$, $r[\varepsilon]$, $a(\varepsilon)$ and $\{\varepsilon\}$ specify the left and right-hand side of products and sums, function application and singleton sets, respectively.
Next we present a type assignment system for associating expressions to non-deterministic functors. This will allow us to associate with each functor $\mathcal{G}$ the expressions $\varepsilon \in \mathsf{Exp}^c$ that are valid specifications of $\mathcal{G}$-coalgebras.

3.1.3 Definition (Type system). We define a typing relation $\vdash \subseteq \mathsf{Exp} \times NDF \times NDF$ that will associate an expression ε with two non-deterministic functors $\mathcal{F}$ and $\mathcal{G}$, which are related by the ingredient relation ($\mathcal{F}$ is an ingredient of $\mathcal{G}$). We shall write $\vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}$ (read "ε is of type $\mathcal{F} \lhd \mathcal{G}$") for $(\varepsilon, \mathcal{F}, \mathcal{G}) \in \vdash$. The rules that define $\vdash$ are the following:

$$\frac{}{\vdash \underline{\emptyset} \colon \mathcal{F} \lhd \mathcal{G}} \qquad \frac{}{\vdash b \colon \mathsf{B} \lhd \mathcal{G}}\ (b \in \mathsf{B}) \qquad \frac{}{\vdash x \colon \mathcal{G} \lhd \mathcal{G}}\ (x \in X)$$

$$\frac{\vdash \varepsilon \colon \mathcal{G} \lhd \mathcal{G}}{\vdash \mu x.\varepsilon \colon \mathcal{G} \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon_1 \colon \mathcal{F} \lhd \mathcal{G} \quad \vdash \varepsilon_2 \colon \mathcal{F} \lhd \mathcal{G}}{\vdash \varepsilon_1 \oplus \varepsilon_2 \colon \mathcal{F} \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{G} \lhd \mathcal{G}}{\vdash \varepsilon \colon \mathsf{Id} \lhd \mathcal{G}}$$

$$\frac{\vdash \varepsilon \colon \mathcal{F}_2 \lhd \mathcal{G}}{\vdash r[\varepsilon] \colon \mathcal{F}_1 \oplus \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}}{\vdash a(\varepsilon) \colon \mathcal{F}^A \lhd \mathcal{G}}\ (a \in A) \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_1 \lhd \mathcal{G}}{\vdash l\langle \varepsilon \rangle \colon \mathcal{F}_1 \times \mathcal{F}_2 \lhd \mathcal{G}}$$

$$\frac{\vdash \varepsilon \colon \mathcal{F}_2 \lhd \mathcal{G}}{\vdash r\langle \varepsilon \rangle \colon \mathcal{F}_1 \times \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_1 \lhd \mathcal{G}}{\vdash l[\varepsilon] \colon \mathcal{F}_1 \oplus \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_1 \lhd \mathcal{G}}{\vdash \{\varepsilon\} \colon \mathcal{P}_\omega \mathcal{F}_1 \lhd \mathcal{G}}$$

♣

We can now formally define the set of $\mathcal{G}$-expressions: well-typed expressions associated with a non-deterministic functor $\mathcal{G}$.

3.1.4 Definition ($\mathcal{G}$-expressions). Let $\mathcal{G}$ be a non-deterministic functor and $\mathcal{F}$ an ingredient of $\mathcal{G}$. We define $\mathsf{Exp}_{\mathcal{F} \lhd \mathcal{G}}$ by:

$$\mathsf{Exp}_{\mathcal{F} \lhd \mathcal{G}} = \{\varepsilon \in \mathsf{Exp}^c \mid \vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}\}.$$

We define the set $\mathsf{Exp}_{\mathcal{G}}$ of well-typed *$\mathcal{G}$-expressions* by $\mathsf{Exp}_{\mathcal{G} \lhd \mathcal{G}}$. ♣

In [SBR10], it was proved that the set of $\mathcal{G}$-expressions for a given non-deterministic functor $\mathcal{G}$ has a coalgebraic structure:

$$\delta_{\mathcal{G}} \colon \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

More precisely, in [SBR10], which we refer to for the complete definition of $\delta_{\mathcal{G}}$, the authors defined a function $\delta_{\mathcal{F} \lhd \mathcal{G}} \colon \mathsf{Exp}_{\mathcal{F} \lhd \mathcal{G}} \to \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$ and then set $\delta_{\mathcal{G}} = \delta_{\mathcal{G} \lhd \mathcal{G}}$.
The coalgebraic structure on the set of expressions enabled the proof of a Kleene-like theorem:

3.1.5 Theorem (Theorems 3.12 and 3.14 in [SBR10]). *Consider $\mathcal{G}$ a non-deterministic functor.*

1. *For any $\varepsilon \in \mathsf{Exp}_{\mathcal{G}}$, there exists a finite $\mathcal{G}$-coalgebra (S, g) and $s \in S$ such that $\varepsilon \sim s$.*
2. *For every finite $\mathcal{G}$-coalgebra (S, g) and $s \in S$ there exists an expression $\varepsilon_s \in \mathsf{Exp}_{\mathcal{G}}$ such that $\varepsilon_s \sim s$.*

In order to provide the reader with intuition regarding the notions presented above, we illustrate them with an example.

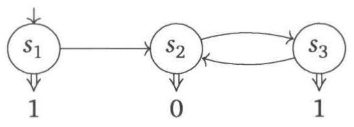
3.1.6 Example. Let us instantiate the definition of $\mathcal{G}$-expressions to the functor of streams $\mathcal{S} = \mathsf{B} \times \mathsf{Id}$ (the ingredients of this functor are B, Id and $\mathcal{S}$ itself). Let X be a set of (recursion or) fixed-point variables. The set $\mathsf{Exp}_{\mathcal{S}}$ of *stream expressions* is given by the set of closed, guarded expressions generated by the following BNF grammar. For $x \in X$:

$$\begin{aligned} \mathsf{Exp}_{\mathcal{S}} \ni \varepsilon &::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid l\langle \tau \rangle \mid r\langle \varepsilon \rangle \\ \tau &::= \underline{\emptyset} \mid b \mid \tau \oplus \tau \end{aligned} \tag{3.4}$$

Intuitively, the expression $l\langle b\rangle$ is used to specify that the head of the stream is b, while $r\langle\varepsilon\rangle$ specifies a stream whose tail behaves as specified by ε. For the two element join-semilattice $\mathsf{B} = \{0,1\}$ (with $\perp_{\mathsf{B}} = 0$) examples of well-typed expressions include $\underline{\emptyset}$, $l\langle 1\rangle \oplus r\langle l\langle\underline{\emptyset}\rangle\rangle$ and $\mu x.r\langle x\rangle \oplus l\langle 1\rangle$. The expressions $l[1]$, $l\langle 1\rangle \oplus 1$ and $\mu x.1$ are examples of non well-typed expressions for $\mathscr{S}$, because the functor $\mathscr{S}$ does not involve $\oplus$, the subexpressions in the sum have different type, and recursion is not at the outermost level (1 has type $\mathsf{B} \lhd \mathscr{S}$), respectively.
By applying the definition in [SBR10], the coalgebra structure on expressions $\delta_{\mathscr{S}}$ is given by:

$$\begin{array}{lcl}
\delta_{\mathscr{S}} \colon \mathsf{Exp}_{\mathscr{S}} \to \mathsf{B} \times \mathsf{Exp}_{\mathscr{S}} & & \\
\delta_{\mathscr{S}}(\underline{\emptyset}) & = & \langle \perp_{\mathsf{B}}, \underline{\emptyset}\rangle \\
\delta_{\mathscr{S}}(\varepsilon_1 \oplus \varepsilon_2) & = & \langle b_1 \vee b_2, \varepsilon_1' \oplus \varepsilon_2'\rangle \text{ where } \langle b_i, \varepsilon_i'\rangle = \delta_{\mathscr{S}}(\varepsilon_i),\ i \in \overline{1,2} \\
\delta_{\mathscr{S}}(\mu x.\varepsilon) & = & \delta_{\mathscr{S}}(\varepsilon[\mu x.\varepsilon/x]) \\
\delta_{\mathscr{S}}(l\langle\tau\rangle) & = & \langle \delta_{\mathsf{B}\lhd\mathscr{S}}(\tau), \underline{\emptyset}\rangle \\
\delta_{\mathscr{S}}(r\langle\varepsilon\rangle) & = & \langle \perp_{\mathsf{B}}, \varepsilon\rangle \\
\delta_{\mathsf{B}\lhd\mathscr{S}}(\underline{\emptyset}) & = & \perp_B \\
\delta_{\mathsf{B}\lhd\mathscr{S}}(b) & = & b \\
\delta_{\mathsf{B}\lhd\mathscr{S}}(\tau \oplus \tau') & = & \delta_{\mathsf{B}\lhd\mathscr{S}}(\tau) \vee \delta_{\mathsf{B}\lhd\mathscr{S}}(\tau')
\end{array}$$

The proof of Kleene's theorem provides algorithms to go from expressions to streams and vice-versa. We illustrate it by means of examples.
Consider the following stream:



We draw the stream with an automata-like flavor. The transitions indicate the tail of the stream represented by a state and the output value the head. In a more traditional notation, the above automata represents the infinite stream $(1,0,1,0,1,0,1,\ldots)$.
To compute expressions ε_1, ε_2 and ε_3 equivalent to s_1, s_2 and s_3 we associate with each state s_i a variable x_i and get the equations:

$$\varepsilon_1 = \mu x_1.l\langle 1\rangle \oplus r\langle x_2\rangle \quad \varepsilon_2 = \mu x_2.l\langle 0\rangle \oplus r\langle x_3\rangle \quad \varepsilon_3 = \mu x_3.l\langle 1\rangle \oplus r\langle x_2\rangle$$

As our goal is to remove all the occurrences of free variables in our expressions, we proceed as follows. First we substitute x_2 by ε_2 in ε_1, and x_3 by ε_3 in ε_2, and obtain the following expressions:

$$\varepsilon_1 = \mu x_1.l\langle 1\rangle \oplus r\langle \varepsilon_2\rangle \quad \varepsilon_2 = \mu x_2.l\langle 0\rangle \oplus r\langle \varepsilon_3\rangle$$

Note that at this point ε_1 and ε_2 already denote closed expressions. Therefore, as a last step, we replace x_2 in ε_3 by ε_2 and get the following closed expressions:

$$\varepsilon_1 = \mu x_1.l\langle 1\rangle \oplus r\langle \varepsilon_2\rangle \quad \varepsilon_2 = \mu x_2.l\langle 0\rangle \oplus r\langle \varepsilon_3\rangle \quad \varepsilon_3 = \mu x_3.l\langle 1\rangle \oplus r\langle \mu x_2.l\langle 0\rangle \oplus r\langle x_3\rangle\rangle$$

satisfying, by construction, $\varepsilon_1 \sim s_1$, $\varepsilon_2 \sim s_2$ and $\varepsilon_3 \sim s_3$.
For the converse construction, consider the expression $\varepsilon = (\mu x.r\langle x\rangle) \oplus l\langle 1\rangle$. We construct an automaton by repeatedly applying the coalgebra structure on expressions $\delta_{\mathscr{S}}$, modulo associativity, commutativity and idempotency (ACI) of $\oplus$ in order to guarantee finiteness.

First, note that $\delta_{\mathcal{S}}(\mu x.r\langle x\rangle) = \delta_{\mathcal{S}}(r\langle \mu x.r\langle x\rangle\rangle) = \langle \bot_{\mathsf{B}}, \mu x.r\langle x\rangle\rangle$. Applying the definition of $\delta_{\mathcal{S}}$ above, we have:

$$\delta_{\mathcal{S}}(\varepsilon) = \langle 1, (\mu x.r\langle x\rangle)\oplus\underline{\emptyset}\rangle \text{ and } \delta_{\mathcal{S}}((\mu x.r\langle x\rangle)\oplus\underline{\emptyset}) = \langle 0, (\mu x.r\langle x\rangle)\oplus\underline{\emptyset}\rangle$$

which leads to the following stream (automaton):

$$\begin{tikzcd}
\varepsilon \arrow[r] \arrow[d, Rightarrow] & (\mu x.r\langle x\rangle)\oplus\underline{\emptyset} \arrow[loop right] \arrow[d, Rightarrow] \\
1 & 0
\end{tikzcd}$$

At this point, we want to remark that the direct application of $\delta_{\mathcal{S}}$, without ACI, might generate infinite automata. Take, for instance, the expression $\varepsilon = \mu x.r\langle x \oplus x\rangle$. Note that $\delta_{\mathcal{S}}(\mu x.r\langle x \oplus x\rangle) = \langle 0, \varepsilon \oplus \varepsilon\rangle$, $\delta_{\mathcal{S}}(\varepsilon \oplus \varepsilon) = \langle 0, (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon)\rangle$, and so on. This would generate the infinite automaton

$$\begin{tikzcd}
\varepsilon \arrow[r] \arrow[d, Rightarrow] & \varepsilon\oplus\varepsilon \arrow[r] \arrow[d, Rightarrow] & (\varepsilon\oplus\varepsilon)\oplus(\varepsilon\oplus\varepsilon) \arrow[r] \arrow[d, Rightarrow] & \dots \\
0 & 0 & 0 & \dots
\end{tikzcd}$$

instead of the intended, simple and very finite, automaton

$$\begin{tikzcd}
\varepsilon \arrow[loop right] \arrow[d, Rightarrow] \\
0
\end{tikzcd}$$

In order to guarantee finiteness, one needs to identify the expressions modulo ACI, as we will discuss further in this chapter. Moreover, the axiom $\varepsilon \oplus \underline{\emptyset} \equiv \varepsilon$ could also be used in order to obtain smaller automata, but it is not crucial for termination.
Streams will be often used as a basic example to illustrate the definitions. It should be remarked that the framework is general enough to include more complex examples, such as deterministic automata, automata on guarded strings, Mealy machines and labelled transition systems. The latter two will be used as examples in Section 3.5.

3.2 Deciding equivalence of expressions

In this section, we briefly describe the decision procedure to determine whether two generalised regular expressions are equivalent or not.
The key observation is that point 1. of Theorem 3.1.5 above guarantees that each expression in the language for a given system can always be associated with a *finite* coalgebra. Given two expressions ε_1 and ε_2 in the language $\mathsf{Exp}_{\mathcal{G}}$ of a given functor $\mathcal{G}$ we can decide whether they are equivalent by constructing a *finite* bisimulation between them. This is because the finite coalgebra generated from an expression contains precisely all states that one needs to construct the equivalence relation. Even though this might seem like a trivial observation, it has very concrete consequences: for (all well-typed) generalised regular expressions we can always either determine that they are bisimilar, and exhibit a proof in the form of a bisimulation, or conclude that they are not bisimilar and pinpoint the difference by showing why the bisimulation construction failed. Hence, we have a decision procedure for equivalence of generalised regular expressions.

We will give the reader a brief example on how the equivalence check works. Further examples, for different types of systems, including examples of non-equivalence, will appear in Section 3.5.
We will show that the stream expressions

$$\varepsilon_1 = \mu x.r\langle x\rangle \oplus l\langle 0\rangle$$

and

$$\varepsilon_2 = r\langle \mu x.r\langle x\rangle \oplus l\langle 0\rangle\rangle \oplus l\langle 0\rangle$$

are equivalent. In order to do that, we have to build a bisimulation relation R on expressions for the stream functor $\mathscr{S}$, defined above, such that $(\varepsilon_1, \varepsilon_2) \in R$. We do this in the following way: we start by taking $R = \{(\varepsilon_1, \varepsilon_2)\}$ and we check whether this is already a bisimulation, by applying $\delta_{\mathscr{S}}$ to each of the expressions and checking whether the expressions have the same output value and, moreover, that no new pairs of expressions (modulo associativity, commutativity and idempotency, for more details see page 38) appear when taking transitions. Note that, for simplicity, we also use the sound axiom $\varepsilon \oplus \underline{\emptyset} \equiv \varepsilon$. If new pairs of expressions appear we add them to R and repeat the process. Intuitively, for this particular example, the transition structure can be depicted as in Figure 3.1.

$$\begin{tikzcd}
\varepsilon_1 \arrow[r, dotted, no head, "R"] \arrow[d] & \varepsilon_2 \arrow[d] & R = \{(\varepsilon_1, \varepsilon_2)\} \\
\varepsilon_1 \oplus \underline{\emptyset} = \varepsilon_1 \arrow[r, dotted, no head, "\text{not yet in } R\text{; add it}"] \arrow[d] & \varepsilon_1 = \varepsilon_1 \oplus \underline{\emptyset} \arrow[d] & R = \{(\varepsilon_1, \varepsilon_2), (\varepsilon_1, \varepsilon_1)\} \\
\varepsilon_1 \arrow[r, dotted, no head, "R"] & \varepsilon_1 & \checkmark
\end{tikzcd}$$

Figure 3.1: Bisimulation construction

In Figure 3.1, we omit the output values of the expressions, which are all 0, and use the notation $\varepsilon_1 \overset{R}{\cdots\cdots} \varepsilon_2$ to denote $(\varepsilon_1, \varepsilon_2) \in R$. Note that $R = \{(\varepsilon_1, \varepsilon_2), (\varepsilon_2, \varepsilon_2)\}$ is closed under transitions and is therefore a bisimulation. Hence, ε_1 and ε_2 are bisimilar and specify the same infinite stream (concretely, the stream with only zeros).

3.3 An algebraic view on the coalgebra of expressions

Recall that our goal is to reason about equality of generalised regular expressions in a fully automated manner. Obtaining this equality can be achieved in two distinct ways: either algebraically, reasoning with the axioms, or coalgebraically, by constructing a bisimulation relation. The latter, because of its algorithmic nature, is particularly suited for automation. Automatic constructions of bisimulations have been widely explored in CIRC and we will use this tool to implement our algorithm. This section contains material that enables us to soundly use CIRC. We want to stress however that the main result of this chapter is the description of a *decision procedure* to determine whether two expressions are equivalent or not. This procedure in turn could be implemented in any other suitable

tool or even as a standalone application. Choosing CIRC was natural for us, given the pre-existent work on bisimulation constructions.
In short, CIRC is a behavioural extension of Maude [CDE+07] enabling the coinductive definition of infinite data structures by means of the so-called "derivatives" ($\delta_{\mathcal{G}}$ for the case of generalised regular expressions). The prover allows the (automated) reasoning on properties of such structures by coinduction (or bisimulation construction). The coinductive definitions are fed to CIRC in the shape of algebraic specifications which are closely related to the original mathematical representations. Once a proof obligation is set, CIRC starts the proving mechanism which repeatedly applies the derivatives, and (potentially) stops when a bisimulation containing the initial obligation is reached. For more insight on CIRC we refer to [GLR00, RL09] and Section 3.4.
In Section 3.4, we show that the process of generating the $\mathcal{G}$-coalgebras associated with expressions by repeatedly applying $\delta_{\mathcal{G}}$ and normalising the expressions obtained at each step is closely related to the proving mechanism already existent in CIRC.
In Section 3.1, we have introduced a (theoretical) framework which, given a functor G, allows for the uniform derivation of 1) a language $\mathsf{Exp}_{\mathcal{G}}$ for specifying behaviours of $\mathcal{G}$-systems, and 2) a coalgebraic structure on $\mathsf{Exp}_{\mathcal{G}}$, which provides an operational semantics to the set of expressions. In this context, given that CIRC is based on algebraic specifications, we need two things in order to reach our final goal:

- extend and adapt the framework of Section 3.1 in order to enable the implementation of a tool which allows the automatic derivation of *algebraic specifications* that model 1) and 2) above, to deliver to CIRC;
- provide a decision procedure, implemented in CIRC based on an *equational entailment relation*, in order to check bisimilarity of expressions.

In the rest of this chapter we will present the algebraic setting for reasoning on bisimilarity of generalised regular expressions. A brief overview on the parallel between the coalgebraic concepts in [SBR10] and their algebraic correspondents introduced in this section is provided later, in Figure 3.2.
An *algebraic specification* is a triple $\mathcal{E} = (S, \Sigma, E)$, where S is a set of *sorts*, Σ is a S*-sorted signature* and E is a set of *conditional equations* of the form $(\forall X)\, t = t'$ *if* $(\bigwedge_{i\in I} u_i = v_i)$, where t, t', u_i, and v_i ($i \in I$ – a set of indices for the conditions) are Σ-terms with variables in X. We say that the *sort of the equation* is s whenever $t, t' \in \mathcal{T}_{\Sigma,s}(X)$. Here, $\mathcal{T}_{\Sigma,s}(X)$ denotes the set of terms of sort s of the Σ-algebra freely generated by X. If $I = \emptyset$ then the equation is *unconditional* and may be written as $(\forall X)\, t = t'$.
Let $\vdash$ be the *equational entailment (deduction) relation* defined as in [GM92]. We write $\mathcal{E} \vdash e$ whenever equation e is deducible from the equations E in $\mathcal{E}$ by reflexivity, symmetry, transitivity, congruence or (conditional) substitutivity (*i.e.*, whenever $E \vdash e$).

The algebraic specification of generalised regular expressions is built on top of definitions based on grammars in Backus-Naur form (BNF), such as (3.1) and (3.2). Next we introduce the general technique for transforming BNF notations into algebraic specifications.

The general rule used for translating definitions based on BNF grammars into algebraic specifications is as follows: each syntactical category and vocabulary is considered as a sort and each production is considered as a constructor operation or a subsort relation.
For instance, according to the grammar (3.1) of non-deterministic functors, we have a sort SltName – representing the vocabulary of join-semilattices B, a sort AlphName – for

the vocabulary of the alphabets A, a sort Functor – associated with the syntactical category of the non-deterministic functors $\mathcal{G}$, a subsort relation SltName<Functor representing the production $\mathcal{G} ::= \mathsf{B}$, and constructor operations for the other productions.
Generally, each production $A ::= rhs$ gives rise to a constructor $(rhs) \rightarrow (A)$, the direction of the arrow being reversed. For instance, for grammar (3.1), the production $\mathcal{G} ::= \mathsf{Id}$ is represented by a constant (nullary operation) $\mathsf{Id}: \rightarrow \mathsf{Functor}$, and the sum construction by the binary operation

$$\_ \oplus \_ : \mathsf{Functor\ Functor} \rightarrow \mathsf{Functor}.$$

Remark 2 *Note that the above mechanism for translating BNF grammars into algebraic specifications makes use of subsort relations for representing productions such as* $\mathcal{G} ::= \mathsf{B}$. *This is because CIRC works with order-sorted algebras, and we want to keep the algebraic specifications of non-deterministic functors as close as possible to their implementation in CIRC.*

The algebraic specifications of coalgebras of generalised regular expressions are defined in a modular fashion, based on the specifications of:

- non-deterministic functors ($\mathcal{G}$);
- generalised regular expressions ($\varepsilon \in \mathsf{Exp}_{\mathcal{G}}$);
- "transition" functions ($\delta_{\mathcal{G}}$);
- "structured" expressions ($\sigma \in \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$, for all $\mathcal{F}$ ingredients of $\mathcal{G}$).

Moreover, recall that for a non-deterministic functor $\mathcal{G}$, bisimilarity of $\mathcal{G}$-expressions is decided based on the relation lifting $\overline{\mathcal{G}}$ over "structured" expressions in $\mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$ (see (2.2) in Section 2.2). Therefore, the deduction relation $\vdash$ has to be extended to allow a restricted contextual reasoning over "structured" expressions in $\mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$, for all ingredients $\mathcal{F}$ of $\mathcal{G}$.
The aforementioned algebraic specifications and the extension of $\vdash$ are modelled as follows.

The algebraic specification of a non-deterministic functor $\mathcal{G}$ includes:

- the translation of the BNF grammar (3.1), as presented above;
- the specification of the functor ingredients, given by a sort Ingredient and a constructor $\_ \triangleleft \_ : \mathsf{Functor\ Functor} \rightarrow \mathsf{Ingredient}$ (according to Definition 3.1.1);
- the specification of each alphabet $A = \{a_1, \ldots, a_n\}$ occurring in the definition of $\mathcal{G}$: this consists of a subsort A<Alph, a constant $a_i: \rightarrow A$ for $i \in \overline{1,n}$, and a distinguished constant A of sort AlphName used to refer the alphabet in the definition of the functor;
- the specification of each semilattice $\mathsf{B} = (\{b_1, \ldots, b_n\}, \vee, \perp_B)$ occurring in the definition of $\mathcal{G}$: this consists of a subsort $\mathsf{B} < \mathsf{Slt}$, a constant $b_i: \rightarrow \mathsf{B}$ for $i \in \overline{1,n}$, a distinguished constant B of sort SltName used to refer the corresponding semilattice in the definition of the functor, and the equations defining $\vee$ and $\perp_B$ (this should be one of the b_i's);
- an equation defining $\mathcal{G}$ (as a functor expression).

The algebraic specification of generalised regular expressions consists of:

- (according to the BNF grammar in Definition 3.1.2) a sort Exp representing expressions ε, FixpVar the sort for the vocabulary of the fixed-point variables, and Slt the sort for the elements of semilattices. Moreover, we consider constructor operations for all the productions. For example, the production $\varepsilon ::= \varepsilon \oplus \varepsilon$ is represented by an operation $\_ \oplus \_$: Exp Exp $\to$ Exp, and $\varepsilon ::= \mu x.\gamma$ is represented by $\mu\_.\_$: FixpVar Exp $\to$ Exp. (We chose not to provide any restriction to guarantee that γ is a guarded expression, at this stage in the definition of $\mu\_.\_$. However, guards can be easily checked by pattern matching, according to the grammars in Definition 3.1.2);
- the specification of the substitution of a fixed-point variable with an expression, given by an operation **_[_/_]**: Exp Exp FixpVar $\to$ Exp and a set of equations, one for each constructor. For example, the equations associated with $\underline{\emptyset}$ and $\oplus$ are: $\underline{\emptyset}[\varepsilon/x] = \underline{\emptyset}$, and respectively, $(\varepsilon_1 \oplus \varepsilon_2)[\varepsilon/x] = (\varepsilon_1[\varepsilon/x]) \oplus (\varepsilon_2[\varepsilon/x])$, where $\varepsilon, \varepsilon_1, \varepsilon_2$ are $\mathscr{G}$-expressions and x is a fixed-point variable;
- the specification of the type-checking relation in Definition 3.1.3, given by an operation $\_ : \_$: Exp Ingredient $\to$ Bool and an equation for each inference rule defining this relation. For example the rule

$$\frac{\vdash \varepsilon_1 : \mathscr{F} \lhd \mathscr{G} \quad \vdash \varepsilon_2 : \mathscr{F} \lhd \mathscr{G}}{\vdash \varepsilon_1 \oplus \varepsilon_2 : \mathscr{F} \lhd \mathscr{G}}$$

  is represented by the equation $\varepsilon_1 \oplus \varepsilon_2 : \mathscr{F} \lhd \mathscr{G} = \varepsilon_1 : \mathscr{F} \lhd \mathscr{G} \wedge \varepsilon_2 : \mathscr{F} \lhd \mathscr{G}$. The type-checking operator is used in order to verify whether the expressions checked for equivalence are well-typed (Definition 3.1.4). Moreover, note that for the consistency of notation, algebraically we write $\varepsilon : \mathscr{F} \lhd \mathscr{G}$ to represent expressions ε of type $\mathscr{F} \lhd \mathscr{G}$.

The algebraic specification of $\delta_{\mathscr{G}}$ consists of:

- the specification of the coalgebra of $\mathscr{G}$-expressions $\delta_{\mathscr{G}}$ given by three operations

  $\delta\_(\_)$: Ingredient Exp $\to$ ExpStruct
  $Empty$: Ingredient $\to$ ExpStruct
  $Plus\_(\_,\_)$: Ingredient ExpStruct ExpStruct $\to$ ExpStruct;

- equations describing the definitions of these operations as in [SBR10].

As mentioned above, the set of $\mathscr{G}$-expressions is provided with a coalgebraic structure given by the function $\delta_{\mathscr{G}}$: $\mathsf{Exp}_{\mathscr{G}} \to \mathscr{G}(\mathsf{Exp}_{\mathscr{G}})$, where $\mathscr{G}(\mathsf{Exp}_{\mathscr{G}})$ can be understood as the set of expressions with structure given by $\mathscr{G}$ (and its ingredients). The set of structured expressions is defined by the following grammar:

$$\sigma ::= \varepsilon \mid b \mid \langle \sigma, \sigma \rangle \mid k_1(\sigma) \mid k_2(\sigma) \mid \bot \mid \top \mid \lambda.(a, \mathscr{F} \lhd \mathscr{G}, \sigma) \mid \{\sigma\} \tag{3.5}$$

where $\varepsilon \in \mathsf{Exp}_{\mathscr{G}}$ and $b \in \mathsf{B}$. The typing rules below give precise meaning to these expressions. Note that $\bot, \top$ are two expressions coming from $\mathscr{G} = \mathscr{G}_1 \oplus \mathscr{G}_2$, used to denote underspecification and overspecification, respectively.
The associated algebraic specification includes:

- a sort ExpStruct representing expressions σ (from $\mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$, with $\mathscr{F} \lhd \mathscr{G}$), and one operation for each production in the BNF grammar (3.5). Note that the construction $\lambda.(a, \mathscr{F} \lhd \mathscr{G}, \sigma)$ has as coalgebraic correspondent a function $f \in \mathscr{F}^A(\mathsf{Exp}_{\mathscr{G}})$, and is defined by cases as follows:

$$\lambda.(a, \mathscr{F} \lhd \mathscr{G}, \sigma)(a') = \text{ if } (a = a') \text{ then } \sigma \text{ else } \mathit{Empty}_{\mathscr{F} \lhd \mathscr{G}};$$

- the extension of the type-checking relation to structured expressions, defined by:

$$\frac{\vdash b\colon \mathsf{B} \lhd \mathscr{G}}{\vdash b \in \mathsf{B}(\mathsf{Exp}\,\mathscr{G})} \qquad \frac{\vdash \varepsilon\colon \mathsf{Id} \lhd \mathscr{G}}{\vdash \varepsilon \in \mathsf{Id}(\mathsf{Exp}\,\mathscr{G})}$$

$$\frac{}{\vdash \bot \in \mathscr{F}_1 \oplus \mathscr{F}_2(\mathsf{Exp}\,\mathscr{G})} \qquad \frac{}{\vdash \top \in \mathscr{F}_1 \oplus \mathscr{F}_2(\mathsf{Exp}\,\mathscr{G})}$$

$$\frac{\vdash \sigma \in \mathscr{F}_i(\mathsf{Exp}\,\mathscr{G})}{\vdash k_i(\sigma) \in \mathscr{F}_1 \oplus \mathscr{F}_2(\mathsf{Exp}\,\mathscr{G})}\ i \in \overline{1,2} \qquad \frac{\vdash \sigma_1 \in \mathscr{F}_i(\mathsf{Exp}\,\mathscr{G}) \quad \vdash \sigma_2 \in \mathscr{F}_i(\mathsf{Exp}\,\mathscr{G})}{\vdash \langle \sigma_1, \sigma_2 \rangle \in \mathscr{F}_1 \times \mathscr{F}_2(\mathsf{Exp}\,\mathscr{G})}$$

$$\frac{\vdash \sigma \in \mathscr{F}(\mathsf{Exp}\,\mathscr{G}),\ a \in A}{\vdash \lambda.(a, \mathscr{F} \lhd \mathscr{G}, \sigma) \in \mathscr{F}^A(\mathsf{Exp}\,\mathscr{G})} \qquad \frac{\vdash \sigma \in \mathscr{F}(\mathsf{Exp}\,\mathscr{G})}{\vdash \{\sigma\} \in \mathscr{P}_\omega\mathscr{F}(\mathsf{Exp}\,\mathscr{G})}$$

and specified by an operation $\_ \in \_(\mathsf{Exp}\,\_)$: ExpStruct Functor Functor $\to$ Bool (where we used a mix-fix notation) and an equation for each of the above inference rules. For example, the first rule has associated the equation $b \in \mathsf{B}(\mathsf{Exp}\,\mathscr{G}) = b\colon \mathsf{B} \lhd \mathscr{G}$. For consistency of notation, we write $\sigma \in \mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$ to denote that σ is an element of $\mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$.

Remark 3 *In terms of membership equational logic (MEL) [BJM00], both $\mathscr{F} \lhd \mathscr{G}$ and $\mathscr{F}(\mathsf{Exp}\,\mathscr{G})$ can be thought of as being sorts and, for example, $\varepsilon\colon \mathscr{F} \lhd \mathscr{G}$ as a membership assertion. Even if MEL is an elegant theory, we prefer not to use it here because this implies the dynamic declaration of sorts and a set of assertions for such a sort. The above approach is generic and therefore more flexible.*

As previously hinted at the beginning of this section, in order to algebraically reason on bisimilarity of $\mathscr{G}$-expressions in CIRC, one has to extend the deduction relation $\vdash$ to allow a restricted contextual reasoning on expressions in $\mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$, for all ingredients $\mathscr{F}$ of a non-deterministic functor $\mathscr{G}$. We call the extended entailment $\vdash_{NDF}$.
The aforementioned restriction refers to inhibiting the use of congruence during equational reasoning, in order to guarantee the soundness of CIRC proofs. This is realised by means of a *freezing operator,* which intuitively behaves as a wrapper on the expressions checked for equivalence, by changing their sort to a fresh sort Frozen. This way, the hypotheses collected during a CIRC proof session cannot be used freely in contextual reasoning, hence preventing the derivation of untrue equations (as illustrated in Example 3.4.1).
We further show how the freezing mechanism is implemented in our algebraic setting, and define $\vdash_{NDF}$.
Let $\mathscr{E}$ be an algebraic specification. We extend $\mathscr{E}$ by adding the freezing operation $\boxed{-}\colon s \to$ Frozen for each sort $s \in \Sigma$, where Frozen is a fresh sort. By $\boxed{t}$ we represent the *frozen* form of a Σ-term t, and by $\boxed{e}$ a *frozen equation* of the shape $(\forall X)\boxed{t} = \boxed{t'}$ if c. Note that, according to [RL09], conditions c need not to be frozen, as their (so-called visible)

sort does not allow their collection into the set of CIRC hypotheses. The entailment relation $\vdash$ is defined over frozen equations following [RL09]; more details are provided in Section 3.4.
Recall that a relation $R \subseteq \mathsf{Exp}_{\mathcal{G}} \times \mathsf{Exp}_{\mathcal{G}}$ is a bisimulation if and only if $(s,t) \in R \Rightarrow (\delta_{\mathcal{G} \lhd \mathcal{G}}(s), \delta_{\mathcal{G} \lhd \mathcal{G}}(t)) \in \overline{\mathcal{G}}(R)$. Here, $\overline{\mathcal{G}}(R) \subseteq \mathcal{G}(\mathsf{Exp}_{\mathcal{G}}) \times \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$ is the lifting of the relation $R \subseteq \mathsf{Exp}_{\mathcal{G}} \times \mathsf{Exp}_{\mathcal{G}}$, defined as

$$\overline{\mathcal{G}}(R) = \{(\mathcal{G}(\pi_1)(x), \mathcal{G}(\pi_2)(x)) \mid x \in \mathcal{G}(R)\} .$$

So, intuitively, reasoning on bisimilarity of two expressions $(\varepsilon, \varepsilon')$ in R reduces to checking whether the application of $\delta_{\mathcal{G}}$ maps them into $\overline{\mathcal{G}}(R)$.
Therefore, checking whether a pair (s^δ, t^δ) is in $\overline{\mathcal{G}}(R)$ consists in checking, for example for the case of $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$, whether $(s_1^\delta, t_1^\delta) \in \overline{\mathcal{G}_1}(R)$ and $(s_2^\delta, t_2^\delta) \in \overline{\mathcal{G}_2}(R)$, where $s^\delta = \langle s_1^\delta, s_2^\delta \rangle$ and $t^\delta = \langle t_1^\delta, t_2^\delta \rangle$. In an algebraic setting, this would reduce to building an algebraic specification $\mathcal{E}$ and defining an entailment relation $\vdash_{NDF}$ such that one can infer $\mathcal{E} \vdash_{NDF} \boxed{\langle s_1^\delta, s_2^\delta \rangle} = \boxed{\langle t_1^\delta, t_2^\delta \rangle}$ (this is the algebraic correspondent we consider for $(\langle s_1^\delta, s_2^\delta \rangle, \langle t_1^\delta, t_2^\delta \rangle) \in \overline{\mathcal{G}}(R)$) by showing $\mathcal{E} \vdash_{NDF} \boxed{s_1^\delta} = \boxed{t_1^\delta}$ (or $(s_1^\delta, t_1^\delta) \in \overline{\mathcal{G}_1}(R)$) and $\mathcal{E} \vdash_{NDF} \boxed{s_2^\delta} = \boxed{t_2^\delta}$ (or $(s_2^\delta, t_2^\delta) \in \overline{\mathcal{G}_2}(R)$). We hint that the aforementioned algebraic specification $\mathcal{E}$ consists of $\mathcal{E}_{\mathcal{G}}$ and a set of frozen equations (see Corollary 3.3.4).
The entailment relation $\vdash_{NDF}$ for reasoning on bisimilarity of $\mathcal{G}$-expressions is based on the definition of $\overline{\mathcal{G}}$.

3.3.1 Definition. The entailment relation $\vdash_{NDF}$ is the extension of $\vdash$ with the following inference rules, which allow a restricted contextual reasoning over the frozen equations of structured expressions:

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\sigma_1} = \boxed{\sigma_1'} \quad \mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\sigma_2} = \boxed{\sigma_2'}}{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\langle \sigma_1, \sigma_2 \rangle} = \boxed{\langle \sigma_1', \sigma_2' \rangle}} \tag{3.6}$$

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\sigma} = \boxed{\sigma'}}{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{k_i(\sigma)} = \boxed{k_i(\sigma')}} \; i \in \overline{1,2} \tag{3.7}$$

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{f(a)} = \boxed{g(a)}, \; \textit{for all } a \in A}{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{f} = \boxed{g}} \tag{3.8}$$

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\sigma_{i_1}} = \boxed{\sigma_{j_1}'}, \ldots, \mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\sigma_{i_k}} = \boxed{\sigma_{j_k}'}}{\mathcal{E}_{\mathcal{G}} \vdash_{NDF} \boxed{\{\sigma_1, \ldots, \sigma_n\}} = \boxed{\{\sigma_1', \ldots, \sigma_m'\}}} \; \begin{array}{l} \{i_1, \ldots, i_k\} = \{1, \ldots, n\} \\ \{j_1, \ldots, j_k\} = \{1, \ldots, m\} \end{array} \tag{3.9}$$

Remark 4 *Note that the extension of the entailment relation $\vdash$ to $\vdash_{\mathrm{NDF}}$ implies that $\mathcal{E}_{\mathcal{G}} \vdash e$ iff $\mathcal{E}_{\mathcal{G}} \vdash_{\mathrm{NDF}} e$ holds, for any equation e of shape $\boxed{\varepsilon_1} = \boxed{\varepsilon_2}$ or $\varepsilon_1 = \varepsilon_2$, with $\varepsilon_1, \varepsilon_2$ non-structured expressions. Below, we will use the notation $\mathcal{E}_{\mathcal{G}} \vdash_{\mathrm{NDF}} E$, where E is a set of possibly frozen equations, to denote $\forall_{e \in E} \cdot \mathcal{E}_{\mathcal{G}} \vdash_{NDF} e$.*

It is interesting to recall the relation lifting for the powerset functor which is encoded in the last rule of Definition 3.3.1. A pair (U, V) is in $\overline{\mathcal{P}_\omega \mathcal{G}}(R)$ if and only if for every $u \in U$ there exists a $v \in V$ such that (u, v) belongs to $\overline{\mathcal{G}}(R)$ and, conversely, for every $v \in V$, there exists a $u \in U$ such that (u, v) belongs to $\overline{\mathcal{G}}(R)$.

Remark 5 *As already hinted (and proved in Corollary 3.3.4), reasoning on bisimilarity of expressions in a binary relation $R \subseteq \mathsf{Exp}_{\mathscr{G}} \times \mathsf{Exp}_{\mathscr{G}}$ reduces to showing that $\boxed{\delta_{\mathscr{G}}(s)} = \boxed{\delta_{\mathscr{G}}(t)}$ is a $\vdash_{\mathrm{NDF}}$-consequence, for all $(s,t) \in R$. The equational proof is performed in a "top-down" fashion, by reasoning on the subsequent equalities between the components of the corresponding structured expression $\delta_{\mathscr{G}}(s)$, $\delta_{\mathscr{G}}(t)$ in an inductive manner. This is realised by applying the inverted rules (3.6)–(3.9).*
Moreover, note that rule (3.9) is not invertible in the usual sense; rather any statement matching the form of the conclusion can only be proved by some instance of the rule.

We will further formalise the connection between the inductive definition of $\overline{\mathscr{G}}$ (on the coalgebraic side) and $\vdash_{NDF}$ (on the algebraic side) in Theorem 3.3.2, hence enabling the definition of bisimulations in algebraic terms, in Corollary 3.3.4.

Remark 6 *Equations in $\mathscr{E}_{\mathscr{G}}$ (built as previously described in this section) are used in the equational reasoning only for reducing terms of shape* $\mathsf{op}(t_1, \dots, t_n)$ *according to the definition of the operation* op. *For the simplicity of the proofs of Theorem 3.3.2 and Corollary 3.3.4, whenever we write* $\mathsf{op}(t_1, \dots, t_n)$, *we refer to the associated term reduced according to the definition of* op.

First we introduce some notational conventions. Let $\mathscr{G}$ be a non-deterministic functor and $R \subseteq \mathsf{Exp}_{\mathscr{G}} \times \mathsf{Exp}_{\mathscr{G}}$. We write:

- R_{id} to denote the set $R \cup \{(\varepsilon, \varepsilon) \mid \mathscr{E}_{\mathscr{G}} \vdash \varepsilon \colon \mathscr{G} \triangleleft \mathscr{G} = true\}$;
- $cl(R)$ for the closure of R under transitivity, symmetry and reflexivity;
- $\boxed{R}$ to represent the set $\bigcup_{e \in R}\{\boxed{e}\}$; (application of the freezing operator to all elements of R)
- $\delta_{\mathscr{G} \triangleleft \mathscr{G}}(\varepsilon = \varepsilon')$ to represent the equation $\delta_{\mathscr{G} \triangleleft \mathscr{G}}(\varepsilon) = \delta_{\mathscr{G} \triangleleft \mathscr{G}}(\varepsilon')$;
- $\mathscr{E}_{\mathscr{G}} \cup \boxed{R}$ as a shorthand for $(S, \Sigma, E \cup \{\boxed{\varepsilon} = \boxed{\varepsilon'} \mid (\varepsilon, \varepsilon') \in R\})$, where $\mathscr{E}_{\mathscr{G}} = (S, \Sigma, E)$;
- $(\sigma, \sigma') \in \overline{\mathscr{G}}(R)$ as a shorthand for: (σ, σ') is among the enumerated elements of a set S explicitly constructed as an enumeration of the finite set $\overline{\mathscr{G}}(R)$ (in the algebraic setting, $\overline{\mathscr{G}}(R)$ is a subset of $\mathscr{T}_{\Sigma,\mathsf{ExpStruct}} \times \mathscr{T}_{\Sigma,\mathsf{ExpStruct}}$ and $\mathscr{E}_{\mathscr{G}} \vdash \overline{\mathscr{G}}(R) = S$).

3.3.2 Theorem. *Consider a non-deterministic functor $\mathscr{G}$. Let $\mathscr{F}$ be an ingredient of $\mathscr{G}$, R a binary relation on the set of $\mathscr{G}$-expressions, and $\sigma, \sigma' \in \mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$.*

a) If $\mathscr{G}$ is not a constant functor, then $(\sigma, \sigma') \in \overline{\mathscr{F}}(\mathrm{cl}(R_{id}))$ iff $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{\mathrm{NDF}} \boxed{\sigma} = \boxed{\sigma'}$;

b) If $\mathscr{G}$ is a constant functor B, *then $(\sigma, \sigma') \in \overline{\mathsf{B}}(\mathrm{cl}(R_{id}))$ iff $\mathscr{E}_{\mathscr{G}} \vdash_{\mathrm{NDF}} \boxed{\sigma} = \boxed{\sigma'}$.*

In order to prove Theorem 3.3.2.a) we introduce the following lemma:

3.3.3 Lemma. *Consider $\mathscr{G}$ a non-deterministic functor and R a binary relation on the set of $\mathscr{G}$-expressions. If $(\varepsilon, \varepsilon') \in \mathrm{cl}(R_{id})$ then $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{\mathrm{NDF}} \boxed{\varepsilon} = \boxed{\varepsilon'}$.*

Proof. The proof is trivial, as equality is reflexive, symmetric and transitive.

We are now ready to prove Theorem 3.3.2.

Proof (Theorem 3.3.2).

– Proof of Theorem 3.3.2.*a*).

- "$\Rightarrow$". The proof is by induction on the structure of $\mathscr{F}$.
  Base case:
  - $\mathscr{F} = \mathsf{B}$. It follows that (σ, σ') is of shape (b, b) where $b \in \mathsf{B}$, therefore $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{b} = \boxed{b}$ holds by reflexivity.
  - $\mathscr{F} = \mathsf{Id}$. In this case $(\sigma, \sigma') \in cl(R_{id}) = \overline{\mathsf{Id}}(cl(R_{id}))$, so the result follows immediately by Lemma 3.3.3.

  Induction step:
  - $\mathscr{F} = \mathscr{F}_1 \times \mathscr{F}_2$. Obviously, $\sigma = \langle \sigma_1, \sigma_2 \rangle$ and $\sigma' = \langle \sigma_1', \sigma_2' \rangle$, where $(\sigma_1, \sigma_1') \in \overline{\mathscr{F}_1}(cl(R_{id}))$ and $(\sigma_2, \sigma_2') \in \overline{\mathscr{F}_2}(cl(R_{id}))$. Therefore, by the induction hypothesis, both $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma_1} = \boxed{\sigma_1'}$ and $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma_2} = \boxed{\sigma_2'}$ hold. Hence, according to the definition of $\vdash_{NDF}$ (see (3.6)), we conclude that $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\langle \sigma_1, \sigma_2 \rangle} = \boxed{\langle \sigma_1', \sigma_2' \rangle}$ holds.
  - The cases $\mathscr{F} = \mathscr{F}_1 \oplus \mathscr{F}_2$, $\mathscr{F} = \mathscr{F}_1^A$ and $\mathscr{F} = \mathscr{P}_\omega \mathscr{F}'$ are handled in a similar way.

- "$\Leftarrow$". We proceed also by induction on the structure of $\mathscr{F}$. Moreover, recall that the observations in Remark 6 hold (for each of the subsequent cases).
  Base case:
  - $\mathscr{F} = \mathsf{B}$. In this case (σ, σ') is of shape (b, b'), where b, b' are two elements of the semilattice B. Also, recall that $\mathscr{G} \neq \mathsf{B}$, therefore, the equations (of type $\mathscr{G} \triangleleft \mathscr{G} \neq \mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$) in R are not involved in the equational reasoning. We deduce that $\boxed{b} = \boxed{b'}$ is proved by reflexivity, hence $(b, b') = (b, b) \in \overline{\mathsf{B}}(cl(R_{id}))$.
  - $\mathscr{F} = \mathsf{Id}$. Note that for this case, σ, σ' are expressions of the same type with the expressions in R. We further identify two possibilities:
    - $\boxed{\sigma} = \boxed{\sigma'}$ is proved by reflexivity. Therefore $(\sigma, \sigma') \in \{(\varepsilon, \varepsilon) \mid \varepsilon : \mathscr{G} \triangleleft \mathscr{G}\} \subseteq R_{id} \subseteq cl(R_{id}) = \overline{\mathsf{Id}}(cl(R_{id}))$.
    - The equations in $\boxed{R}$ are used in the equational reasoning $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma} = \boxed{\sigma'}$. In addition, the freezing operator inhibits contextual reasoning, therefore $\boxed{\sigma} = \boxed{\sigma'}$ is proved according to the equations in $\boxed{R}$, based on the symmetry and transitivity of $\vdash_{NDF}$. In other words, $(\sigma, \sigma') \in cl(R_{id}) = \overline{\mathsf{Id}}(cl(R_{id}))$.

  Induction step:
  - $\mathscr{F} = \mathscr{F}_1 \times \mathscr{F}_2$. Obviously, due to their type, the equations in R are not involved in the equational reasoning. Therefore, $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\langle \sigma_1, \sigma_2 \rangle} = \boxed{\langle \sigma_1', \sigma_2' \rangle}$ is a consequence of the inverted rule (3.6). More explicitly, it follows that $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma_1} = \boxed{\sigma_1'}$ and $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma_2} = \boxed{\sigma_2'}$ must hold. By the induction hypothesis, we deduce that $(\sigma_1, \sigma_1') \in \overline{\mathscr{F}}_1(cl(R_{id}))$ and $(\sigma_2, \sigma_2') \in \overline{\mathscr{F}}_2(cl(R_{id}))$. So by the definition of $\overline{\mathscr{F}_1 \times \mathscr{F}_2}$ we conclude that $(\langle \sigma_1, \sigma_2 \rangle, \langle \sigma_1', \sigma_2' \rangle) = (\sigma, \sigma') \in \overline{\mathscr{F}_1 \times \mathscr{F}_2}(R)$.

  * The cases $\mathscr{F} = \mathscr{F}_1 \oplus \mathscr{F}_2$, $\mathscr{F} = (\mathscr{F}_1)^A$ and $\mathscr{F} = \mathscr{P}_\omega\mathscr{F}'$ follow a similar reasoning.

- Proof of Theorem 3.3.2.b). It follows immediately by the definition of $\overline{\mathsf{B}}$ and Remark 6.

Remark 7 *For a more intuitive justification on the distinction of constant/non-constant functor in Theorem 3.3.2, note that in CIRC, proof obligations $\boxed{\varepsilon} = \boxed{\varepsilon'}$ of a type (sort) that serves as "base case" in the co-recursive definitions are not collected as hypotheses during a proof session. Hence, in the context of $\mathscr{G}$-expressions, whenever $\mathscr{G} = \mathsf{B}$, the hypotheses set R is empty. Consequently, a corresponding obligation $\boxed{\varepsilon} = \boxed{\varepsilon'}$ of type B is proved only according to the equations in $\mathscr{E}_\mathscr{G}$, by applying transitivity, symmetry and reflexivity.*

3.3.4 Corollary. *Let $\mathscr{G}$ be a non-deterministic functor and R a binary relation on the set of $\mathscr{G}$-expressions.*

a) If $\mathscr{G}$ is a non-constant functor, then $\mathrm{cl}(R_{id})$ is a bisimulation iff $\mathscr{E}_\mathscr{G} \cup \boxed{R} \vdash_{\mathrm{NDF}} \boxed{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(R)}$;

b) If $\mathscr{G}$ is a constant functor B, then $\mathrm{cl}(R_{id})$ is a bisimulation iff $\mathscr{E}_\mathscr{G} \vdash_{\mathrm{NDF}} \boxed{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(R)}$.

Proof.

- Proof of Corollary 3.3.4.a). We reason as follows:

$$\begin{aligned}
& cl(R_{id}) \text{ is a bisimulation} & \\
\Leftrightarrow\ & (\forall(\varepsilon,\varepsilon') \in cl(R_{id})).((\delta_{\mathscr{G}\triangleleft\mathscr{G}}(\varepsilon), \delta_{\mathscr{G}\triangleleft\mathscr{G}}(\varepsilon')) \in \overline{\mathscr{G}}(cl(R_{id})) & \\
\Leftrightarrow\ & \mathscr{E}_\mathscr{G} \cup \boxed{R} \vdash_{NDF} \boxed{\delta_{\mathscr{G}\triangleleft\mathscr{G}}(cl(R_{id}))} & \text{(Thm. 3.3.2)} \\
\Leftrightarrow\ & \mathscr{E}_\mathscr{G} \cup \boxed{R} \vdash_{NDF} \boxed{\delta_{\mathscr{G}\triangleleft\mathscr{G}}(R)} & (cl(R_{id}), \vdash_{NDF})
\end{aligned}$$

- Proof of Corollary 3.3.4.b). It follows immediately by the definition of bisimulation relations and according to the observations in Remark 6.

In Figure 3.2 we briefly summarise the results of the current section, namely, the algebraic encoding of the coalgebraic setting presented in [SBR10].

3.4 Deciding bisimilarity in CIRC

We next describe how the coinductive theorem prover CIRC [LGCR09] can be used to implement the decision procedure for the bisimilarity of generalised regular expressions, which we discussed above.

CIRC can be seen as an extension of Maude with behavioural features and its implementation is derived from that of Full-Maude. In order to use the prover, one needs to provide a specification (a CIRC theory) and a set of goals. A CIRC theory $\mathscr{B} = (S, (\Sigma, \Delta), (E, \mathscr{I}))$ consists of an algebraic specification (S, Σ, E), a set Δ of *derivatives*, and a set $\mathscr{I}$ of equational interpolants, which are expressions of the form $e \Rightarrow \{e_i \mid i \in I\}$ where e and e_i are equations. The intuition for this type of expressions is simple: e holds whenever for any i in I the equation e_i holds. In other words, to prove $E \vdash e$ one can chose to instead prove $E \vdash \{e_i \mid i \in I\}$. For the particular case of non-deterministic functors, we

coalgebraic	algebraic
$\vdash \varepsilon : \mathscr{F} \triangleleft \mathscr{G}$	$\mathscr{E}_{\mathscr{G}} \vdash \varepsilon : \mathscr{F} \triangleleft \mathscr{G} = true$
$\mathsf{Exp}_{\mathscr{F} \triangleleft \mathscr{G}}$	$\{\varepsilon \in \mathscr{T}_{\Sigma,\mathsf{Exp}} \mid \mathscr{E}_{\mathscr{G}} \vdash \varepsilon : \mathscr{F} \triangleleft \mathscr{G} = true\}$
$\mathsf{Exp}_{\mathscr{G}}$	$\{\varepsilon \in \mathscr{T}_{\Sigma,\mathsf{Exp}} \mid \mathscr{E}_{\mathscr{G}} \vdash \varepsilon : \mathscr{G} \triangleleft \mathscr{G} = true\}$
$\mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$	$\{\sigma \in \mathscr{T}_{\Sigma,\mathsf{ExpStruct}} \mid \mathscr{E}_{\mathscr{G}} \vdash \sigma \in \mathscr{F}(\mathsf{Exp}\,\mathscr{G}) = true\}$
$\delta_{\mathscr{F} \triangleleft \mathscr{G}} : \mathsf{Exp}_{\mathscr{F} \triangleleft \mathscr{G}} \to \mathscr{F}(\mathsf{Exp}_{\mathscr{G}})$	$\delta\_(\_) : \mathsf{Ingredient\ Exp} \to \mathsf{ExpStruct}$
$(\sigma, \sigma') \in \overline{\mathscr{F}}(cl(R_{id}))$	$\mathscr{E}_{\mathscr{G}} \vdash \sigma \in \mathscr{F}(\mathsf{Exp}\,\mathscr{G}) = true,$ $\mathscr{E}_{\mathscr{G}} \vdash \sigma' \in \mathscr{F}(\mathsf{Exp}\,\mathscr{G}) = true$ $\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\sigma} = \boxed{\sigma'}$ if $\mathscr{G} \neq \mathsf{B}$ or $\mathscr{E}_{\mathscr{G}} \vdash_{NDF} \boxed{\sigma} = \boxed{\sigma'}$ if $\mathscr{G} = \mathsf{B}$ (Thm. 3.3.2)
$cl(R_{id})$ is a bisimulation	$\mathscr{E}_{\mathscr{G}} \cup \boxed{R} \vdash_{NDF} \boxed{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(R)}$ if $\mathscr{G} \neq \mathsf{B}$ or $\mathscr{E}_{\mathscr{G}} \vdash_{NDF} \boxed{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(R)}$ if $\mathscr{G} = \mathsf{B}$ (Cor. 3.3.4)

Figure 3.2: Non-deterministic functors - coalgebraic vs. algebraic approach

use equational interpolants to extend the initial entailment relation in a consistent way with rules (3.6)–(3.9). (For more information on equational interpolants see [GLR10]). A derivative $\delta \in \Delta$ is a Σ-term containing a special variable $*{:}s$ (*i.e.*, a Σ-context), where s is the sort of the variable $*$. If e is an equation $t = t'$ with t and t' of sort s, then $\delta[e]$ is $\delta[t/*{:}s] = \delta[t'/*{:}s]$. We call this type of equation a *derivable equation*. The other equations are *non-derivable*. We write $\delta[R]$ to represent $\{\delta[e] \mid e \in R\}$, where R is a set of derivable equations, and $\Delta[e]$ for the set $\{\delta[e] \mid \delta \in \Delta \text{ appropriate for } e\}$.
Moreover, note that CIRC works with an extension of the entailment relation $\vdash$ over frozen equations (introduced in Section 3.3), with two more axioms, as in [RL09]:

$$E \cup R \vdash \boxed{e} \textit{ iff } E \vdash e \tag{3.10}$$

$$E \cup R \vdash G \textit{ implies } E \cup \delta[R] \vdash \delta[G] \textit{ for each } \delta \in \Delta \tag{3.11}$$

Above, E ranges over unfrozen equations, e over non-derivable unfrozen equations, and R, G over derivable frozen equations.

Remark 8 *Note that the new entailment $\vdash_{NDF}$ extended over frozen equations (in Definition 3.3.1) satisfies the assumptions (3.10) and (3.11).*

CIRC implements the coinductive proof system given in [RL09] using a set of reduction rules of the form $(\mathscr{B}, F, G) \Rightarrow (\mathscr{B}, F', G')$, where $\mathscr{B}$ represents a specification, F is the coinductive hypothesis (a set of frozen equations) and G is the current set of goals. The freezing operator is defined as described in Section 3.3. Here is a brief description of these rules:

[Done]: $(\mathscr{B}, F, \{\}) \Rightarrow \cdot$
Whenever the set of goals is empty, the system terminates with success.

[Reduce]: $(\mathscr{B}, F, G \cup \{\boxed{e}\}) \Rightarrow (\mathscr{B}, F, G)$ *if* $\mathscr{B} \cup F \vdash \boxed{e}$
If the current goal is a $\vdash$-consequence of $\mathscr{B} \cup F$ then $\boxed{e}$ is removed from the set of goals.

[Derive]: $(\mathscr{B}, F, G \cup \{\boxed{e}\}) \Rightarrow (\mathscr{B}, F \cup \{\boxed{e}\}, G \cup \boxed{\Delta[e]})$ *if* $\mathscr{B} \cup F \nvdash \boxed{e}$
When the current goal e is derivable and it is not a $\vdash$-consequence, it is added to the hypothesis and its derivatives to the set of goals.

[Simplify]: $(\mathscr{B}, F, G \cup \{\boxed{\theta(e)}\}) \Rightarrow (\mathscr{B}, F, G \cup \{\boxed{\theta(e_i)} \mid i \in I\})$
if $e \Rightarrow \{e_i \mid i \in I\}$ is an equational interpolant from the specification and $\theta : X \to \mathscr{T}_\Sigma(Y)$ is a substitution.

[Fail]: $(\mathscr{B}, F, G \cup \{\boxed{e}\}) \Rightarrow$ *failure if* $\mathscr{B} \cup F \nvdash \boxed{e} \wedge e$ *is non-derivable*
This rule stops the reduction process with failure whenever the current goal e is non-derivable and is not a $\vdash$-consequence of $\mathscr{B} \cup F$.

It is worth noting that there is a strong connection between a CIRC proof and the construction of a bisimulation relation. We illustrate this fact and the importance of the freezing operator with a simple example.

3.4.1 Example. Consider the case of infinite streams. The set B^ω of infinite streams over a set B is the final coalgebra of the functor $\mathscr{S} = \mathsf{B} \times \mathsf{Id}$, with a coalgebra structure given by hd and tl, the functions that return the head and the tail of the stream, respectively. Our purpose is to prove that $0^\infty = (00)^\infty$. Let z and zz represent the stream on the left-hand side and, respectively, on the right-hand side. These streams are defined by the equations: $hd(z) = 0, tl(z) = z, hd(zz) = 0, tl(zz) = 0{:}zz$. Note that equations over B like $hd(z) = 0$ are not derivable and equations over streams like $tl(z) = z$ are derivable.
In Figure 3.3 we present the correlation between the CIRC proof and the construction of the bisimulation relation. Note how CIRC collects the elements of the bisimulation as frozen hypotheses.
Let us analyze what would happen if the freezing operator $\boxed{-}$ was not used. Suppose the circular coinduction algorithm would add the equation $z = zz$ in its unfrozen form to the hypotheses. After applying the derivatives we obtain the goals $hd(z) = hd(zz), tl(z) = tl(zz)$. At this point, the prover could use the freshly added equation $z = zz$, and according to the congruence rule, both goals would be proven directly, though we would still be in the process of showing that the hypothesis holds. By following a similar reasoning, we could also prove that $0^\infty = 1^\infty$! In order to avoid these situations, the hypotheses are frozen, (*i.e.*, their sort is changed from Stream to Frozen) and this stops the application of the congruence rule, forcing the application of the derivatives according to their definition in the specification. Therefore, the use of the freezing operator is vital for the soundness of circular coinduction.

Next, we focus on using CIRC for automatically reasoning on the equivalence of $\mathscr{G}$-expressions. As we will show, the implementation of both the algebraic specifications associated with non-deterministic functors and the equational entailment relation described in Section 3.3 is immediate. Given a non-deterministic functor $\mathscr{G}$, we define a CIRC theory $\mathscr{B}_\mathscr{G} = (S, (\Sigma, \Delta), (E, \mathscr{I}))$ as follows:

- (S, Σ, E) is $\mathscr{E}_\mathscr{G}$
- $\Delta = \{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(*{:}\mathsf{Exp})\}$, so the only derivable equations are those of sort Exp. As we have already seen for the example of streams, equations of sort Slt must not be

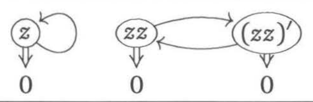
CIRC proof	Bisimulation construction
(add goal z = zz .)	
$(\mathscr{B}, \emptyset, \{\boxed{z} = \boxed{zz}\})$	$F = \emptyset;\ z \sim zz$?
$\xrightarrow{\text{[Derive]}} \left(\mathscr{B}, \{\boxed{z} = \boxed{zz}\}, \left\{\begin{matrix}\boxed{hd(z)}=\boxed{hd(zz)} \\ \boxed{tl(z)}=\boxed{tl(zz)}\end{matrix}\right\}\right)$	$F = \{(z,zz)\};\ \begin{matrix} z \xrightarrow{0} z \\ zz \xrightarrow{0} (zz)' \end{matrix}$
$\xrightarrow{\text{[Reduce]}} (\mathscr{B}, \{\boxed{z} = \boxed{zz}\}, \{\boxed{z} = \boxed{0{:}zz}\})$	$F = \{(z,zz)\};\ z \sim (zz)'$?
$\xrightarrow{\text{[Derive]}} \left(\mathscr{B}, \left\{\begin{matrix}\boxed{z}=\boxed{zz} \\ \boxed{z}=\boxed{0{:}zz}\end{matrix}\right\}, \left\{\begin{matrix}\boxed{hd(z)}=\boxed{hd(0{:}zz)} \\ \boxed{tl(z)}=\boxed{tl(0{:}zz)}\end{matrix}\right\}\right)$	$F = \{(z,zz),(z,(zz)')\};\ \begin{matrix} z \xrightarrow{0} z \\ (zz)' \xrightarrow{0} zz \end{matrix}$
$\xrightarrow{\text{[Reduce]}} \left(\mathscr{B}, \left\{\begin{matrix}\boxed{z}=\boxed{zz} \\ \boxed{z}=\boxed{0{:}zz}\end{matrix}\right\}, \{\}\right)$	$F = \{(z,zz),(z,(zz)')\}$ ✓

Figure 3.3: Parallel between a CIRC proof and the bisimulation construction

derivable. Since we have the subsort relation Slt < Exp, we avoid the application of the derivative $\delta_{\mathscr{G} \triangleleft \mathscr{G}}(*{:}\mathsf{Exp})$ over equations of sort Slt by means of an interpolant (see below).

- $\mathscr{I}$ consists of the following equational interpolants , whose role is to replace current proof obligations over non-trivial structures with simpler ones:

$$\langle\sigma_1,\sigma_2\rangle = \langle\sigma_1',\sigma_2'\rangle \ \Rightarrow\ \{\sigma_1 = \sigma_1',\ \sigma_2 = \sigma_2'\} \tag{3.12}$$
$$k_i(\sigma) = k_i(\sigma') \ \Rightarrow\ \{\sigma = \sigma'\} \tag{3.13}$$
$$f = g \ \Rightarrow\ \{f(a) = g(a) \mid a \in A\} \tag{3.14}$$
$$\begin{aligned}\cup_{i\in\overline{1,n}}\{\sigma_i\} = \cup_{j\in\overline{1,m}}\{\sigma_j'\} \ \Rightarrow\ &\{\wedge_{i\in\overline{1,n}}(\vee_{j\in\overline{1,m}}\,\sigma_i = \sigma_j') \\ &\wedge_{j\in\overline{1,m}}(\vee_{i\in\overline{1,n}}\,\sigma_i = \sigma_j')\}\end{aligned} \tag{3.15}$$

  together with an equational interpolant

$$t = t' \ \Rightarrow\ \{t \simeq t' = \mathsf{true}\} \tag{3.16}$$

  where $\simeq$ is the equality predicate equationally defined over the sort Slt. The last interpolant transforms the equations of sort Slt from derivable (because of the subsort relation Slt < Exp) into non-derivable and equivalent ones.

The interpolants (3.12–3.16) in $\mathscr{I}$ extend the entailment relation $\vdash_{NDF}$ (introduced in Definition 3.3.1) as follows:

$$\frac{E \vdash_{NDF} \{e_i \mid i \in I\}}{E \vdash_{NDF} e}\ \ if\ e \Rightarrow \{e_i \mid i \in I\}\ in\ \mathscr{I}$$

3.4.2 Theorem (Soundness). *Let $\mathcal{G}$ be a non-deterministic functor, and G a binary relation on the set of $\mathcal{G}$-expressions.*
If $(\mathscr{B}_{\mathcal{G}}, F_0 = \emptyset, G_0 = \boxed{G}) \overset{*}{\Rightarrow} (\mathscr{B}_{\mathcal{G}}, F_n, G_n = \emptyset)$ using [Reduce], [Derive] and [Simplify], then $G \subseteq \sim_{\mathcal{G}}$.

Proof. The idea of the proof is to find a bisimulation relation $\widetilde{F}$ s.t. $G \subseteq \widetilde{F}$.
First let F be the set of hypotheses (or derived goals) collected during the proof session. We distinguish between two cases:

a) $\mathcal{G} = \mathsf{B}$. For this case, the set of expressions in G is given by the following grammar:

$$\varepsilon ::= \underline{\emptyset} \mid b \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon\,. \tag{3.17}$$

Note that the goals $\varepsilon = \varepsilon'$ in G are proven

1. either according to [Simplify], applied in the context of the equational interpolant (3.16). If this is the case, then $\varepsilon = \varepsilon'$ holds by reflexivity, therefore

$$\mathscr{B}_{\mathcal{G}} \vdash_{NDF} \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon)} = \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon')} \tag{3.18}$$

also holds;

2. or after the application of [Derive], case in which $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon)} = \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon')}$ holds. Note that $\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon)$ and $\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon')$ are reduced to b, respectively $b' \in \mathsf{B}$, according to (3.17) and the definition of $\delta_{\mathsf{B}\triangleleft\mathsf{B}}$. Consequently, the non-derivable (due to the subsort relation $\mathsf{B} < \mathsf{Slt}$) goal $\boxed{b} = \boxed{b'}$ holds by reflexivity, so the following is a sound statement:

$$\mathscr{B}_{\mathcal{G}} \vdash_{NDF} \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon)} = \boxed{\delta_{\mathsf{B}\triangleleft\mathsf{B}}(\varepsilon')}. \tag{3.19}$$

Based on (3.18), (3.19) and Corollary 3.3.4.b), we conclude that $\widetilde{F} = cl(G_{id})$ is a bisimulation, hence $G \subseteq cl(G_{id}) \subseteq \sim_{\mathcal{G}}$.

b) $\mathcal{G} \neq \mathsf{B}$. Based on the reduction rules implemented in CIRC, it is quite easy to see that the initial set of goals G is a $\vdash_{NDF}$-consequence of $\mathscr{B}_{\mathcal{G}} \cup \boxed{F}$. In other words, $G \subseteq cl(F_{id})$. So, if we anticipate a bit, we should show that $\widetilde{F} = cl(F_{id})$ is a bisimulation, *i.e.*, according to Corollary 3.3.4, $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} \boxed{\delta_{\mathcal{G}\triangleleft\mathcal{G}}(F)}$. This is achieved by proving that $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} G_i \, (i \in \overline{0,n})$ (note that $\boxed{\delta_{\mathcal{G}\triangleleft\mathcal{G}}(F)} \subseteq \bigcup_{i\in\overline{0,n}} G_i$, according to [Derive]). The proof is by induction on j, where $n-j$ is the current proof step, and by case analysis on the CIRC reduction rules applied at each step.

We further provide a sketch of the proof.
The *base case* $j = n$ follows immediately, as $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} G_n = \emptyset$.
For the *induction step* we proceed as follows. Let $\boxed{e} \in G_j$. If $\boxed{e} \in G_{j+1}$ then $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} \boxed{e}$ by the induction hypothesis. If $\boxed{e} \notin G_{j+1}$ then, for example, if [Reduce] was applied then it holds that $\mathscr{B}_{\mathcal{G}} \cup F_j \vdash_{NDF} \boxed{e}$. Recall that $F_j \subseteq \boxed{F}$, so $\mathscr{B}_{\mathcal{G}} \cup \boxed{F} \vdash_{NDF} \boxed{e}$ also holds. The result follows in a similar fashion for the application of [Derive] or [Simplify].

Remark 9 *The soundness of the proof system we describe in this chapter does not follow directly from Theorem 3 in [RL09]. This is due to the fact that we do not have an experiment-based definition of bisimilarity. So, even though the mechanism we use for proving* $\mathscr{B}_{\mathscr{G}} \cup \boxed{F} \vdash_{NDF} \boxed{\delta_{\mathscr{G} \triangleleft \mathscr{G}}(F)}$ *(for the case* $\mathscr{G} \neq \mathsf{B}$*) is similar to the one described in [RL09], the current soundness proof is conceived in terms of bisimulations (and not experiments).*

Remark 10 *The entailment relation* $\vdash_{NDF}$ *that CIRC uses for checking the equivalence of generalised regular expressions is an instantiation of the parametric entailment relation* $\vdash$ *from the proof system in [RL09]. This approach allows CIRC to reason automatically on a large class of systems which can be modelled as non-deterministic coalgebras.*

As already stated, our final goal is to use CIRC as a decision procedure for the bisimilarity of generalised regular expressions. That is, whenever provided a set of expressions, the prover stops with a yes/no answer with respect to their equivalence. In this context, an important aspect is that the sub-coalgebra generated by an expression $\varepsilon \in \mathsf{Exp}_{\mathscr{G}}$ by repeatedly applying $\delta_{\mathscr{G}}$ is, in general, infinite. Take for example the non-deterministic functor $\mathscr{S} = \mathsf{B} \times \mathsf{Id}$ associated with infinite streams, and consider the property $\mu x.\underline{\emptyset} \oplus r\langle x\rangle = \mu x.r\langle x\rangle$. In order to prove this, CIRC builds an infinite proof sequence by repeatedly applying $\delta_{\mathscr{S}}$ as follows:

$$\begin{array}{rcl}
\delta_{\mathscr{S}}(\mu x.\underline{\emptyset} \oplus r\langle x\rangle) & = & \delta_{\mathscr{S}}(\mu x.r\langle x\rangle) \\
 & \downarrow & \\
\langle 0, \underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x\rangle)\rangle & = & \langle 0, \mu x.r\langle x\rangle\rangle \\
\delta_{\mathscr{S}}(\underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x\rangle)) & = & \delta_{\mathscr{S}}(\mu x.r\langle x\rangle) \\
 & \downarrow & \\
\langle 0, \underline{\emptyset} \oplus \underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x\rangle)\rangle & = & \langle 0, \mu x.r\langle x\rangle\rangle \ [\ldots]
\end{array}$$

In this case, the prover never stops. We observed in Section 3.2 that Theorem 3.1.5 guarantees we can associate a finite coalgebra to a certain expression. In the proof of the aforementioned theorem, which is presented in [SBR10], it is shown that the axioms for associativity, commutativity and idempotency (ACI) of $\oplus$ guarantee finiteness of the generated sub-coalgebra (note that these axioms have also been proven sound with respect to bisimulation). ACI properties can easily be specified in CIRC as the prover is an extension of Maude, which has a powerful matching modulo ACUI (ACI plus unity) capability. The idempotence is given by the equation $\varepsilon \oplus \varepsilon = \varepsilon$, and the commutativity and associativity are specified as attributes of $\oplus$. It is interesting to remark that for the powerset functor termination is guaranteed without the axioms, because the coalgebra structure on the expressions for the powerset functor already includes ACI (since $\mathscr{P}_\omega(\mathsf{Exp})$ is itself a join-semilattice).

3.4.3 Theorem. *Let G be a set of proof obligations over generalised regular expressions. CIRC can be used as a decision procedure for the equivalences in G, that is, it can decide whenever a goal* $(\varepsilon_1, \varepsilon_2) \in G$ *is a true or false equality.*

Proof. Note that as proven in [SBR10], the ACI axioms for $\oplus$ guarantee that $\delta_{\mathscr{G}}$ is applied for a finite number of times in the generation of the sub-coalgebra associated with a $\mathscr{G}$-expression. Therefore, it straightforwardly follows that by implementing the ACI axioms in CIRC (as attributes of $\oplus$), the set of new goals obtained by applying $\delta_{\mathscr{G}}$ is finite. In these circumstances, whenever CIRC stops according to the reduction rule [Done], the initial proof obligations are bisimilar. On the other hand, whenever it terminates with [Fail], the goals are not bisimilar.

3.5 A CIRC-based Tool

We have implemented a tool that, when provided with a functor $\mathcal{G}$, automatically generates a specification for CIRC which can then be used in order to automatically check whether two $\mathcal{G}$-expressions are bisimilar.
The tool is implemented as a metalanguage application in Maude. It can be downloaded from the address `http://goriac.info/tools/functorizer/`. In order to start the tool, one needs to launch Maude along with the extension Full-Maude and load the downloaded file using the command `in functorizer.maude` .
The general use case consists in providing the join-semilattices, the alphabets and the expressions. After these steps, the tool automatically checks if the provided expressions are guarded, closed and correctly typed. If this check succeeds, then it outputs a specification that can be further processed by CIRC. In the end, the prover outputs either the bisimulation, if the expressions are equivalent, or a negative answer, otherwise.
We present two case studies in order to emphasise the high degree of generality for the types of systems we can handle, and show how the tool is used.

3.5.1 Example. We consider the case of Mealy machines, which are coalgebras for the functor $(\mathsf{B} \times \mathsf{Id})^A$.
Formally, a Mealy machine is a pair (S, α) consisting of a set S of states and a transition function $\alpha\colon S \to (\mathsf{B} \times S)^A$, which for each state $s \in S$ and input $a \in A$ associates an output value b and a next state s'. Typically, we write $\alpha(s)(a) = (b, s') \iff \textcircled{s} \xrightarrow{a|b} \textcircled{s'}$.

In this example and in what follows we will consider for the output the two-value join-semilatice $\mathsf{B} = \{0, 1\}$ (with $\bot_{\mathsf{B}} = 0$) and for the input alphabet $A = \{a, b\}$. The expressions for Mealy machines are given by the grammar:

$$
\begin{array}{lll}
E & ::= & \underline{\emptyset} \mid x \mid E \oplus E \mid \mu x.E_2 \mid a(r\langle E\rangle) \mid b(r\langle E\rangle) \mid a(l\langle E_1\rangle) \mid b(l\langle E_1\rangle) \\
E_1 & ::= & \underline{\emptyset} \mid E_1 \oplus E_1 \mid 0 \mid 1 \\
E_2 & ::= & \underline{\emptyset} \mid E_2 \oplus E_2 \mid \mu x.E_2 \mid a(r\langle E\rangle) \mid b(r\langle E\rangle) \mid a(l\langle E_1\rangle) \mid b(l\langle E_1)
\end{array}
$$

Intuitively, an expression of shape $a(l\langle E_1\rangle)$ specifies a state that for an input a has an output value specified by E_1. For example, the expression $a(l\langle 1\rangle)$ specifies a state that for input a outputs 1, whereas in the case of $a(l\langle\underline{\emptyset}\rangle)$ the output is 0. An expression of shape $a(r\langle E\rangle)$ specifies a state that for a certain input a has a transition to a new state represented by E. For example, the expression $\mu x.a(r\langle x\rangle)$ states that for input a, the machine will perform a "a-loop" transition, whereas $a(r\langle\underline{\emptyset}\rangle)$ states that for input a there is a transition to the state denoted by $\underline{\emptyset}$. It is interesting to note that a state will only be fully specified in what concerns transitions and output (for a given input a if both $a(l\langle E_1\rangle)$ and $a(r\langle E\rangle)$ appear in the expression (combined by $\oplus$). In the case only transition (respectively, output) are specified, the underspecification is solved by setting the target state (respectively, output) to $\underline{\emptyset}$ (respectively, $\bot_B = 0$). ♠

Next, to provide the reader with intuition, we will explain how one can reason on the bisimilarity of two simple expressions, by constructing bisimulation relations. Later on, we show how CIRC can be used in conjunction with our tool in order to act as a decision procedure when checking equivalence of two expressions, in a fully automated manner.
We will start with the expressions $\varepsilon_1 = \mu x.a(r\langle x\rangle)$ and $\varepsilon_2 = \underline{\emptyset}$. We have to build a bisimulation relation R on $\mathcal{G}$-expressions, such that $(\varepsilon_1, \varepsilon_2) \in R$. We do this in the following

way: we start by taking $R = \{(\varepsilon_1, \varepsilon_2)\}$ and we check whether this is already a bisimulation, by considering the output values and transitions and check whether no new expressions appear in this process. If new pairs of expressions appear we add them to R and repeat the process. Intuitively, this can be represented as follows:

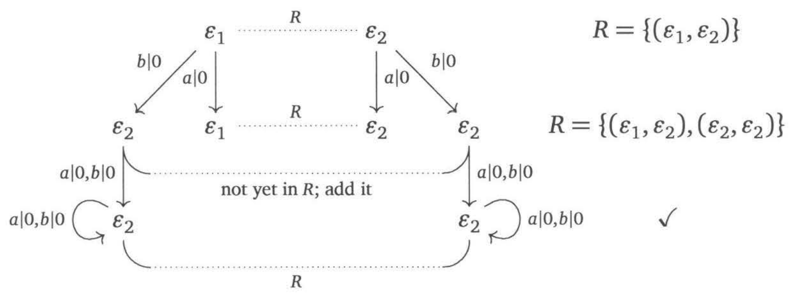
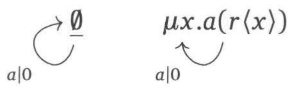

$R = \{(\varepsilon_1, \varepsilon_2)\}$
$R = \{(\varepsilon_1, \varepsilon_2), (\varepsilon_2, \varepsilon_2)\}$
not yet in R; add it
✓

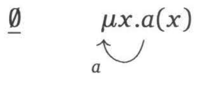
Figure 3.4: Bisimulation construction

In the figure above, and as before, we use the notation $\varepsilon_1 \overset{R}{\text{———}} \varepsilon_2$ to denote $(\varepsilon_1, \varepsilon_2) \in R$. As illustrated in Figure 3.4, $R = \{(\varepsilon_1, \varepsilon_2), (\varepsilon_2, \varepsilon_2)\}$ is closed under transitions and is therefore a bisimulation. Hence, $\varepsilon_1 \sim_{\mathcal{G}} \varepsilon_2$.

The proved equality $\underline{\emptyset} = \mu x.a(r\langle x\rangle)$ might seem unexpected, if the reader is familiar with labelled transition systems. The equality is sound because these are expressions specifying behaviour of a Mealy machine and, semantically, both denote the function that for every non-empty word outputs 0 (the semantics of Mealy machines is given by functions B^{A^+}, intuitively one can think of these expressions as both denoting the empty language). This is visible if one draws the automata corresponding to both expressions (say, for simplicity, the alphabet is $A = \{a\}$):



Note that (i) the $\underline{\emptyset}$ expression for Mealy machines is mapped with δ to a function that for input a gives $\langle 0, \underline{\emptyset}\rangle$, which represents a state with an a-loop to itself and output 0; (ii) the second expression specifies explicitly an a-loop to itself and it also has output 0, since no output value is explicitly defined. Now, also note that similar expressions for labelled transition systems (LTS's), or coalgebras of the functor $\mathcal{P}_\omega(-)^A$, would not be bisimilar since one would have an a-transition and the other one not. This is because the $\underline{\emptyset}$ expression for LTS's really denotes a deadlock state. In operational terms they would be converted to the systems



which now have an obvious difference in behaviour.

By performing a similar reasoning as in the example above one can show that the expressions $\varepsilon_1 = \mu x.a(r\langle x\rangle) \oplus b(r\langle x\rangle)$ and $\varepsilon_2 = \mu x.a(r\langle x\rangle)$ are bisimilar, and the bisimulation relation is built as illustrated in Figure 3.5:

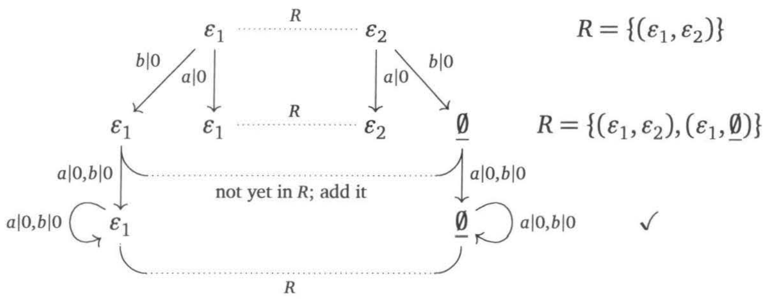

R
b|0
a|0
a|0
b|0
R
a|0,b|0
not yet in R; add it
a|0,b|0
a|0,b|0
a|0,b|0
R
$R = \{(\varepsilon_1, \varepsilon_2)\}$
$R = \{(\varepsilon_1, \varepsilon_2), (\varepsilon_1, \underline{\emptyset})\}$
✓

Figure 3.5: Bisimulation construction

Let us further consider the Mealy machine depicted in Figure 3.6, where all states are bisimilar.

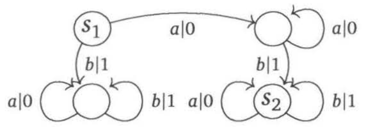


Figure 3.6: Mealy machine: $s_1 \sim s_2$

We show how to check the equivalence of two expression characterising the states s_1 and s_2, in a fully automated manner, using CIRC. These expressions are $\varepsilon_1 = \mu x.b(l\langle 1\rangle) \oplus b(r\langle \varepsilon_2\rangle) \oplus a(\mu y.a(r\langle y\rangle) \oplus b(r\langle \varepsilon_2\rangle) \oplus b(l\langle 1\rangle))$ and $\varepsilon_2 = \mu x.b(l\langle 1\rangle) \oplus b(r\langle x\rangle) \oplus a(r\langle x\rangle)$, respectively.
In order to check bisimilarity of ε_1 and ε_2 we load the tool and define the semilattice $\mathsf{B} = \{0, 1\}$ and the alphabet $A = \{a, b\}$:

```
(jslt B is 0 1 bottom 0 . 0 v 0 = 0 . 0 v 1 = 1 . 1 v 1 = 1 . endjslt)
(alph A is a b endalph)
```

We provide the functor $\mathcal{G}$ using the command (functor (B x Id)^A .). The command (set goal ... .) specifies the goal we want to prove:

```
(set goal
 \mu X:FixpVar . b(l<1>) (+) a(l<0>) (+) b(r<X:FixpVar>) (+)
                 a(r<X:FixpVar>) =
 \mu X:FixpVar . b(l<1>) (+) b(<\mu X:FixpVar . b(l<1>) (+)
                 b(r<X:FixpVar>) (+) a(r<X:FixpVar>)>) (+)
                 a(\mu Y:FixpVar . a(r<Y:FixpVar>) (+)
                 b(<\mu X:FixpVar . b(l<1>) (+) a(l<0>) (+)
                 b(r<X:FixpVar>) (+) a(r<X:FixpVar>)>) (+) b(l<1>)) .)
```

In order to generate the CIRC specification we use the command (generate coalgebra .). Next we need to load CIRC along with the resulting specification and start the proof engine using the command (coinduction .).
As already shown, behind the scenes, CIRC builds a bisimulation relation that includes the initial goal. The proof succeeds and the output consists of (a subset of) this bisimulation:

```
Proof succeeded.
  Number of derived goals: 2
  Number of proving steps performed: 50
  Maximum number of proving steps is set to: 256

Proved properties:
- phi (+) (\mu X . a(l<0>) (+) a(r<X>) (+) b(l<1>) (+) b(r<X>)) =
  phi (+) (\mu Y . a(r<Y>) (+) b(l<1>) (+)
  b(r<\mu X . a(l<0>) (+) a(r<X>) (+) b(l<1>)(+)b(r<X>)>))

- \mu X . a(l<0>) (+) a(r<X>) (+) b(l<1>) (+) b(r<X>) =
  \mu Z . a(r<\mu Y . a(r<Y>) (+) b(l<1>) (+)
          b(r<\mu X . a(l<0>) (+) a(r<X>) (+) b(l<1>) (+) b(r<X>)>)>) (+)
          b(l<1>) (+) b(r<\mu X . a(l<0>) (+) a(r<X>) (+)
          b(l<1>) (+) b(r<X>)>)
```

For the ease of understanding, here we printed a readable version of the proved properties. In Section 3.5.1, however, we show that internally each expression is brought to a canonical form by renaming the variables. Moreover, note that in our tool, $\underline{\emptyset}$ is represented by the constant phi. All the examples provided in the current section make use of this convention.
As previously mentioned, CIRC is also able to detect when two expressions are not equivalent. Take, for instance, the expressions $\mu x.a(l\langle 0\rangle) \oplus a(r\langle a(l\langle 1\rangle) \oplus a(r\langle x\rangle)\rangle)$ and $a(l\langle 0\rangle) \oplus a(r\langle a(r\langle \mu x.a(r\langle x\rangle) \oplus a(l\langle 0\rangle)\rangle) \oplus a(l\langle 1\rangle)\rangle)$, characterising the states s_1 and s_3 from the Mealy machines in Figure 3.7. After following some steps similar to the ones previously enumerated, the proof fails and the output message is Visible goal [...] failed during coinduction.

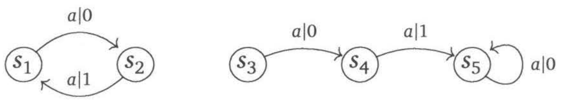


Figure 3.7: Mealy machines: $s_1 \not\sim s_3$

3.5.2 Example. Next we show how to check strong bisimilarity of non-deterministic processes of a non-trivial CCS-like language with termination, deadlock, and divergence, as studied in [AH92]. A process is a guarded, closed term defined by the following grammar:

$$P \quad ::= \quad \surd \mid \delta \mid \Omega \mid a.P \mid P + P \mid x \mid \mu x.P \tag{3.20}$$

where:

- $\surd$ is the constant for successful termination,
- δ denotes deadlock,
- Ω is the divergent computation (*i.e.*, the undefined process),
- $a.P$ is the process executing the action a and then continuing as the process P, for any action a from a given set A,
- $P_1 + P_2$ is the non-deterministic process behaving as either P_1 or P_2, and

- $\mu x.P$ is the recursive process $P[\mu x.P/x]$.

In [SBR10] is is shown that, up to strong bisimilarity, the above syntax of processes is equivalent to the canonical set of (guarded, closed) regular expressions derived for the functor $1 \oplus \mathcal{P}_\omega(\mathsf{Id})^A$,

$$\begin{array}{lcl} E & ::= & \emptyset \mid E \oplus E \mid x \mid \mu x.E \mid l[E_1] \mid r[E_2] \\ E_1 & ::= & \emptyset \mid E_1 \oplus E_1 \mid 1 \\ E_2 & ::= & \emptyset \mid E_2 \oplus E_2 \mid a(E_3) \\ E_3 & ::= & \emptyset \mid E_3 \oplus E_3 \mid \{E\} \end{array}$$

The translation map $(-)^\dagger$ from processes to expressions is defined by induction on the structure of the process:

$$\begin{array}{lcl@{\qquad}lcl} (\checkmark)^\dagger & = & l[1] & (a.P)^\dagger & = & r[a(\{P^\dagger\})] \\ (\delta)^\dagger & = & r[\emptyset] & (P_1+P_2)^\dagger & = & (P_1)^\dagger \oplus (P_2)^\dagger \\ (\Omega)^\dagger & = & \emptyset & (\mu x.P)^\dagger & = & \mu x.P^\dagger \\ x^\dagger & = & x\,. & & & \end{array}$$

Consider now two processes P and Q over the alphabet $A = \{a, b\}$:

$$\begin{array}{lcl} P & = & \mu x.(a.x + a.P_1 + b.b.\checkmark + b.(\delta + \Omega)) \\ Q & = & \mu z.(a.z + b.(\delta + b.\checkmark) + b.\delta) \end{array}$$

where $P_1 = \mu y.(a.(y+\delta)+b.\delta+b.(\delta+b.\checkmark)+\delta)$. Graphically, the two processes can be represented by the following labelled transition systems (for simplicity we omit annotating states with information regarding the satisfiability of successful termination, divergence, and deadlock):

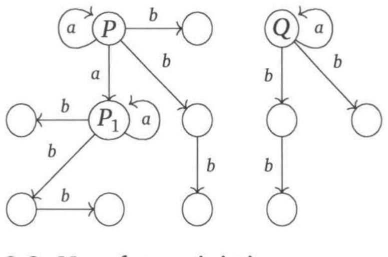


Figure 3.8: Non-deterministic processes: $Q \sim P$

We want to check if the process P is strongly bisimilar to the process Q. By using the above translation, process P is represented by the expression

$$\begin{array}{r} \mu x.(r[a(\{\mu y.(r[a(\{y \oplus r[\emptyset]\})] \oplus r[b(\{r[\emptyset]\})] \oplus \\ r[b(\{r[\emptyset] \oplus r[b(\{l[1]\})]\})] \oplus r[\emptyset])\})] \oplus \\ r[a(\{x\})] \oplus r[b(\{r[b(\{l[1]\})]\})] \oplus r[b(\{r[\emptyset] \oplus \emptyset\})]) \end{array}$$

whereas process Q is represented by the expression

$$\mu z.(r[a(\{z\})] \oplus r[b(\{r[\emptyset] \oplus r[b(\{l[1]\})]\})] \oplus r[b(\{r[\emptyset]\})]). \qquad \spadesuit$$

In order to use the tool, one needs to specify the semilattice, the alphabet, the functor, and the goal in a manner similar to the one previously presented:

```
(jslt B is 1 bottom 1 . 1 v 1 = 1 . endjslt)
(alph A is a b endalph)
(functor B + (P Id)^A .)
```

```
(set goal \mu X:FixpVar .
          r[ a( { X:FixpVar } ) ] (+)
          r[ a( { \mu Y:FixpVar .
                  r[ a( { Y:FixpVar (+) r[ phi ] } ) ] (+)
                  r[ b( { r[ phi ] } ) ] (+)
                  r[ b( { r[ phi ] (+) r[ b( { l[ 1 ] } ) ] } ) ] (+)
                  r[ phi ]
              } )
          ] (+)
          r[ b( { r[ b( { l[ 1 ] } ) ] } ) ] (+)
          r[ b( { r[ phi ] (+) phi } ) ]
          =
          \mu Z:FixpVar .
          r[ a( { Z:FixpVar } ) ] (+)
          r[ b( { r[ phi ] (+) r[ b( { l[ 1 ] } ) ] } ) ] (+)
          r[ b( { r[ phi ] } ) ]  .)
```

For the generated specification CIRC terminates and outputs a positive result:

```
Proof succeeded.
  Number of derived goals: 15
  Number of proving steps performed: 58
  Maximum number of proving steps is set to: 256

Proved properties:
- r[phi] (+) (\mu Y. r[phi] (+) r[a({r[phi] (+) Y})] (+) r[b({r[phi]})]
  (+) r[b({r[phi] (+) r[b({l[1]})]})])
  =
  \mu Z. r[a({Z})] (+) r[b({r[phi]})] (+) r[b({r[phi] (+) r[b({l[1]})]})]
- r[b({l[1]})] = r[phi] (+) r[b({l[1]})]
- \mu Y. r[phi] (+) r[a({r[phi] (+) Y})] (+) r[b({r[phi]})] (+)
  r[b({r[phi] (+) r[b({l[1]})]})]
  =
  \mu Z. r[a({Z})] (+) r[b({r[phi]})] (+) r[b({r[phi] (+) r[b({l[1]})]})]
- \mu X. r[a({X})] (+) r[a({\mu Y. r[phi] (+) r[a({r[phi] (+) Y})] (+)
  r[b({r[phi]})] (+) r[b({r[phi] (+) r[b({l[1]})]})]})] (+)
  r[b({r[phi] + phi})] (+) r[b({r[b({l[1]})]})]
  =
  \mu Z. r[a({Z})] (+) r[b({r[phi]})] (+) r[b({r[phi] (+) r[b({l[1]})]})]
```

3.5.1 Implementation

In this section we present details on the implementation of the algebraic specification given in Section 3.3, based on the examples from Section 3.5.
In order to generate the algebraic specifications for CIRC when provided a functor and two expressions, we used the Maude system [CDE$^+$07]. We choose it for its suitability for performing equational and rewriting logic based computations, and because of its reflective properties allowing for the development of advanced metalanguage applications. As

the technical aspects on how to work at the meta-level are beyond the scope of this paper, we refrain from presenting them and show, instead, what the generated specifications consist of.
Most of the algebraic specifications from Section 3.3 have a straightforward implementation in Maude. Consider, for instance, the case of Mealy machines presented in Example 3.5.1. The generated grammars for functors (3.1) and expressions (Definition 3.1.2) are coded as:

```
sort Functor .                          sorts Exp ExpStruct Alph Slt .
sorts AlphName SltName .                subsort Exp < ExpStruct .
subsort SltName < Functor .             enum A is a b . enum B is 0 1 .
                                        subsort A < Alph .
op A : -> AlphName .                    subsort B < Slt .
op B : -> SltName .
op G : -> Functor .                     op _`(+`)_ : Exp Exp -> Exp .
op Id : -> Functor .                    op _`(_`) : Alph Exp -> Exp .
op _+_ : Functor Functor -> Functor .   op \mu_._ : FixpVar Exp -> Exp .
op _^_ : Functor AlphName -> Functor .  ops l<_> r<_> : Exp -> Exp .
op _x_ : Functor Functor -> Functor .   op phi : -> Exp .

                         eq G = (B x Id) ^ A .
```

Most of the syntactical constructs are Maude-specific: sorts and subsort declare the sorts we work with and, respectively, the relations between them; op declares operators; eq declares equations (the equation in our case defines the shape of the functor G). The only CIRC-specific construct, enum, is syntactic sugar for declaring enumerable sorts, *i.e.*, sorts that consist only of the specified constants. As a side note, if brackets ((, [, {) are used in the declaration of an operation, then they must be preceded by a backquote (`).
As mentioned in Section 3.1, in order to guarantee the finiteness of our procedure, one needs to include the ACI axioms for $(+)$. Moreover, we have observed that the unity axiom for $(+)$ plays an important role in decreasing the number of states generated by the repeated application of $\delta_{\mathcal{G}}$, therefore improving the overall time performance of the tool. For example, the number of rewritings CIRC performed in order to prove the bisimilarity of ε_1 and ε_2 in Figure 3.5 was halved when the unity axiom was used.
By turning on the axiomatisation flag using the command (axioms on .), the following code is generated:

```
op _`(+`)_ : Exp Exp -> Exp [assoc comm] .
eq E:Exp (+) E:Exp = E:Exp .
eq E:Exp (+) phi = E:Exp .
```

It is an obvious question why not to add other axioms to the tool, since the unity axiom has improved performance. At this stage we have not studied in detail how much adding other axioms would help. It is in any case a trade-off on how many extra axioms one should include, which will get the automaton produced from an expression closer to the minimal automaton, and how much time the tool will take to reduce the expressions in each step modulo the axioms. For classical regular expressions, there is an interesting empirical study on this [ORT09]. We leave it as future work to carry on a similar study for our expressions and axioms.
The process of substituting fixed-point variables has a natural implementation. We present the equations handling the basic expressions $\underline{\emptyset}$ and x, and the operation $(+)$:

```
op _'[_/_'] : Exp Exp FixpVar -> Exp .
eq phi [ E:Exp / X:FixpVar ] = phi .
ceq Y:FixpVar [ E:Exp / X:FixpVar ] = E:Exp if (X:FixpVar == Y:FixpVar) .
eq Y:FixpVar [ E:Exp / X:FixpVar ] = Y:FixpVar [owise] .
eq (E1:Exp (+) E2:Exp) [ E:Exp / X:FixpVar ] =
   (E1:Exp [E:Exp / X:FixpVar]) (+) (E2:Exp [E:Exp / X:FixpVar]) .
```

In order to avoid matching problems and to overcome the fact that in Maude one cannot handle an equation that has fresh variables in its right-hand-side (*i.e.*, they do not appear in the left-hand-side), we replace expression variables with parameterised constants: op var : Nat -> FixpVar . The operation that obtains this canonical form has an inductive definition on the structure of the given expression and makes use of the substitution operation presented above. For this reason, the bisimulation CIRC builds contains parameterised constants instead of the user declared variables. The property proved in Example 3.5.2 is, therefore, written as:

```
\mu var(2) . r[a({var(2)})] (+) r[a({\mu var(1) . r[phi] (+)
r[a({r[phi] (+) var(1)})] (+) r[b({r[phi]})] (+) r[b({r[phi] (+)
r[b({l[1]})]})]})] (+) r[b({r[phi] (+) phi})] (+) r[b({r[b({l[1]})]})]
=
\mu var(1) . r[a({var(1)})] (+) r[b({r[phi]})] (+)
r[b({r[phi] (+) r[b({l[1]})]})]
```

The most important part of the algebraic specification consists of the equations defining the operations $\delta\_(\_)$, *Plus*_(_,_), and *Empty*. Most of these equations are implemented as presented in [SBR10]. The only difficulties we encountered were for the exponentiation case, as Maude does not handle higher-order functions. Without entering into details, as a workaround, we introduced a new sort Function < ExpStruct and an operation \. : ExpoCase Alph Functor ExpStruct -> Function in order to emulate function-passing. The first argument is used to memorize the origin where the exponentiation ingredient is encountered: δ, *Plus*, or *Empty*. Its purpose is purely technical – we use it in order to avoid some internal matching problems. The other three parameters are those of the structured expression $\lambda.(a, \mathscr{F} \triangleleft \mathscr{G}, \sigma)$ presented in Section 3.3: a letter in the alphabet, an ingredient, and some other structured expression.
Another thing worth describing is the way we enable CIRC to prove equivalences when the powerset functor occurs. Namely, we present how interpolant (3.15) is implemented. Recall that we want to show that two sets of expressions are equivalent, which means that for each expression in the first set there must be an equivalent one in the second set and vice-versa.
In order to handle sets of structured expressions we introduce a new sort, ExpStructSet as a supersort for ExpStruct. We also consider the set separator _,_ : ExpStructSet ExpStructSet -> ExpStructSet [assoc,comm], the empty set emptyS : -> ExpStructSet, and the set wrapping operation {_} : ExpStructSet -> ExpStruct. In order to mimic universal quantification over a set, we use a special constant referred to as token "[/]". In what follows, we consider two variables of sort ExpStructSet: ES and ES', and two variables of sort ExpStructSet: ESS and ESS'. We now describe the process of finding the equivalence between two sets:

- whenever encountering two wrapped expression sets we add the universal quantification token to each of them in two distinct goals:

```
srl {ESS} = {ESS'} => {[/] ESS} = {ESS'} /\ {ESS} = {[/] ESS'} .
```

– iterate through the expressions on the left-hand-side (similarly for the other direction):

```
srl {[/] (ES , ESS)} = {ESS'} =>
    {[/] ES} = {ESS'} /\ {[/] ESS} = {ESS'} .
srl {ESS} = {[/] (ES' , ESS')} =>
    {ESS} = {[/] ES'} /\ {ESS} = {[/] ESS'} .
```

– when left with one expression on the left-hand-side, start iterating through the expressions on the right-hand-side until finding an equivalence (similarly for the other direction):

```
srl {[/] ES} = {ES' , ESS'} => ES = ES' \/ {[/] ES} = {ESS'} .
srl {ES , ESS} = {[/] ES'}  => ES = ES' \/ {ESS} = {[/] ES'} .
```

– if no equivalence has been found, transform the current goal into a visible failure:

```
srl {ESS} = emptyS => true = false .
srl emptyS = {ESS} => true = false .
```

Finally, the type checker for structured expressions has a straightforward implementation. Its code does not appear in the generated specification as it is only used when the tool receives the expressions as input. This prevents obtaining the specification and starting the prover in case invalid expressions are provided.

3.6 Discussion

In this chapter we provided a decision procedure for the bisimilarity of generalised regular expressions. In order to enable the implementation of the decision procedure, we have exploited an encoding of coalgebra into algebra, and we formalised the equivalence between the coalgebraic concepts associated with non-deterministic coalgebras [SBR10] and their algebraic correspondents. This led to the definition of algebraic specifications ($\mathscr{E}_{\mathscr{G}}$) that model both the language and the coalgebraic structure of expressions. Moreover, we defined an equational deduction relation ($\vdash_{NDF}$), used on the algebraic side for reasoning on the bisimilarity of expressions.
The most important result of the parallel between the coalgebraic and algebraic approaches is given in Corollary 3.3.4, which formalises the definition of the bisimulation relations in algebraic terms. Actually, this result is the key for proving the soundness of the decision procedure implemented in the automated prover CIRC [LGCR09]. As a coinductive prover, CIRC builds a relation F closed under the application of $\delta_{\mathscr{G}}$ with respect to $\vdash_{NDF}$ ($\mathscr{E}_{\mathscr{G}} \cup \boxed{F} \vdash_{NDF} \boxed{\delta_{\mathscr{G}}(F)}$), hence automatically computing a bisimulation the initial proof obligations belong to.
The approach we present in this chapter enables CIRC to perform reasoning based on bisimulations (instead of experiments [RL09]). This way, the prover is extended to checking bisimilarity in a large class of systems that can be modelled as non-deterministic coalgebras. Note that the constructions above are all automated – the (non-trivial) CIRC algebraic specification describing $\mathscr{E}_{\mathscr{G}}$, together with the interpolants implementing $\vdash_{NDF}$ are generated with the Maude tool presented in Section 3.5.

Chapter 4

Decorated trace and testing semantics coalgebraically

The study of behavioural equivalence of systems has been a research topic in concurrency for many years now. For different kinds of systems, several types of behavioural equivalences and preorders have been proposed throughout the years, each suitable for use in different contexts of application.

In Chapter 3 we showed how (co)algebras can be used in order to model and reason on bisimilarity of expressions describing non-deterministic systems.

The focus of this chapter is on a suite of other semantics of interest for labelled transition systems (LTS's), generative probabilistic systems (GPS's) and labelled transition systems with divergence. More explicitly, we consider *decorated trace semantics* including ready, failure, (complete) trace, possible-futures, ready trace and failure trace for LTS's, as described in [vG01a] and ready, (maximal) failure and (maximal) trace for GPS's, as introduced in [JS90]. For the case of divergent LTS's, the emphasis is on *must* and *may testing semantics* [CH89].

In short, our approach consists in providing a coalgebraic modelling of the aforementioned systems and their semantics. The latter are derived by employing the generalised powerset construction [SBBR13] and proved equivalent with their counterparts as defined in [CH89, vG01a, JS90]. This further allows reasoning on the corresponding notions of behavioural equivalence/preorder in terms of (Moore-) bisimulations.

We further provide the intuition behind decorated trace and testing semantics.

At the left-hand side of Figure 4.1 we illustrate the hierarchy (based on the coarseness level) among bisimilarity, ready, failure, (complete) trace, possible-futures, ready trace and failure trace semantics for LTS's, as introduced in [vG01a]. On the right-hand side a similar hierarchy is depicted for bisimilarity, ready, (maximal) failure and (maximal) trace semantics for GPS's, as in [JS90]. For example, for both types of systems, bisimilarity (the standard behavioural equivalence on $\mathcal{F}$-coalgebras) is the finest of the semantics, whereas trace semantics is the coarsest one. Moreover, note that for the case of GPS's, maximality does not yield more distinguishing power and ready and failure semantics are equivalent.

In order to get some intuition on the type of distinctions the equivalences above encom-

LTS's:

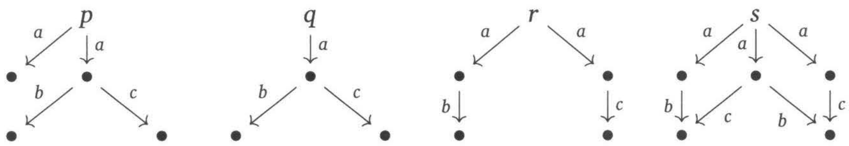
$$\begin{tikzcd}
 & \text{bisimilarity} \arrow[d] \arrow[dr] & \\
 & & \text{possible-futures} \arrow[dd] \\
 & \text{ready trace} \arrow[dl] \arrow[dr] & \\
\text{failure trace} \arrow[dr] & & \text{ready} \arrow[dl] \\
 & \text{failure} \arrow[d] & \\
 & \text{complete trace} \arrow[d] & \\
 & \text{trace} &
\end{tikzcd}$$

GPS's:

$$\begin{tikzcd}
 & \text{bisimilarity} \arrow[dl] \arrow[dr] & \\
\text{ready} \arrow[rr, Leftrightarrow] \arrow[dr] & & \text{(max.) failure} \arrow[dl] \\
 & \text{(max.) trace} &
\end{tikzcd}$$

Figure 4.1: Lattices of semantic equivalences for LTS's and GPS's.

pass, consider the following LTS's:



None of the top states of the systems above are bisimilar. The state p is the only one among the four in which an action a can lead to a deadlock state, whereas q, r and s have a different branching structures.
The traces of the states p, q, r and s are $\{a, ab, ac\}$, and therefore they are all trace equivalent. Of the four states above, q and r and s are complete trace equivalent as they can execute the same traces that lead to states where no further action are possible, whereas p is the only state that can trigger a and terminate.
Ready (respectively, failure) semantics identifies states according to the set of actions they can (respectively, fail to) trigger immediately after a certain trace has been executed. None of the states above are ready equivalent; for example, after the execution of action a, process p can reach a deadlock state whereas q has always to choose between actions b and c. Orthogonally, only r and s are failure equivalent.
Possible-futures semantics identifies states that can perform the same traces w and, moreover, the states reached by executing such w's are trace equivalent. None of the states above are possible-futures equivalent. For example, after triggering action a, p can reach a deadlock state (with no further behaviour) whereas q can execute the set of traces $\{b, c\}$.
Ready (respectively failure) trace semantics identifies states that can trigger the same traces w and the (pairwise-taken) intermediate states determined by such w's are ready (respectively refuse) to trigger the same sets of actions. None of the systems above is ready trace equivalent. For example, after performing action a, process q reaches a state that is ready to trigger both b and c, whereas r cannot. The analysis on failure trace equivalence follows a similar reasoning, but different results.
The corresponding semantic equivalences in Figure 4.1 distinguish between p, q, r and s

as summarised in the table below:

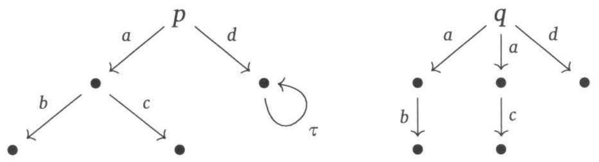
	p,q	p,r	p,s	q,r	q,s	r,s
bisimilarity	×	×	×	×	×	×
trace	✓	✓	✓	✓	✓	✓
complete trace	×	×	×	✓	✓	✓
ready	×	×	×	×	×	×
failure	×	×	×	×	×	✓
possible-futures	×	×	×	×	×	×
ready trace	×	×	×	×	×	×
failure trace	×	×	×	×	×	✓

where ✓ to stands for a "yes" answer with respect to the behavioural equivalence of two of the states p, q, r and s, whereas × represents a "no" answer.
Intuitively, GPS's resemble LTS's, with the difference that each transition is labelled by both an action and the probability of that action being executed. For more insight on decorated trace semantics for GPS's, consider the following systems:

$$\begin{tikzcd}
 & p' \arrow[d, "{a[1]}"] & \\
 & \bullet \arrow[dl, "{b[x]}"'] \arrow[dr, "{b[1-x]}"] & \\
\bullet \arrow[d, "{c[1]}"'] & & \bullet \arrow[d, "{d[1]}"] \\
\bullet & & \bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
 & q' \arrow[dl, "{a[x]}"'] \arrow[dr, "{a[1-x]}"] & \\
\bullet \arrow[d, "{b[1]}"'] & & \bullet \arrow[d, "{b[1]}"] \\
\bullet \arrow[d, "{c[1]}"'] & & \bullet \arrow[d, "{d[1]}"] \\
\bullet & & \bullet
\end{tikzcd}$$

In the setting of GPS's, decorated trace semantics take into consideration paths w which can be executed by a probabilistic process p. Reasoning on the corresponding equivalences is based on the sum of probabilities of occurrence of such w's that, for example, lead p to a set of processes, for the case of trace semantics, or to a set of processes that (fail to) trigger the same sets of actions as a first step, for ready (respectively, failure) semantics.
In [JS90] a notion of *maximality* was introduced for the case of trace and failure semantics. Intuitively, the former takes into consideration the probability of a process p to execute a certain trace w and terminate, whereas the latter takes into consideration the largest set of actions p fails to trigger as a first step after the execution of w. However, it has been proven in [JS90] that maximality does not increase the distinguishing power of decorated trace semantics and, moreover, ready and failure equivalence of GPS's coincide.
With respect to (maximal) trace semantics, amongst the systems above, p' and q' are equivalent: they have the same probability of executing traces $w \in \{\varepsilon, a, ab, abc, abd\}$. Moreover, each such w leads p' and q' to sets of processes S_1, S_2 ready to fire the same actions. Consequently, S_1 and S_2 fail to trigger the same sets of actions as a first step. Hence, p' and q' are both ready and maximal failure equivalent at the same time. None of the processes above are bisimilar: the corresponding states reached via transitions labelled a (with total probability 1) display different behaviour as they either have different branching structure, or can trigger different actions.
Orthogonally, as previously stated, in this chapter we also focus on providing a coalgebraic modelling of must and may testing semantics for divergent LTS's.
Intuitively, in the setting of testing semantics, fixed a set of tests, two systems are deemed to be equivalent if they pass exactly the same tests. With concurrent non-deterministic

processes, a system may pass a test in some, but not all, its executions. This leads to the definitions of *may testing* (a system may pass a test in some execution) and *must testing* (a system must pass a test in all its executions).
However, alternative trace-based characterisations of must and may testing were provided in [CH89, DH84, Hen88]. Intuitively, must testing preorder abstracts from infinite internal computations. It relates two processes p and q only if, for each trace w, whenever p does not engage in divergent behaviour in its attempt to execute w, then so does q. Moreover, q has to be "less non-deterministic" than p – a property established based on the inclusion of the acceptance (ready) sets associated with q and p, respectively. Two processes are must equivalent whenever the must preorder relates them in both directions. May testing preorder (respectively, equivalence) coincides with the usual language inclusion (respectively, equality).
Consider for an example the following two systems, where τ is used to represent an internal computation step:



Processes p and q cannot be related in terms of the must testing semantics. On the one hand, q does not diverge with respect to action d, whereas p diverges. On the other hand, p is less non-deterministic than q, as the ready set $\{\{b,c\}\}$ of p after performing action a is not included in the ready set $\{\{b\},\{c\}\}$ of q. However, p and q are may testing equivalent as they both execute the same sets of (visible) traces $\{\varepsilon, a, d, ab, ac\}$.
In this chapter we show how decorated trace, must and may testing semantics can be recovered in a coalgebraic setting by employing the generalised powerset construction in [SBBR13]. The derived coalgebraic characterisations leads to canonical representatives in terms of final Moore automata which further enabled reasoning by constructing bisimulations witnessing the desired notion of behavioural equivalence/preorder. Moreover, as we also saw in the previous chapter, this result is interesting from the point of view of tool development as well: construction of bisimulations is known to be particularly suitable for automation.
It is also interesting to observe that the spectrum of decorated trace semantics in Figure 4.1 can be recovered from our coalgebraic modelling. The procedure is briefly summarised in Section 4.5, for the case of failure and complete trace semantics for LTS's, and ready and trace semantics for GPS's, respectively.
Organisation of the chapter. In Section 4.1 and Section 4.2, we show how the powerset construction can be applied for determinising LTS's and GPS's, respectively, in terms of Moore automata $(X, f : X \to B \times X^A)$, in order to coalgebraically characterise the corresponding decorated trace semantics. Here we also prove that the obtained coalgebraic models are equivalent to the original definitions, and illustrate how one can reason about decorated trace equivalence by constructing (Moore) bisimulations. A compact overview on the uniform coalgebraic framework is given in Section 4.3. Section 4.4 discusses that the canonical representatives of LTS's and GPS's we obtain coalgebraically coincide with the corresponding minimal automata one would obtain by identifying all states equivalent with respect to a particular decorated trace semantics. In Section 4.5 we show that the

spectrum of decorated trace semantics can be obtained from the coalgebraic modelling. A coalgebraic modelling of may and must testing semantics, respectively, is provided in Section 4.6 by exploiting extensions of trace and failure semantics, respectively, to the context of LTS's with internal behaviour. Finally, Section 4.7 contains a summary of the results in this chapter.

4.1 Decorated trace semantics of LTS's

In this section, our aim is to provide a coalgebraic view on decorated trace equivalences of labelled transition systems (LTS's). We use the generalised powerset construction and show how one can determinise arbitrary LTS's obtaining particular instances of Moore automata (with different output sets) in order to model ready, failure, (complete) trace, possible-futures, ready trace and failure trace equivalences. This paves the way to building a general framework for reasoning on decorated trace equivalences in a uniform fashion, in terms of bisimulations (up-to context).

Note that our results are derived in the context of image finite LTS's, in accordance with the setting proposed in [vG01a]. An LTS is a pair (X,δ) where X is a set of states and $\delta : X \to (\mathscr{P}_\omega X)^A$ is a function assigning to each state $x \in X$ and to each label $a \in A$ a finite set of possible successors states. We write $x \xrightarrow{a} y$ whenever $y \in \delta(x)(a)$. We extend the notion of transition to words $w = a_1 \ldots a_n \in A^*$ as follows: $x \xrightarrow{w} y$ if and only if $x \xrightarrow{a_1} \ldots \xrightarrow{a_n} y$. For $w = \varepsilon$, we have $x \xrightarrow{\varepsilon} y$ if and only if $y = x$.

The coalgebraic characterisation of ready, failure and (complete) trace was originally obtained in [SBBR13]. We recall it here, with a slight adaptation which will be useful for the generalisations we will explore. Given an arbitrary LTS

$$(X, \delta : X \to (\mathscr{P}_\omega X)^A),$$

one constructs a *decorated* LTS, which is a coalgebra of the functor $\mathscr{F}_{\mathscr{I}}(X) = B_{\mathscr{I}} \times (\mathscr{P}_\omega X)^A$. More precisely, we construct

$$(X, \langle \overline{o}_{\mathscr{I}}, \delta \rangle : X \to B_{\mathscr{I}} \times (\mathscr{P}_\omega X)^A),$$

where the output operation

$$\overline{o}_{\mathscr{I}} : X \to B_{\mathscr{I}}$$

provides the observations of interest (the decorations) corresponding to the original LTS and depending on the equivalence ($\mathscr{I}$) we want to study. Note that both the output operation and its codomain are parameterised by $\mathscr{I}$.

Then, the decorated LTS is determinised as depicted in Figure 4.2, according to the powerset construction summarised in diagram (2.7) in Section 2.3. Recall that the generalised powerset construction is applied in the framework of coalgebras $f : X \to \mathscr{F}T(X)$ for a functor $\mathscr{F}$ and a monad T, with $\mathscr{F}T(X)$ a T-algebra. Intuitively, monads are used to hide computational effects such as non-determinism, whereas the requirement that $\mathscr{F}T(X)$ is an algebra for T guarantees the unique extension of f to a T-algebra homomorphism $f^\sharp$ representing a new coalgebra with state space hiding the computational effects. Consequently, this extension enables reasoning on $\mathscr{F}$-equivalence in the coalgebra $f^\sharp$, rather than reasoning on the (finer) $\mathscr{F}T$-equivalence in the coalgebra f.

For the case of decorated LTS's, we instantiate T with the powerset monad $(\mathscr{P}_\omega, \eta, \mu)$ such that $\eta(x) = \{x\}$ and $\mu(U) = \bigcup_{S_i \in U} S_i$, and $\mathscr{F}$ with $\mathscr{F}_{\mathscr{I}} = B_{\mathscr{I}} \times (\mathscr{P}_\omega(-))^A$. Moreover, $\mathscr{F}T(X)$

carries a T-algebra structure, that is a semilattice, as $\mathscr{P}(X)$ and $B_{\mathscr{I}}$ are a semilattices (as we shall see later, for each of the semantics $\mathscr{I}$), and product and exponentiation preserve the algebra structure. We will see that this extension enables shifting from reasoning on bisimilarity of decorated LTS's to reasoning on the (coarser) language (trace) equivalence. Note that the semilattice structures ensure the existence of least upper bounds, which further enable the definition of $f^\sharp = \langle o, t\rangle$ and $[\![-]\!]$ as semilattice morphisms. In Figure 4.2 we use $\bigsqcup$ to denote both the operation of $B_{\mathscr{I}}$ and the union of subsets in $\mathscr{P}_\omega X$.

$$\begin{tikzcd}
X \arrow[r, "\{-\}"] \arrow[d, "\langle \overline{o}_{\mathscr{I}}, \delta\rangle"'] & \mathscr{P}_\omega X \arrow[r, dashed, "{[\![-]\!]}"] \arrow[dl, "\langle o, t\rangle"] & (B_{\mathscr{I}})^{A^*} \arrow[d, "\langle \epsilon, (-)_a\rangle"] \\
\mathscr{F}_{\mathscr{I}} X = B_{\mathscr{I}} \times (\mathscr{P}_\omega X)^A \arrow[rr, dashed, "id_{B_{\mathscr{I}}} \times {[\![-]\!]}^A"'] & & B_{\mathscr{I}} \times ((B_{\mathscr{I}})^{A^*})^A
\end{tikzcd}$$

$$o(Y) = \textstyle\bigsqcup_{y \in Y} \overline{o}_{\mathscr{I}}(y) \qquad [\![Y]\!](\varepsilon) = o(Y)$$
$$t(Y)(a) = \textstyle\bigsqcup_{y \in Y} \delta(y)(a) \qquad [\![Y]\!](aw) = [\![\bigsqcup_{y \in Y} \delta(y)(a)]\!](w)$$

Figure 4.2: The powerset construction for decorated LTS's.

The coalgebraic modelling of possible-futures semantics could easily be recovered by following a similar approach. However, for the case of ready and failure trace semantics the transition structure of the LTS also needs to be slightly modified before the determinisation. This consists in changing the alphabet A to include additional information represented by sets of actions ready to be triggered as a first step. Consequently, to each LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ a unique coalgebra $(X, \langle \overline{o}_{\mathscr{I}}, \bar{\delta} : X \to (\mathscr{P}_\omega X)^{\bar{A}}\rangle)$ is associated, defined in a natural fashion, as we will present later on. The construction in Figure 4.2 is then applied on $(X, \langle \overline{o}_{\mathscr{I}}, \bar{\delta}\rangle)$.

The explicit instantiations of $\overline{o}_{\mathscr{I}}$ and $B_{\mathscr{I}}$ are provided later in this section, where we will also show that the coalgebraic modelling in fact coincides with the original definitions of the corresponding equivalences. This was not formally shown in [SBBR13], for any of the aforementioned semantics.

The coalgebraic modelling of decorated trace semantics enables the definition of the corresponding equivalences as Moore bisimulations [Rut00] (*i.e.*, bisimulations for a functor $\mathscr{M} = B_{\mathscr{I}} \times X^A$). This way, checking behavioural equivalence of x_1 and x_2 reduces to checking the equality of their unique representatives in the final coalgebra: $[\![\{x_1\}]\!]$ and $[\![\{x_2\}]\!]$.

In the subsequent sections we a) prove the details on the coalgebraic modelling of ready, failure, (complete) trace, possible-futures, ready trace and failure trace semantics, b) show that the corresponding representations coincide with their original definitions in [vG01a] and c) demonstrate, by means of examples, how the associated coalgebraic framework can be used in order to reason on (some of) the aforementioned equivalences in terms of Moore bisimulations.

4.1.1 Ready and failure semantics

In this section we show how the ingredients of Figure 4.2 can be instantiated in order to provide a coalgebraic modelling of ready and failure semantics. We also prove that the resulting coalgebraic characterisations of these semantics are equivalent to their original definitions in [vG01a]. Moreover, we provide an optimisation that can be used when reasoning on failure equivalence, based on the isomorphism of downsets and antichains.
Consider an LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ and define, for a function $\varphi : A \to \mathscr{P}_\omega X$, the set of *actions enabled by* φ:

$$I(\varphi) = \{a \in A \mid \varphi(a) \neq \emptyset\}, \tag{4.1}$$

and the set of *actions* φ *fails to enable*:

$$Fail(\varphi) = \{Z \subseteq A \mid Z \cap I(\varphi) = \emptyset\}.$$

For the particular case $\varphi = \delta(x)$, $I(\delta(x))$ denotes the set of all (initial) actions ready to be fired by $x \in X$, and $Fail(\delta(x))$ represents the set of subsets of all (initial) actions that cannot be triggered by such x.
A *ready pair* of x is a pair $(w, Z) \in A^* \times \mathscr{P}_\omega A$ such that $x \xrightarrow{w} y$ and $Z = I(\delta(y))$. A *failure pair* of x is a pair $(w, Z) \in A^* \times \mathscr{P}_\omega A$ such that $x \xrightarrow{w} y$ and $Z \in Fail(\delta(y))$. We denote by $\mathscr{R}(x)$ and $\mathscr{F}(x)$, respectively, the sets of *all ready pairs* and *failure pairs*, respectively, associated with x.
Intuitively, ready semantics identifies states in X based on the actions $a \in A$ they can immediately trigger after performing a certain action sequence $w \in A^*$, i.e., based on their ready pairs. It was originally defined as follows:

4.1.1 Definition (Ready equivalence [OH86, vG01a]). Let $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ be an LTS and $x, y \in X$ two states. States x and y are *ready equivalent* ($\mathscr{R}$-equivalent) if and only if they have the same set of ready pairs, that is $\mathscr{R}(x) = \mathscr{R}(y)$, where

$$\mathscr{R}(x) = \{(w, Z) \in A^* \times \mathscr{P}_\omega A \mid \exists x' \in X. x \xrightarrow{w} x' \wedge Z = I(\delta(x'))\}. \qquad \clubsuit$$

Failure semantics identifies behaviours of states in X according to their failure pairs.

4.1.2 Definition (Failure equivalence [vG01a]). Let $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ be an LTS and $x, y \in X$ two states. States x and y are *failure equivalent* ($\mathscr{F}$-equivalent) if and only if $\mathscr{F}(x) = \mathscr{F}(y)$, where

$$\mathscr{F}(x) = \{(w, Z) \in A^* \times \mathscr{P}_\omega A \mid \exists x' \in X. x \xrightarrow{w} x' \wedge Z \in Fail(\delta(x'))\}. \qquad \clubsuit$$

The coalgebraic modelling of ready, respectively, failure semantics is obtained in a uniform fashion, by instantiating the ingredients of Figure 4.2 as follows. For $\mathscr{I} \in \{\mathscr{R}, \mathscr{F}\}$, $\overline{o}_{\mathscr{I}} : X \to \mathscr{P}_\omega(\mathscr{P}_\omega A)$ is defined as:

$$\overline{o}_{\mathscr{R}}(x) = \{I(\delta(x))\} \qquad \overline{o}_{\mathscr{F}}(x) = Fail(\delta(x)).$$

Intuitively, in the setting of ready semantics, the observations provided by the output operation refer to the sets of actions ready to be executed by the states of the LTS. Similarly, for failure semantics, the output operation refers to the sets of actions the states of the LTS cannot immediately fire.

Remark 11 *Observe that the codomain of $\overline{o}_{\mathscr{R}}$ is $\mathscr{P}_\omega(\mathscr{P}_\omega A)$, and not $\mathscr{P}_\omega A$, as one might expect. However, this is consistent with the intended semantics. For $B_{\mathscr{I}} = B_{\mathscr{R}} = B_{\mathscr{F}} = \mathscr{P}_\omega(\mathscr{P}_\omega A)$, the final Moore coalgebra has carrier $(\mathscr{P}_\omega(\mathscr{P}_\omega A))^{A^*}$ which is isomorphic to $\mathscr{P}(A^* \times \mathscr{P}_\omega(A))$ the type of $\mathscr{R}(x)$ and $\mathscr{F}(x)$. The unique homomorphism into the final coalgebra will associate to each state $\{x\}$ a function that for each $w \in A^*$ returns a set containing all sets $R_{x'}$ of ready (resp. failed) actions triggered by all x' such that $x \xrightarrow{w} x'$, for $x, x' \in X$.*

Next, we will prove the equivalence between the coalgebraic modelling of ready and failure semantics and their original definitions, presented above. More explicitly, given an arbitrary LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ and a state $x \in X$, we want to show that $[\![\{x\}]\!]$ is equal to $\mathscr{I}(x)$, for $\mathscr{I} \in \{\mathscr{R}, \mathscr{F}\}$, depending on the semantics of interest. However, note that the definition of $[\![-]\!]$ is independent of $\mathscr{I}$; the difference is (implicitly) made by the output function $\overline{o}_{\mathscr{I}}$.
The behaviour of a state $x \in X$ is a function $[\![\{x\}]\!] : A^* \to \mathscr{P}_\omega(\mathscr{P}_\omega A)$, whereas $\mathscr{I}(x)$ is defined as a set of pairs in $A^* \times \mathscr{P}_\omega A$. We represent the set $\mathscr{I}(x) \in \mathscr{P}(A^* \times \mathscr{P}_\omega A)$ by a function $\varphi_x^{\mathscr{I}} : \mathscr{P}_\omega(\mathscr{P}_\omega A)^{A^*}$, where, for $w \in A^*$,

$$\begin{aligned}\varphi_x^{\mathscr{R}}(w) &= \{I(\delta(y)) \mid x \xrightarrow{w} y\}\\ \varphi_x^{\mathscr{F}}(w) &= \{Z \subseteq A \mid x \xrightarrow{w} y \wedge Z \in Fail(\delta(y))\}.\end{aligned}$$

Showing the equivalence between the coalgebraic and the original definitions of ready, respectively, failure semantics reduces to proving that

$$(\forall x \in X).\, [\![\{x\}]\!] = \varphi_x^{\mathscr{I}}. \tag{4.2}$$

4.1.3 Theorem. *Let $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ be an LTS. Then for all $x \in X$, $w \in A^*$, and $\mathscr{I} \in \{\mathscr{R}, \mathscr{F}\}$, $[\![\{x\}]\!](w) = \varphi_x^{\mathscr{I}}(w)$.*

Proof. For $\mathscr{I}$ ranging over $\{\mathscr{R}, \mathscr{F}\}$, the proof is by induction on words $w \in A^*$. We provide the details for the case of ready semantics. A similar reasoning can be applied for failure semantics.

- *Base case.* $w = \varepsilon$. We have:

$$\begin{aligned}[\![\{x\}]\!](\varepsilon) &= o(\{x\}) = \overline{o}_{\mathscr{I}}(x) = \{I(\delta(x))\}\\ \varphi_x^{\mathscr{R}}(\varepsilon) &= \{I(\delta(y)) \mid x \xrightarrow{\varepsilon} y\} = \{I(\delta(x))\}\end{aligned}$$

- *I*nduction step. Consider $w \in A^*$ and assume, for all $x \in X$, $[\![\{x\}]\!](w) = \varphi_x^{\mathscr{R}}(w)$. We want to prove that $[\![\{x\}]\!](aw) = \varphi_x^{\mathscr{R}}(aw)$, where $a \in A$.

$$\begin{aligned}[\![\{x\}]\!](aw) &= [\![\delta(x)(a)]\!](w) = [\![t(\{x\})(a)]\!](w)\\ &= \bigcup_{x \xrightarrow{a} z} [\![\{z\}]\!](w) \overset{\text{IH}}{=} \bigcup_{x \xrightarrow{a} z} \varphi_z^{\mathscr{R}}(w)\\ \varphi_x^{\mathscr{R}}(aw) &= \{I(\delta(y)) \mid x \xrightarrow{aw} y\}\\ &= \{I(\delta(y)) \mid x \xrightarrow{a} z \wedge z \xrightarrow{w} y\}\\ &= \bigcup_{x \xrightarrow{a} z} \{I(\delta(y)) \mid z \xrightarrow{w} y\}\\ &= \bigcup_{x \xrightarrow{a} z} \varphi_z^{\mathscr{R}}(w)\end{aligned}$$

4.1.4 Example. In what follows we illustrate the equivalence between the coalgebraic and the original definitions of ready semantics by means of an example. Consider the following LTS.

$$\begin{tikzcd}
 & & p_0 \arrow[loop right, "a"] \arrow[d, "a"] & & \\
p_4 & p_2 \arrow[l, "c"'] & p_1 \arrow[l, "b"'] \arrow[r, "b"] & p_3 \arrow[r, "d"] & p_5
\end{tikzcd}$$

We write a^n to represent the action sequence $aa\dots a$ of length $n \geq 1$, with $n \in \mathbb{N}$. The set $\mathcal{R}(p_0)$ of all ready pairs associated with p_0 is:

$$\{(\varepsilon,\{a\}),(a^n,\{a\}),(a^n,\{b\}),(a^nb,\{c\}),(a^nb,\{d\}),(a^nbc,\emptyset),(a^nbd,\emptyset) \mid n \geq 1\}.$$

We can construct a Moore automaton, for $S = \{p_0, p_1, \dots, p_5\}$,

$$(\mathcal{P}_\omega S, \langle o,t\rangle \colon \mathcal{P}_\omega S \to \mathcal{P}_\omega(\mathcal{P}_\omega A) \times (\mathcal{P}_\omega S)^A)$$

by applying the generalised powerset construction on the LTS above. The automaton will have $2^6 = 64$ states. We depict the accessible part from state $\{p_0\}$, where the output sets are indicated by double arrows: The output sets of a state Y of the Moore automaton in

$$\begin{tikzcd}
 & & \{p_0\} \arrow[r, Rightarrow] \arrow[d, "a"] & \{\{a\}\} & & \\
 & & \{p_0,p_1\} \arrow[r, Rightarrow] \arrow[d, "b"] & \{\{a\},\{b\}\} & & \\
\{\emptyset\} & \{p_4\} \arrow[l, Rightarrow] & \{p_2,p_3\} \arrow[l, "c"'] \arrow[u, bend left, "a"] \arrow[r, Rightarrow] \arrow[rr, bend right, "d"'] & \{\{c\},\{d\}\} & \{p_5\} \arrow[r, Rightarrow] & \{\emptyset\}
\end{tikzcd}$$

Figure 4.3: Ready determinisation when starting from $\{p_0\}$.

Figure 4.3 is the set of actions associated with a certain state $y \in Y$ which can immediately be performed. For example, process p_0 in the original LTS above is ready to perform action a, whereas p_1 can immediately perform b. Therefore it holds that $o(\{p_0\}) = \{\{a\}\}$ and $o(\{p_0,p_1\}) = \{\{a\},\{b\}\}$.
By simply looking at the automaton in Figure 4.3, one can easily see that the set of action sequences $w \in A^*$ the state $\{p_0\}$ can execute, together with the corresponding possible next actions equals $\mathcal{R}(p_0)$. Therefore, the automaton generated according to the generalised powerset construction captures the set of all ready pairs of the initial LTS. ♠

4.1.5 Example. The last example considered in this section shows how the coalgebraic framework can be applied in order to reason on failure equivalence of LTS's. (Checking ready equivalence follows a similar approach.) Consider the following two systems.

$$\begin{tikzcd}
p_1 & & p_0 \arrow[ll, "b"'] \arrow[rr, "c"] \arrow[loop above, "a"] \arrow[dl, "a"'] \arrow[dr, "a"] & & p_2 \\
 & p_3 \arrow[loop left, "a"] \arrow[dl, "b"'] \arrow[d, "c"] & & p_4 \arrow[loop right, "a"] \arrow[d, "c"'] \arrow[dr, "f"] & \\
p_5 & p_6 \arrow[d, "d"] & & p_7 \arrow[d, "e"'] & p_8 \\
 & p_9 & & p_{10} &
\end{tikzcd}
\qquad
\begin{tikzcd}
q_1 & & q_0 \arrow[ll, "b"'] \arrow[rr, "c"] \arrow[loop above, "a"] \arrow[dl, shift right, "a"'] \arrow[dr, shift left, "a"] & & q_2 \\
 & q_3 \arrow[ur, shift right, "a"'] \arrow[dl, "b"'] \arrow[d, "c"] & & q_4 \arrow[ul, shift left, "a"] \arrow[d, "c"'] \arrow[dr, "f"] & \\
q_5 & q_6 \arrow[d, "e"] & & q_7 \arrow[d, "d"'] & q_8 \\
 & q_9 & & q_{10} &
\end{tikzcd}$$

Let $Z = \{a_1, a_2, \ldots, a_n\}$ be the set of actions a process fails executing as a first step. For the simplicity of notation, we write $[a_1 a_2 \ldots a_n]$ to denote the set of all non-empty subsets $Z' \subseteq Z$. For example, if $Z = \{a_1, a_2\}$, then $[a_1 a_2]$ stands for $\{\{a_1\}, \{a_2\}, \{a_1, a_2\}\}$.
Note that p_0 and q_0 are $\mathscr{F}$-equivalent, according to Definition 4.1.2, as they have the same sets of failure pairs $\mathscr{F}(p_0)$ and $\mathscr{F}(q_0)$, respectively, equal to:

$$\begin{aligned}&\{(\varepsilon, [def]), (b, [abcdef]), (c, [abcdef])\} \cup \{(a^n, [def]), (a^n, [bde]),\\ &(a^n b, [abcdef]), (a^n c, [abcdef]), (a^n c, [abcef]), (a^n c, [abcdf]),\\ &(a^n f, [abcdef]), (a^n cd, [abcdef]), (a^n ce, [abcdef]) \mid n \in \mathbb{N}, n \geq 1\}.\end{aligned}$$

The same conclusion can be reached by checking behavioural equivalence of the two Moore automata generated according to the powerset construction, starting with $\{p_0\}$ and $\{q_0\}$. The fragments of the two automata starting from the states $\{p_0\}$ and $\{q_0\}$ are depicted in Figure 4.4 at page 58. The states $\{p_0\}$ and $\{q_0\}$ are Moore bisimilar, since their corresponding automata are isomorphic.

$$\begin{tikzcd}
 & \{p_0\} \arrow[r, Rightarrow] \arrow[dl, "b"'] \arrow[d, "a"'] \arrow[dr, "c"] & {[def]} \\
\{p_1\} \arrow[d, Rightarrow] & \{p_0, p_3, p_4\} \arrow[loop above, "a"] \arrow[d, Rightarrow] \arrow[ddl, bend right, "b"] \arrow[dd, bend left] \arrow[ddr, "f"] & \{p_2\} \arrow[d, Rightarrow] \\
{[abcdef]} & {[def] \cup [bde]} & {[abcdef]} \\
\{p_1, p_5\} \arrow[d, Rightarrow] & \{p_2, p_6, p_7\} \arrow[d, Rightarrow] \arrow[ddl, bend right, "d"] \arrow[ddr, "e"] & \{p_8\} \arrow[d, Rightarrow] \\
{[abcdef]} & {[abcdef] \cup [abcef] \cup [abcdf]} & {[abcdef]} \\
\{p_9\} \arrow[d, Rightarrow] & & \{p_{10}\} \arrow[d, Rightarrow] \\
{[abcdef]} & & {[abcdef]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \{q_0\} \arrow[r, Rightarrow] \arrow[dl, "b"'] \arrow[d, "a"'] \arrow[dr, "c"] & {[def]} \\
\{q_1\} \arrow[d, Rightarrow] & \{q_0, q_3, q_4\} \arrow[loop above, "a"] \arrow[d, Rightarrow] \arrow[ddl, bend right, "b"] \arrow[dd, bend left] \arrow[ddr, "f"] & \{q_2\} \arrow[d, Rightarrow] \\
{[abcdef]} & {[def] \cup [bde]} & {[abcdef]} \\
\{q_1, q_5\} \arrow[d, Rightarrow] & \{q_2, q_6, q_7\} \arrow[d, Rightarrow] \arrow[ddl, bend right, "d"] \arrow[ddr, "e"] & \{q_8\} \arrow[d, Rightarrow] \\
{[abcdef]} & {[abcdef] \cup [abcef] \cup [abcdf]} & {[abcdef]} \\
\{q_9\} \arrow[d, Rightarrow] & & \{q_{10}\} \arrow[d, Rightarrow] \\
{[abcdef]} & & {[abcdef]}
\end{tikzcd}$$

Figure 4.4: Failure determinisation when starting from $\{p_0\}$ and $\{q_0\}$.

optimisation for failure semantics. In this section we showed how failure semantics can be modelled in a coalgebraic setting, by employing the generalised powerset construction. More explicitly, given a state p of an LTS $(S, \delta : S \to (\mathscr{P}_\omega S)^A)$, we showed how to build a (final) Moore coalgebra $(\mathscr{P}_\omega S, \langle o, t\rangle : \mathscr{P}_\omega S \to (\mathscr{P}_\omega(\mathscr{P}_\omega))^{A^*})$ "capturing" the corresponding set of failure pairs $\mathscr{F}(p)$, hence enabling reasoning on failure equivalence in terms of Moore bisimulations.
An optimised, equivalent modelling of failure semantics can be provided by exploiting the standard isomorphism between downsets and antichains. As we shall see, this enables reasoning on the corresponding equivalence more effectively, based on bisimulations of Moore automata with "smaller" output sets consisting of ready actions.
A *downset* in $\mathscr{P}_\omega(A)$ is a set $D \subseteq \mathscr{P}_\omega(A)$ such that if $Z \in D$ and $Z' \subseteq Z$ then $Z' \in D$. We use $\overline{\mathscr{D}}(\mathscr{P}_\omega(A))$ denote the set of downsets of $\mathscr{P}_\omega(A)$. Note that we can define a semilattice $(\overline{\mathscr{D}}(\mathscr{P}_\omega(A)), \sqcup, 0)$ by taking $\sqcup$ as being the union and 0 as the empty set.
An *antichain* on $\mathscr{P}_\omega(A)$ is a set $I \subseteq \mathscr{P}_\omega(A)$ such that if $Z \in I$ then there exists no $Z' \in I$ such that $Z' \subset Z$. We use $\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ denote the set of antichains of $\mathscr{P}_\omega(A)$. Note that the union of antichains is not necessarily an antichain. However, we can define a semilattice on $\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ by taking the $\sqcup$ defined as $I_1 \sqcup I_2 = min(I_1 \cup I_2)$ where

$$min(I) = \{Z \in I \mid (\nexists Z' \in I). Z' \subset Z\}. \tag{4.3}$$

Now consider the homomorphisms $i : \overline{\mathscr{D}}(\mathscr{P}_\omega(A)) \to \overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ defined as

$$i(F) = min(\cup_{F_i \in F}\{A - F_i\}) \tag{4.4}$$

and $j : \overline{\mathscr{A}}(\mathscr{P}_\omega(A)) \to \overline{\mathscr{D}}(\mathscr{P}_\omega(A))$ defined as

$$j(I) = \downarrow(\cup_{I_i \in I}\{A - I_i\}), \tag{4.5}$$

where $\downarrow S$ denotes the downward closure of a set S. It is easy to see that one homomorphism is the inverse of the other and thus the semilattices $\overline{\mathscr{D}}(\mathscr{P}_\omega(A))$ and $\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ are isomorphic.
At this point, it is worth to observe that for all $X \in \mathscr{P}_\omega(S)$, the Moore output function $o(X)$ is a downset (since $\overline{o}_{\mathscr{F}}(x)$ is a downset for all x, and since the union of downset is a downset). Therefore we can safely restrict the codomain of $o : \mathscr{P}_\omega(S) \to \mathscr{P}_\omega(\mathscr{P}_\omega(A))$, to $o : \mathscr{P}_\omega(S) \to \overline{\mathscr{D}}(\mathscr{P}_\omega(A))$. By exploiting the isomorphism discussed above, we can instead define the function $o_1 : \mathscr{P}_\omega(S) \to \overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ as follows: for all $X \in \mathscr{P}_\omega(S)$

$$o_1(X) = \begin{cases} \{I(\delta(x))\} & \text{if } X = \{x\} \text{ with } x \in S \\ 0 & \text{if } X = 0 \\ min(o_1(X_1) \sqcup o_1(X_2)) & \text{if } X = X_1 \sqcup X_2 \end{cases}$$

4.1.6 Proposition. *For all $X, Y \in \mathscr{P}_\omega(S)$, $o(X) = o(Y)$ iff $o_1(X) = o_1(Y)$.*

Proof. The proof follows from the fact that $o_1 = i \circ o$ and that $i : \overline{\mathscr{D}}(\mathscr{P}_\omega(A)) \to \overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ and $j : \overline{\mathscr{A}}(\mathscr{P}_\omega(A)) \to \overline{\mathscr{D}}(\mathscr{P}_\omega(A))$ are isomorphic. □

This optimisation can be applied also for the case of failure trace semantics in Section 4.1.4. Moreover, as presented in Section 4.6.2, the isomorphism of downsets and antichains is used for the coalgebraic modelling of must testing semantics.

4.1.2 (Complete) trace semantics

In this section we model coalgebraically trace and complete trace semantics. Similar to the previous section, we also show that the corresponding coalgebraic representations of these semantics are equivalent to their original definitions.
Consider an LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$. Trace semantics identifies states in X according to the set of words $w \in A^*$ they can execute, whereas complete trace semantics identifies states $x \in X$ based on their set of complete traces. A trace $w \in A^*$ of x is complete if and only if x can perform w and reach a deadlock state y or, equivalently,

$$(\exists y \in X) . x \xrightarrow{w} y \wedge I(\delta(y)) = \emptyset.$$

The difference between trace and complete semantics is that the latter enables an external observer to detect stagnation, or deadlock states of a system.
Formally, trace and complete trace equivalences are defined as follows.

4.1.7 Definition (Trace equivalence [Hoa78, vG01a]). Let $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ be an LTS and $x, y \in X$ two states. States x and y are *trace equivalent* ($\mathscr{T}$-equivalent) if and only if $\mathscr{T}(x) = \mathscr{T}(y)$, where

$$\mathscr{T}(x) = \{w \in A^* \mid \exists x' \in X . x \xrightarrow{w} x'\}. \tag{4.6}$$

4.1.8 Definition (Complete trace equivalence [AFV99]). Consider an LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ and $x, y \in X$ two states. States x and y are *complete trace equivalent* ($\mathscr{C}\mathscr{T}$-equivalent) if and only if $\mathscr{C}\mathscr{T}(x) = \mathscr{C}\mathscr{T}(y)$, where

$$\mathscr{C}\mathscr{T}(x) = \{w \in A^* \mid \exists x' \in X . x \xrightarrow{w} x' \wedge I(\delta(x')) = \emptyset\}. \qquad \clubsuit$$

In what follows we instantiate the constituents of Figure 4.2 in order to provide the associated coalgebraic modellings.
For $\mathscr{I} \in \{\mathscr{T}, \mathscr{C}\mathscr{T}\}$, the output function $\overline{o}_{\mathscr{I}} : X \to 2$ is:

$$\overline{o}_{\mathscr{T}}(x) = 1 \qquad \overline{o}_{\mathscr{C}\mathscr{T}}(x) = \begin{cases} 1 & \text{if } I(\delta(x)) = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Note that, for trace semantics, one does not distinguish between traces and complete traces. Intuitively, all states are accepting, so they have the same observable behaviour (i.e., $\overline{o}_{\mathscr{T}}(\varphi) = 1$), no matter the transitions they perform. On the other hand, complete trace semantics distinguishes between deadlock states and states that can still execute actions $a \in A$.
Consider, for example, the following LTS:

$$\begin{tikzcd} p_1 & p_0 \arrow[l, "a"'] \arrow[r, bend left, "a"] & p_2 \arrow[l, bend left, "b"] \end{tikzcd}$$

Observe that, for each $n \geq 0$, $(ab)^n a$ is a complete trace of p_0, as

$$p_0 \xrightarrow{a} p_2 \xrightarrow{b} p_0 \xrightarrow{a} p_2 \xrightarrow{b} \ldots \xrightarrow{b} p_0 \xrightarrow{a} p_1 \tag{4.7}$$

where p_1 cannot perform any further action.

The above behaviour, described in terms of transitions between states of the Moore automaton derived according to the generalised powerset construction, can be depicted as follows:

$$\{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{b} \{p_0\} \xrightarrow{a} \{p_1,p_2\} \xrightarrow{b} \ldots \xrightarrow{b} \{p_0\} \xrightarrow{a} \{p_1,p_2\}$$

where p_1 is a deadlock state and p_2 is not.
Intuitively, for $n \geq 0$, we can state that $(ab)^n a$ is a complete trace of $\{p_0\}$, as the deadlock state $p_2 \in \{p_1,p_2\}$ can be reached from $\{p_0\}$ by performing $(ab)^n a$ (see (4.7)).
Therefore, given $Y_1, Y_2 \subseteq X$ and $w \in A^*$ such that $Y_1 \xrightarrow{w} Y_2$, we observe that w is a complete trace of Y_1 whenever there exists a deadlock state $y \in Y_2$. Otherwise, w is not a complete trace of Y_1.
In the coalgebraic modelling, the above observations with respect to the (non)stagnating states appear in the definition of the function $o\colon \mathscr{P}_\omega(X) \to 2$. Note that, for example, $o(\{p_1,p_2\}) = 1$ and $o(\{p_0\}) = 0$ for the case of complete trace equivalence, as p_1 is a deadlock state and p_0 is not. For trace semantics we have $o(\{p_1,p_2\}) = o(\{p_0\}) = 1$. Here, $B_{\mathscr{I}} = 2$ and the final Moore coalgebra in Figure 4.2 is the set of languages 2^{A^*} over A (and the transition structure $\langle \epsilon, (-)_a \rangle$ is simply given by Brzozowski derivatives). Therefore, we can state that the map into the final coalgebra associates to each state $Y \in \mathscr{P}_\omega X$ the set of all traces corresponding to states $y \in Y$, namely, the language:

$$L = \bigcup_{y \in Y} \{w \in A^* \mid (\exists y' \in X) \,.\, y \xrightarrow{w} y'\}.$$

The set $\mathscr{P}(A^*)$ is isomorphic to the set of functions 2^{A^*} which enables us to represent the set $\mathscr{I}(x)$ in terms its characteristic function $\varphi^{\mathscr{I}}_x : A^* \to 2$ defined, for $\mathscr{I} \in \{\mathscr{T}, \mathscr{C}\mathscr{T}\}$, $w \in A^*$, as follows:

$$\varphi^{\mathscr{T}}_x(w) = 1 \text{ if } \exists y \in X \,.\, x \xrightarrow{w} y \qquad \varphi^{\mathscr{C}\mathscr{T}}_x(w) = \begin{cases} 1 & \text{if } \exists y \in X \,.\, x \xrightarrow{w} y \wedge I(\delta(y)) = \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Proving the equivalence between the coalgebraic and the classic definition of (complete) trace semantics reduces to showing that

$$(\forall x \in X) \,.\, [\![\{x\}]\!] = \varphi^{\mathscr{I}}_x. \tag{4.8}$$

4.1.9 Theorem. *Let $(X, \delta\colon X \to (\mathscr{P}_\omega X)^A)$ be an LTS. Then for all $x \in X$ and $w \in A^*$, $[\![\{x\}]\!](w) = \varphi^{\mathscr{I}}_x(w)$.*

Proof. The proof is by induction on words $w \in A^*$ (similar to the proof of Theorem 4.1.3).□

4.1.10 Example. Consider the following two LTS's:

$$\begin{tikzcd} w_1 & w_0 \arrow[l, "a"'] \arrow[loop right, "a"] & & w'_0 \arrow[loop right, "a"] \end{tikzcd}$$

Observe that w_0 and w'_0 are trace equivalent (according to Definition 4.1.7), as they output the same sets of traces

$$\mathscr{T}(w_0) = \mathscr{T}(w'_0) = \{\varepsilon\} \cup \{a^n \mid n \in \mathbb{N}, n \geq 1\}$$ ♠

$$\begin{tikzcd} 0 & \{w_0\} \arrow[l, Leftarrow] \arrow[r, "a"] & \{w_0, w_1\} \arrow[r, Rightarrow] \arrow[loop below, "a"] & 1 & 0 & \{w_0'\} \arrow[l, Leftarrow] \arrow[loop right, "a"] \end{tikzcd}$$

Figure 4.5: Complete trace determinisation when starting from $\{w_0\}, \{w_0'\}$.

but they are not complete trace equivalent (according to Definition 4.1.8), as w_0' can never reach a deadlock state, whereas w_0 can reach the stagnating state w_1.
The complete trace determinisation contains the sub-automata starting from states $\{w_0\}$ and $\{w_0'\}$ depicted in Figure 4.5: States $\{w_0\}$ and $\{w_0'\}$ are not behaviourally equivalent, since $\{w_0, w_1\}$ outputs 1, whereas $\{w_0'\}$ never reaches a state with this output. Hence, as expected, we will never be able to build a bisimulation containing states $\{w_0\}$ and $\{w_0'\}$.
On the other hand, in the setting of trace semantics, the determinised (Moore) automata associated with w_0 and w_0', respectively, are similar to those depicted in Figure 4.5, with the difference that now all their states output value 1. This makes the aforementioned automata bisimilar, hence providing a "yes" answer with respect to $\mathscr{T}$-equivalence of w_0 and w_0', as anticipated.

4.1.3 Possible-futures semantics

In what follows we provide a coalgebraic modelling of possible-futures semantics and show that it coincides with the original definition in [vG01a]. We also give an example on how the generalised powerset construction and Moore bisimulations can be used in order to reason on possible-futures equivalence.
Let $(X, \delta: X \to (\mathscr{P}_\omega X)^A)$ be an LTS. A *possible future* of $x \in X$ is a pair $\langle w, T\rangle \in A^* \times \mathscr{P}(A^*)$ such that $x \xrightarrow{w} y$ and $T = \mathscr{T}(y)$ (where $\mathscr{T}(y)$ is the set of traces of y, as in Section 4.1.2). Possible-futures semantics identifies states that can trigger the same sets of traces $w \in A^*$ and moreover, by executing such w, they reach trace-equivalent states.

4.1.11 Definition (Possible-futures equivalence [RB81, vG01a]). Consider an LTS $(X, \delta: X \to (\mathscr{P}_\omega X)^A)$ and $x, y \in X$ two states. States x and y are *possible-futures equivalent* ($\mathscr{P}\mathscr{F}$-equivalent) if and only if $\mathscr{P}\mathscr{F}(x) = \mathscr{P}\mathscr{F}(y)$, where

$$\mathscr{P}\mathscr{F}(x) = \{\langle w, T\rangle \in A^* \times \mathscr{P}(A^*) \mid \exists x' \in X. x \xrightarrow{w} x' \wedge T = \mathscr{T}(x')\}. \qquad \clubsuit$$

The ingredients of Figure 4.2 are instantiated as follows.
The output function $\bar{o}_{\mathscr{P}\mathscr{F}}: X \to \mathscr{P}(\mathscr{P}A^*)$, which refers to the set of traces enabled by states $x \in X$ of the LTS, is defined as

$$\bar{o}_{\mathscr{P}\mathscr{F}}(x) = \{\mathscr{T}(x)\}.$$

Here, $B_{\mathscr{I}} = B_{\mathscr{P}\mathscr{F}} = \mathscr{P}(\mathscr{P}A^*)$ and the behaviour of a state $x \in X$ in the final coalgebra is given in terms of a function $[\![\{x\}]\!]: A^* \to \mathscr{P}(\mathscr{P}A^*)^{A^*}$, which, intuitively, for each $w \in A^*$ returns the set of sets T_y of traces corresponding to states $y \in X$ such that $x \xrightarrow{w} y$.
Next we want to show that for each $x \in X$, $[\![\{x\}]\!]$ and $\mathscr{P}\mathscr{F}(x)$ coincide.
First we choose to equivalently represent $\mathscr{P}\mathscr{F}(x) \in \mathscr{P}(A^* \times \mathscr{P}(A^*))$ – the set of all possible futures of a state $x \in X$ – in terms of $\varphi_x^{\mathscr{P}\mathscr{F}} \in (\mathscr{P}(\mathscr{P}A^*))^{A^*}$, where

$$\varphi_x^{\mathscr{P}\mathscr{F}}(w) = \{\mathscr{T}(y) \mid x \xrightarrow{w} y\},$$

Showing the equivalence between the coalgebraic and the original definition of possible-futures semantics reduces to proving that

$$(\forall x \in X).\, [\![\{x\}]\!] = \varphi_x^{\mathscr{P}\mathscr{F}}. \tag{4.9}$$

4.1.12 Theorem. *Let* $(X, \delta\colon X \to (\mathscr{P}_\omega X)^A)$ *be an LTS. Then for all* $x \in X$ *and* $w \in A^*$, $[\![\{x\}]\!](w) = \varphi_x^{\mathscr{P}\mathscr{F}}(w)$.

Proof. The proof is by induction on $w \in A^*$ (similar to the proof of Theorem 4.1.3). □

4.1.13 Example. Consider the following LTS's.

$$\begin{tikzcd}
 & & p_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & & \\
 & p_1 \arrow[dl, "b"'] \arrow[d, "a"] \arrow[dr, "a"] & & p_2 \arrow[d, "a"] \arrow[dr, "a"] & & \\
p_3 & p_4 \arrow[dl, "b"'] \arrow[d, "c"] & p_5 \arrow[d, "c"] & p_6 \arrow[d, "c"] & p_7 \arrow[d, "c"] \arrow[dr, "b"] & \\
p_8 & p_9 \arrow[d, "d"] & p_{10} \arrow[d, "e"] & p_{11} \arrow[d, "d"] & p_{12} \arrow[d, "e"] & p_{13} \\
 & p_{14} & p_{15} & p_{16} & p_{17} &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & & q_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & \\
 & & q_1 \arrow[dl, "a"'] \arrow[d, "a"] & & q_2 \arrow[dl, "a"'] \arrow[d, "a"] \arrow[dr, "b"] & \\
 & q_3 \arrow[dl, "b"'] \arrow[d, "c"] & q_4 \arrow[d, "c"] & q_5 \arrow[d, "c"] & q_6 \arrow[d, "c"] \arrow[dr, "b"] & q_7 \\
q_8 & q_9 \arrow[d, "d"] & q_{10} \arrow[d, "e"] & q_{11} \arrow[d, "d"] & q_{12} \arrow[d, "e"] & q_{13} \\
 & q_{14} & q_{15} & q_{16} & q_{17} &
\end{tikzcd}$$

Note that p_0 and q_0 are possible-futures equivalent, as the traces both can follow are sequences $w \in \{a, ab, aa, aab, aac, aacd, aace\}$ and moreover, by triggering the same w they reach states with equal sets of traces. The equivalence between p_0 and q_0 can be formally captured in terms of a bisimulation relation R on the associated Moore automata (generated according to the generalised powerset construction) depicted in Figure 4.6, where

$$\begin{aligned} R = \{&(\{p_0\},\{q_0\}), (\{p_1,p_2\},\{q_1,q_2\}), (\{p_3\},\{q_7\}), (\{p_8,p_{13}\},\{q_8,q_{13}\}),\\ &(\{p_5,p_5,p_6,p_7\},\{q_3,q_4,q_5,q_6\}), (\{p_9,p_{10},p_{11},p_{12}\},\{q_9,q_{10},q_{11},q_{12}\}),\\ &(\{p_{14},p_{16}\},\{q_{14},q_{16}\}), (\{p_{15},p_{17}\},\{q_{15},q_{17}\})\,\}. \end{aligned}$$

It is easy to check that R is a bisimulation, since both automata in Figure 4.6 are isomorphic. (Note that equality of the outputs – which are sets of traces – can be established using the framework introduced in Section 4.1.2.)

$$\begin{tikzcd}
 & & \{p_0\} \arrow[r, phantom, "\Rrightarrow"] \arrow[d, "a"'] & \{\mathscr{T}(p_0)\} \\
\{\emptyset\} & \{p_8, p_{13}\} \arrow[l, Lleftarrow] & \{p_1, p_2\} \arrow[r, phantom, "\Rrightarrow"] \arrow[dl, "a"'] \arrow[d, "b"'] & \{\mathscr{T}(p_1), \mathscr{T}(p_2)\} \\
o_1 & \{p_4, p_5, p_6, p_7\} \arrow[l, phantom, "\Lleftarrow"] \arrow[u, "b"'] \arrow[d, "c"'] & \{p_3\} \arrow[r, phantom, "\Rrightarrow"] & \{\emptyset\} \\
o_2 & \{p_9, p_{10}, p_{11}, p_{12}\} \arrow[l, phantom, "\Lleftarrow"] \arrow[d, "d"'] \arrow[dr, "e"] & & \\
\{\emptyset\} & \{p_{14}, p_{16}\} \arrow[l, phantom, "\Lleftarrow"] & \{p_{15}, p_{17}\} \arrow[r, phantom, "\Rrightarrow"] & \{\emptyset\}
\end{tikzcd}$$

$$\begin{tikzcd}
 & & \{q_0\} \arrow[r, phantom, "\Rrightarrow"] \arrow[d, "a"'] & \{\mathscr{T}(q_0)\} \\
\{\emptyset\} & \{q_8, q_{13}\} \arrow[l, Lleftarrow] & \{q_1, q_2\} \arrow[r, phantom, "\Rrightarrow"] \arrow[dl, "a"'] \arrow[d, "b"'] & \{\mathscr{T}(q_1), \mathscr{T}(q_2)\} \\
o'_1 & \{q_3, q_4, q_5, q_6\} \arrow[l, phantom, "\Lleftarrow"] \arrow[u, "b"'] \arrow[d, "c"'] & \{q_7\} \arrow[r, phantom, "\Rrightarrow"] & \{\emptyset\} \\
o'_2 & \{q_9, q_{10}, q_{11}, q_{12}\} \arrow[l, phantom, "\Lleftarrow"] \arrow[d, "d"'] \arrow[dr, "e"] & & \\
\{\emptyset\} & \{q_{14}, q_{16}\} \arrow[l, phantom, "\Lleftarrow"] & \{q_{15}, q_{17}\} \arrow[r, phantom, "\Rrightarrow"] & \{\emptyset\}
\end{tikzcd}$$

♠

Figure 4.6: Possible-futures determinisation when starting from $\{p_0\}, \{q_0\}$. $o_1 = \{\mathscr{T}(p_4), \mathscr{T}(p_5), \mathscr{T}(p_6), \mathscr{T}(p_7)\}, o_2 = \{\mathscr{T}(p_9), \mathscr{T}(p_{10}), \mathscr{T}(p_{11}), \mathscr{T}(p_{12})\}$, $o'_1 = \{\mathscr{T}(q_3), \mathscr{T}(q_4), \mathscr{T}(q_5), \mathscr{T}(q_6)\}$, $o'_2 = \{\mathscr{T}(q_9), \mathscr{T}(q_{10}), \mathscr{T}(q_{11}), \mathscr{T}(q_{12})\}$.

4.1.4 Ready and failure trace semantics

In this section we provide a coalgebraic modelling of ready and failure trace semantics by employing the generalised powerset construction. Similarly to the other semantics tackled so far, we show a) that the coalgebraic representation coincides with the original definition in [vG01a] and b) how to apply the coalgebraic machinery in order to reason on the corresponding equivalences.

Intuitively, ready trace semantics identifies two states if and only if they can follow the same traces w, and moreover, the corresponding (pairwise-taken) states determined by such w's have equivalent one-step behaviours. Failure trace semantics identifies states that can trigger the same traces w, and moreover, the (pairwise-taken) intermediate states occurring during the execution of a such w fail triggering the same (sets of) actions. Formally, the associated definitions are as follows:

4.1.14 Definition (Ready trace equivalence [Pnu85, vG01a]). Consider an LTS $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ and $x, y \in X$ two states. States x and y are *ready trace equivalent*

($\mathcal{RT}$-equivalent) if and only if $\mathcal{RT}(x) = \mathcal{RT}(y)$, where

$$\begin{aligned}\mathcal{RT}(x) = \{ \quad & I_0 a_1 I_1 a_2 \ldots a_n I_n \in \mathcal{P}_\omega(A) \times (A \times \mathcal{P}_\omega(A))^* \mid \\ & (\exists x_1, \ldots, x_n \in X).\, x = x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} x_n \wedge \\ & (\forall i = 0, \ldots, n).\, I_i = I(\delta(x_i)) \}.\end{aligned}$$

We call an element of $\mathcal{RT}(x)$ a *ready trace* of x. ♣

4.1.15 Definition (Failure trace equivalence [Phi87]). Let $(X, \delta : X \to (\mathcal{P}_\omega X)^A)$ be an LTS and $x, y \in X$ two states. States x and y are *failure trace equivalent* ($\mathcal{FT}$-equivalent) if and only if $\mathcal{FT}(x) = \mathcal{FT}(y)$, where

$$\begin{aligned}\mathcal{FT}(x) = \{ \quad & F_0 a_1 F_1 a_2 \ldots a_n F_n \in \mathcal{P}_\omega(A) \times (A \times \mathcal{P}_\omega(A))^* \mid \\ & (\exists x_1, \ldots, x_n \in X).\, x = x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} x_n \wedge F_i \in Fail(\delta(x_i)) \}.\end{aligned}$$

We call an element of $\mathcal{FT}(x)$ a *failure trace* of x. ♣

In order to model these two equivalences coalgebraically we will have to apply the generalised powerset construction, from Figure 4.2, not only by adding the output function but also by changing the transitions of the LTS.
In particular, we have to add to transitions of shape $x \xrightarrow{a} y$ information regarding the sets of actions ready to be triggered by x. In the new LTS we consider transitions of shape $x \xrightarrow{\langle a, I(\delta(x))\rangle} y$ therefore enabling the construction of Moore automata "collecting" states that have been reached not only via one-step transitions with the same label, but also from processes sharing the same initial behaviour. (Note that $F \in Fail(\delta(x))$ whenever $F \subseteq A - I(\delta(x))$.)
We apply the generalised powerset construction to the decorated LTS:

$$X \xrightarrow{\langle \bar{o}_{\mathcal{I}}, \bar{\delta}\rangle} \mathcal{P}_\omega(\mathcal{P}_\omega(A)) \times \mathcal{P}_\omega(X)^{A \times \mathcal{P}_\omega(A)}$$

where $\bar{\delta}$ is defined by first computing the set I and then appending it to every successor of a state by using the strength of powerset:

$$\bar{\delta} = X \xrightarrow{\delta} \mathcal{P}_\omega(X)^A \xrightarrow{\langle I, id\rangle} \mathcal{P}_\omega(A) \times \mathcal{P}_\omega(X)^A \xrightarrow{st} \mathcal{P}_\omega(\mathcal{P}_\omega(A) \times X)^A \to \mathcal{P}_\omega(X)^{A \times \mathcal{P}_\omega(A)}$$

$$\bar{\delta}(x)(\langle a, Z\rangle) = \begin{cases} \delta(x)(a) & \text{if } Z = I(\delta(x)) \\ \emptyset & \text{otherwise.} \end{cases}$$

For $\mathcal{I} \in \{\mathcal{RT}, \mathcal{FT}\}$, the output function $\bar{o}_{\mathcal{I}}$ provides information with respect to the actions ready, respectively, failed to be triggered by a state $x \in X$ as a first step:

$$\bar{o}_{\mathcal{RT}}(x) = \{I(\delta(x))\} \qquad\qquad \bar{o}_{\mathcal{FT}}(x) = Fail(\delta(x)).$$

We need to show that for $x_0 \in X$, there is a one-to-one correspondence between $[\![\{x_0\}]\!]$ and $\mathcal{I}(x_0)$. Intuitively, for ready trace semantics, for example, each behaviour

$$[\![\{x_0\}]\!](\bar{w}) = \{Z_n^j \mid x_a \xrightarrow{w} x_j\}, \quad \text{with } \bar{w} = \langle a_1, Z_0\rangle \ldots \langle a_n, Z_{n-1}\rangle \in (A \times \mathcal{P}_\omega(A))^* \text{ and } w = a_1 \ldots a_n \in A^*$$

corresponds to a set of sequences of shape

$$Z_0 a_1 Z_1 a_2 \ldots Z_{n-1} a_n Z_n^j \in \mathscr{I}(x_0).$$

Given $x \in X$, for $\mathscr{I} \in \{\mathscr{RT}, \mathscr{FT}\}$, we again represent $\mathscr{I}(x) \in \mathscr{P}(\mathscr{P}_\omega(A) \times (A \times \mathscr{P}_\omega(A))^*)$ by a function $\varphi_x^{\mathscr{I}}$:

$$\varphi_x^{\mathscr{RT}}(\bar{w}) = \{Z \subseteq A \mid x \xrightarrow{\bar{w}} y \wedge Z = I(\delta(y))\}$$

$$\varphi_x^{\mathscr{FT}}(\bar{w}) = \{Z \subseteq A \mid x \xrightarrow{\bar{w}} y \wedge Z \in Fail(\delta(y))\}$$

Showing the equivalence between the coalgebraic and the original definition of ready and failure trace semantics consists in proving that

$$(\forall x \in X).\, [\![\{x\}]\!] = \varphi_x^{\mathscr{I}}. \tag{4.10}$$

4.1.16 Theorem. *Let $(X, \delta : X \to (\mathscr{P}_\omega X)^A)$ be an LTS. Then for all $x \in X$ and $\bar{w} \in (A \times \mathscr{P}_\omega(A))^*$, $[\![\{x\}]\!](\bar{w}) = \varphi_x^{\mathscr{I}}(\bar{w})$.*

Proof. The proof follows by induction on words $w \in (A \times \mathscr{P}_\omega(A))^*$ (similar to the proof of Theorem 4.1.3). □

4.1.17 Example. Consider the following two systems:

$$\begin{tikzcd}
 & & p_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & \\
 & p_1 \arrow[dl, "b"'] \arrow[d, "c"] & & p_2 \arrow[d, "c"] \arrow[dr, "f"] & \\
p_3 & p_4 \arrow[d, "d"] & & p_5 \arrow[d, "e"] & p_6 \\
 & p_7 & & p_8 &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & q_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & \\
 & q_1 \arrow[dl, "b"'] \arrow[d, "c"] & & q_2 \arrow[d, "c"] \arrow[dr, "f"] & \\
q_3 & q_4 \arrow[d, "e"] & & q_5 \arrow[d, "d"] & q_6 \\
 & q_7 & & q_8 &
\end{tikzcd}$$

Note that they are not ready trace equivalent as, for example, $\{a\}a\{c,f\}c\{e\}$ is a ready trace of p_0 but not of q_0. Moreover, they are not failure trace equivalent as, for example, $\{b,c,d,e,f\}a\{a,d,e,f\}c\{a,b,c,e,f\}d\{a,b,c,d,e,f\}$ is a failure trace of p_0 but not of q_0.
It is easy to check that by taking exactly the generalised powerset construction (starting with $\{p_0\}, \{q_0\}$) without changing the transition function, as in Section 4.1.1, one gets two bisimilar Moore automata (for both the case of ready and failure trace equivalence). This would indicate that the initial LTS's are behavioural equivalent (which is not the case for ready and failure trace!).
The change in the transition function generates the automata (with labels in $A \times \mathscr{P}_\omega(A)$) in Figure 4.7. Then, for both semantics studied in this section, the determinisation derives the two Moore automata in Figure 4.8.
For ready trace semantics it holds that:

$$o_0 = \overline{o}_0 = \{\{a\}\} \quad o_{12} = \overline{o}_{12} = \{\{b,c\},\{c,f\}\} \quad o_4 = \overline{o}_5 = \{\{d\}\} \quad o_5 = \overline{o}_4 = \{\{e\}\}$$
$$o_3 = o_6 = o_7 = o_8 = \overline{o}_3 = \overline{o}_6 = \overline{o}_7 = \overline{o}_8 = \{\emptyset\}.$$

Hence, the systems in Figure 4.8 are not bisimilar as, for example, both states $\{p_4\}$ and $\{q_4\}$ can be reached via transitions labelled the same, but they output different sets of

$$\begin{tikzcd}
& & p_0 \arrow[dl, "{\langle a,\{a\}\rangle}"'] \arrow[dr, "{\langle a,\{a\}\rangle}"] & & \\
& p_1 \arrow[dl, "{\langle b,\{b,c\}\rangle}"'] \arrow[d, "{\langle c,\{b,c\}\rangle}"] & & p_2 \arrow[d, "{\langle c,\{c,f\}\rangle}"'] \arrow[dr, "{\langle f,\{c,f\}\rangle}"] & \\
p_3 & p_4 \arrow[d, "{\langle d,\{d\}\rangle}"'] & & p_5 \arrow[d, "{\langle e,\{e\}\rangle}"] & p_6 \\
& p_7 & & p_8 &
\end{tikzcd}
\qquad
\begin{tikzcd}
& & q_0 \arrow[dl, "{\langle a,\{a\}\rangle}"'] \arrow[dr, "{\langle a,\{a\}\rangle}"] & & \\
& q_1 \arrow[dl, "{\langle b,\{b,c\}\rangle}"'] \arrow[d, "{\langle c,\{b,c\}\rangle}"] & & q_2 \arrow[d, "{\langle c,\{c,f\}\rangle}"'] \arrow[dr, "{\langle f,\{c,f\}\rangle}"] & \\
q_3 & q_4 \arrow[d, "{\langle e,\{e\}\rangle}"'] & & q_5 \arrow[d, "{\langle d,\{d\}\rangle}"] & q_6 \\
& q_7 & & q_8 &
\end{tikzcd}$$

Figure 4.7: Altered transition function before determinisation.

$$\begin{tikzcd}
\{p_3\} \arrow[d, Rightarrow] & & \{p_4\} \arrow[r, "{\langle d,\{d\}\rangle}"] \arrow[d, Rightarrow] & \{p_7\} \arrow[d, Rightarrow] \\
o_3 & & o_4 & o_7 \\
\{p_0\} \arrow[r, "{\langle a,\{a\}\rangle}"] \arrow[d, Rightarrow] & \{p_1,p_2\} \arrow[uul, "{\langle b,\{b,c\}\rangle}"] \arrow[uur, "{\langle c,\{b,c\}\rangle}"] \arrow[r, "{\langle c,\{c,f\}\rangle}"] \arrow[d, Rightarrow] \arrow[ddr, "{\langle f,\{c,f\}\rangle}"'] & \{p_5\} \arrow[r, "{\langle e,\{e\}\rangle}"] \arrow[d, Rightarrow] & \{p_8\} \arrow[d, Rightarrow] \\
o_0 & o_{12} & o_5 & o_8 \\
& & \{p_6\} \arrow[r, Rightarrow] & o_6
\end{tikzcd}
\qquad
\begin{tikzcd}
\{q_3\} \arrow[d, Rightarrow] & & \{q_4\} \arrow[r, "{\langle e,\{e\}\rangle}"] \arrow[d, Rightarrow] & \{q_7\} \arrow[d, Rightarrow] \\
\bar{o}_3 & & \bar{o}_4 & \bar{o}_7 \\
\{q_0\} \arrow[r, "{\langle a,\{a\}\rangle}"] \arrow[d, Rightarrow] & \{q_1,q_2\} \arrow[uul, "{\langle b,\{b,c\}\rangle}"] \arrow[uur, "{\langle c,\{b,c\}\rangle}"] \arrow[r, "{\langle c,\{c,f\}\rangle}"] \arrow[d, Rightarrow] \arrow[ddr, "{\langle f,\{c,f\}\rangle}"'] & \{q_5\} \arrow[r, "{\langle d,\{d\}\rangle}"] \arrow[d, Rightarrow] & \{q_8\} \arrow[d, Rightarrow] \\
\bar{o}_0 & \bar{o}_{12} & \bar{o}_5 & \bar{o}_8 \\
& & \{q_6\} \arrow[r, Rightarrow] & \bar{o}_6
\end{tikzcd}$$

♠

Figure 4.8: Determinisation starting from $\{p_0\}, \{q_0\}$.

ready actions – namely $\{\{d\}\}$ and $\{\{e\}\}$, respectively. Therefore, we conclude that p_0 and q_0 are not ready trace equivalent.
Similarly, for failure trace we have:

$$\begin{array}{ll}
o_0 = \bar{o}_0 = [bcdef] & o_{12} = \bar{o}_{12} = [adef] \cup [abde] \\
o_4 = \bar{o}_5 = [abcef] & o_5 = \bar{o}_4 = [abcdf] \\
\multicolumn{2}{l}{o_3 = o_6 = o_7 = o_8 = \bar{o}_3 = \bar{o}_6 = \bar{o}_7 = \bar{o}_8 = [abcdef].}
\end{array}$$

As before, the automata in Figure 4.8 are not bisimilar as, for example, both $\{p_4\}$ and $\{q_4\}$ are reached via transitions labelled the same, but have different outputs. Therefore we conclude that p_0 and q_0 are not failure trace equivalent.
The purpose of changing the transition labels with sets of ready actions is to collect in a Moore state only states of the initial LTS's that have been reached from "parents" with the same one-step (initial) behaviour. Or dually, to distinguish between states that have "parents" ready, respectively, failing to trigger different sets of actions. This way one avoids the unfortunate situation of encapsulating, for example, the states p_4, p_5, respectively q_4, q_5, fact which eventually would lead to providing a positive answer with respect to both ready and failure trace equivalence of p_0 and q_0.
In other words, the change in the transition function is needed in order to guarantee that whenever two states of an LTS are ready/failure trace equivalent, the (pairwise-taken) states determined by the executions of a given trace have the same initial behaviour.

4.2 Decorated trace semantics of GPS's

In this section we show how the generalised powerset construction for coalgebras $f: X \to \mathcal{F}T(X)$ for a functor $\mathcal{F}$ and a monad T in (2.7), Section 2.3, can be instantiated in order to provide coalgebraic modellings of decorated trace semantics for generative probabilistic systems (GPS's). More explicitly, we show how the determinisation procedure can be applied in order to derive coalgebraic representations of ready, (maximal) failure and (maximal) trace semantics, equivalent to their standard definitions in [JS90].
A GPS is similar to an LTS, but each transition is labelled by both an action and a probability p. More precisely, the transition dynamics is given by a *probabilistic transition function* $\mu: X \times A \times X \to [0,1]$ satisfying

$$(\forall x \in X).\sum_{\substack{a\in A\\ y\in X}} \mu(x,a,y) \leq 1, \tag{4.11}$$

where X is the state space and A is the alphabet of actions. For simplicity, we write $\mu_a(x,y)$ in lieu of $\mu(x,a,y)$ and we will use the notation $x \xrightarrow{a[v]} y$ for $\mu_a(x,y) = v$. We extend μ to words $w \in A^*$:

$$\mu_\varepsilon(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \qquad \mu_{aw}(x,y) = \sum_{x' \in X} \mu_a(x,x') \times \mu_w(x',y)$$

Intuitively, $\mu_w(x,y)$ represents the sum of the probabilities associated with all traces w from x to y. Moreover, we write

$$\mu_0(x,\mathbf{0}) = 1 - \sum_{\substack{a\in A\\ y\in X}} \mu(x,a,y)$$

for the probability of x to *terminate*, where 0 is a special symbol not in A, called the *zero action*, and $\mathbf{0}$ is the (deadlock-like) *zero process* whose only transition is $\mu_0(\mathbf{0},\mathbf{0}) = 1$.
Similarly to the case of LTS's, the set of initial actions that can be triggered (with a probability greater than 0) from $x \in X$ is given by

$$I(x) = \{a \in A \mid (\exists y \in X).\mu_a(x,y) > 0\},$$

whereas failure sets $Z \in \mathcal{P}_\omega A$ satisfy the condition $Z \cap I(x) = \emptyset$. We write $Fail(x)$ to represent the set of all failure sets of x.
The decorated trace semantics for GPS's considered in this paper can be intuitively described as follows. Given two states $x, y \in X$, we say that x and y are equivalent whenever traces $w \in A^*$

- lead, with the same probability, x and y to processes that trigger (respectively, fail to execute) as a first step the same sets of actions, for the case of ready (respectively, failure) semantics. Note that maximal failure semantics takes into consideration only the largest sets of failure actions (*i.e.*, $A - I(x), A - I(y)$).

- can be executed with the same probability from both x and y, for the case of trace semantics and, moreover, lead x and y to processes that have the same probability to terminate, for the case of maximal trace semantics.

To model GPS's, we consider $\mathscr{D}_\omega(X)$ – the (finitely supported sub)probability distribution functor defined on **Set**. $\mathscr{D}_\omega$ maps a set X to

$$\mathscr{D}_\omega(X) = \{\varphi : X \to [0,1] \mid supp(\varphi) \text{ is finite and } \sum_{x\in X} \varphi(x) \leq 1\},$$

where $supp(\varphi) = \{x \in X \mid \varphi(x) > 0\}$ is the *support* of φ. Given a function $g : X \to Y$, $\mathscr{D}_\omega(g) : \mathscr{D}_\omega(X) \to \mathscr{D}_\omega(Y)$ is defined as

$$\mathscr{D}_\omega(g)(\varphi) = \lambda y \,.\, \sum_{g(x)=y} \varphi(x).$$

A GPS is a coalgebra

$$(X, \delta : X \to (\mathscr{D}_\omega(X))^A)$$

such that $\delta(x)(a)(y) = \mu_a(x,y)$[1].
To each GPS we associate a *decorated* GPS's

$$(X, \langle \overline{o}_{\mathscr{I}}, \delta\rangle : X \to B_{\mathscr{I}} \times (\mathscr{D}_\omega(X))^A)$$

"parameterised" by $\mathscr{I}$, depending on the semantics under consideration.
Decorated GPS's can be determinised according to the generalised powerset construction as illustrated in Figure 4.9, where $\mathscr{F}$ is $B_{\mathscr{I}} \times (-)^A$ and T is instantiated with the probability distribution monad $(\mathscr{D}_\omega, \eta, \mu)$:

$$\begin{aligned} &\eta : X \to \mathscr{D}_\omega(X) && \mu : \mathscr{D}_\omega(\mathscr{D}_\omega(X)) \to \mathscr{D}_\omega(X) \\ &\eta(x) = \lambda y \,.\, \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} && \mu(\psi) = \lambda x \,.\, \sum_{\varphi \in supp(\psi)} \varphi(x) \times \psi(\varphi) \end{aligned}$$

Algebras for this monad are the so-called positive convex structures [Dob08].
Moreover, for each of the semantics of interest the observations set $B_{\mathscr{I}}$ has to carry a $\mathscr{D}_\omega$-algebra structure, or, equivalently, there has to exist a morphism $h_{\mathscr{I}}$ such that $(B_{\mathscr{I}}, h_{\mathscr{I}} : \mathscr{D}_\omega(B_{\mathscr{I}}) \to B_{\mathscr{I}})$ is a $\mathscr{D}_\omega$-algebra (as introduced in Definition 2.3.2, in Section 2.3).

The ingredients $\overline{o}_{\mathscr{I}}, B_{\mathscr{I}}$ and $h_{\mathscr{I}}$ of Figure 4.9 are explicitly defined in the subsequent sections for each of the coalgebraic decorated trace semantics. The latter are also proven to be equivalent with their corresponding definitions in [JS90].

4.2.1 Ready and (maximal) failure semantics

In this section we provide the detailed coalgebraic modelling of ready and (maximal) failure semantics and show the equivalence with their counterparts defined in [JS90], as follows:

4.2.1 DEFINITION (Ready equivalence [JS90]). The *ready function*

$$\mathscr{R}_p : X \to ((A^* \times \mathscr{P}_\omega A) \to [0,1])$$

[1]Note that the coalgebraic type directly corresponds to reactive systems [BSdV04]. The embedding of generative into reactive is injective and poses no problems semantic-wise. In the sequel, when we write "Let $(X, \delta : X \to (\mathscr{D}_\omega(X))^A)$ be a GPS" we implicitly mean a coalgebra of this type originating from a GPS defined by a probabilistic function $\mu : X \times A \times X \to [0,1]$ as in (4.11).

$$\begin{tikzcd}
X \arrow[r, "\eta"] \arrow[d, "\langle \overline{o}_{\mathscr{I}}, \delta\rangle"'] & \mathscr{D}_\omega(X) \arrow[r, dashed, "\llbracket - \rrbracket"] \arrow[dl, "\langle o,t\rangle"] & (B_{\mathscr{I}})^{A^*} \arrow[d, "\langle \epsilon, (-)_a\rangle"] \\
B_{\mathscr{I}} \times (\mathscr{D}_\omega(X))^A \arrow[rr, dashed, "id_{B_{\mathscr{I}}} \times \llbracket - \rrbracket^A"'] & & B_{\mathscr{I}} \times ((B_{\mathscr{I}})^{A^*})^A
\end{tikzcd}$$

$$o = h_{\mathscr{I}} \circ \mathscr{D}_\omega(\overline{o}_{\mathscr{I}})$$
$$t(\varphi)(a)(y) = \sum_{x \in supp(\varphi)} \delta(x)(a)(y) \times \varphi(x)$$

$$\llbracket \varphi \rrbracket(\varepsilon) = o(\varphi)$$
$$\llbracket \varphi \rrbracket(aw) = \llbracket t(\varphi)(a) \rrbracket(w)$$

Figure 4.9: The powerset construction for decorated GPS's.

is given by
$$\mathscr{R}_p(x)((w,I)) = \sum_{I=I(y)} \mu_w(x,y).$$
We say that $x, x' \in X$ are *ready equivalent* whenever $\mathscr{R}_p(x) = \mathscr{R}_p(x')$. ♣

4.2.2 Definition (Failure equivalence [JS90]). The *failure function*
$$\mathscr{F}_p : X \to ((A^* \times \mathscr{P}_\omega A) \to [0,1])$$
is given by
$$\mathscr{F}_p(x)((w,Z)) = \sum_{Z \cap I(y) = \emptyset} \mu_w(x,y).$$
We say that $x, x' \in X$ are *failure equivalent* whenever $\mathscr{F}_p(x) = \mathscr{F}_p(x')$. ♣

4.2.3 Definition (Maximal failure equivalence [JS90]). The *maximal failure function* $\mathscr{M}\mathscr{F}_p : X \to ((A^* \times \mathscr{P}_\omega A) \to [0,1])$ is given by
$$\mathscr{M}\mathscr{F}_p(x)((w,Z)) = \sum_{Z = A - I(y)} \mu_w(x,y).$$
We say that $x, x' \in X$ are *maximal failure equivalent* whenever $\mathscr{M}\mathscr{F}_p(x) = \mathscr{M}\mathscr{F}_p(x')$. ♣

Intuition: *ready* and *(maximal) failure semantics*, respectively, identify states which have the same probability of reaching processes sharing the same sets of ready actions I, or (maximal) sets of failure actions Z, respectively, by executing the same traces $w \in A^*$. Consequently, appropriate modellings in the coalgebraic setting should capture sets of traces w, together with some notion of observations based on execution probabilities of such w's and sets of ready/(maximal) failure actions.
As a first step we define $B_{\mathscr{I}}$, the observation set in Figure 4.9, as $[0,1]^{\mathscr{P}_\omega(A)}$, for ready, failure and maximal failure semantics (for which, for consistency of notation, $\mathscr{I}$ will be instantiated with $\mathscr{R}_p$, $\mathscr{F}_p$ and $\mathscr{M}\mathscr{F}_p$, respectively).
The associated "decorating" functions $\overline{o}_{\mathscr{I}} : X \to [0,1]^{\mathscr{P}_\omega(A)}$ are defined for $x \in X$ as:

$$\overline{o}_{\mathscr{R}_p}(x)(I) = \begin{cases} 1 & \text{if } I = I(x) \\ 0 & \text{otherwise.} \end{cases} \qquad \overline{o}_{\mathscr{F}_p}(x)(Z) = \begin{cases} 1 & \text{if } Z \cap I(x) = \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

$$\overline{o}_{\mathscr{MF}_p}(x)(Z) = \begin{cases} 1 & \text{if } Z = A - I(x) \\ 0 & \text{otherwise.} \end{cases}$$

For the generalised powerset construction for GPS's, $B_{\mathscr{I}} = [0,1]^{\mathscr{P}_\omega(A)}$ is required to carry a $\mathscr{D}_\omega$-algebra structure. This structure is given by the pointwise extension of the free algebra structure in $[0,1] = \mathscr{D}_\omega(1)$:

$$\begin{aligned} h_{\mathscr{I}} &: \mathscr{D}_\omega([0,1]^{\mathscr{P}_\omega(A)}) \to [0,1]^{\mathscr{P}_\omega(A)} \\ h_{\mathscr{I}}(\varphi)(Z) &= \sum_{f \in supp(\varphi)} \varphi(f) \times f(Z). \end{aligned}$$

It is easy to check that, for $\mathscr{I} \in \{\mathscr{R}_p, \mathscr{F}_p, \mathscr{MF}_p\}$, the output function $o = h_{\mathscr{I}} \circ \mathscr{D}_\omega(\overline{o}_{\mathscr{I}})$ is explicitly defined, for $\varphi \in \mathscr{D}_\omega(X)$, as:

$$o(\varphi)(S) = \sum_{x \in supp(\varphi)} \varphi(x) \times \overline{o}_{\mathscr{I}}(x)(S).$$

This enables the modelling of the behaviour of GPS's in terms of (final) Moore machines with state space in $(B_{\mathscr{I}})^{A^*}$ and observations in $B_{\mathscr{I}}$. More explicitly, given a GPS (X, δ), the decorated trace behaviour of $x \in X$ is represented in the coalgebraic setting by $[\![\eta(x)]\!] \in (B_{\mathscr{I}})^{A^*} = ([0,1]^{\mathscr{P}_\omega(A)})^{A^*} \cong [0,1]^{A^* \times \mathscr{P}_\omega(A)}$, precisely the type of the functions in Definitions 4.2.1–4.2.3. This paves the way for reasoning on ready and (maximal) failure equivalence by coinduction, in terms of Moore bisimulations.

4.2.4 Example. Consider, for example, the following GPS's:

$$\begin{tikzcd} & p' \arrow[dl, "{a[x]}"'] \arrow[dr, "{a[1-x]}"] & & & u' \arrow[d, "{a[1]}"] & \\ q' \arrow[d, "{a[1]}"'] & & r' \arrow[d, "{a[1]}"] & & v' \arrow[dl, "{a[y]}"'] \arrow[dr, "{a[1-y]}"] & \\ s' & & t' & w' & & w'' \end{tikzcd}$$

States p' and u' are ready equivalent, as their corresponding ready functions in Definition 4.2.1 are equal:

$$\begin{array}{rcl} \mathscr{R}_p(p')(\varepsilon,\emptyset) & = & 0 \quad \mathscr{R}_p(p')(\varepsilon,\{a\}) = 1 \quad \mathscr{R}_p(p')(a,\emptyset) = 0 \\ \mathscr{R}_p(p')(a,\{a\}) & = & \mu_a(p',q') + \mu_a(p',r') \ = x + (1-x) = 1 \\ \mathscr{R}_p(p')(aa,\emptyset) & = & \mu_{aa}(p',s') + \mu_{aa}(p',t') \ = x \times 1 + (1-x) \times 1 = 1 \\ \mathscr{R}_p(p')(aa,\{a\}) & = & 0 \quad \mathscr{R}_p(u')(\varepsilon,\emptyset) = 0 \quad \mathscr{R}_p(u')(\varepsilon,\{a\}) = 1 \\ \mathscr{R}_p(u')(a,\{a\}) & = & \mu_a(u',v') = 1 \quad \mathscr{R}_p(u')(a,\emptyset) = 0 \quad \mathscr{R}_p(u')(aa,\{a\}) = 0 \\ \mathscr{R}_p(u')(aa,\emptyset) & = & \mu_{aa}(u',w') + \mu_{aa}(u',w'') = 1 \times y + 1 \times (1-y) = 1 \end{array}$$

Intuitively, $\mathscr{R}_p(p')(\varepsilon,\emptyset) = 0$ states that from p', by executing the empty trace ε, the probability to reach states that cannot further trigger any action is 0. This is indeed the case, as p' can always fire a as a first step. Similarly, $\mathscr{R}_p(u')(a,\{a\}) = 1$ states that the probability of performing a from u' and reaching states with the ready set $\{a\}$ is 1. This because $u' \xrightarrow{a[1]} v'$ and $I(v') = \{a\}$. Nevertheless, the aforementioned ready equivalence follows according to the hierarchy in the right-hand side of Figure 4.1, as p' and u' are probabilistic bisimilar as well.

The same answer with respect to the ready equivalence of p' and u' is obtained by applying the coalgebraic framework. As illustrated below, the corresponding Moore automata derived starting from p' and u', respectively, are bisimilar; they have the same branching structure and equal outputs:

$$p': \quad \begin{tikzcd} \varphi_1 \arrow[r, "a"] \arrow[d, Rightarrow] & \varphi_2 \arrow[r, "a"] \arrow[d, Rightarrow] & \varphi_3 \arrow[d, Rightarrow] \\ o_{\varphi_1} & o_{\varphi_2} & o_{\varphi_3} \end{tikzcd} \qquad u': \quad \begin{tikzcd} \alpha_1 \arrow[r, "a"] \arrow[d, Rightarrow] & \alpha_2 \arrow[r, "a"] \arrow[d, Rightarrow] & \alpha_3 \arrow[d, Rightarrow] \\ o_{\alpha_1} & o_{\alpha_2} & o_{\alpha_3} \end{tikzcd}$$

The state spaces of the aforementioned Moore machines consist of the functions:

$$\begin{array}{lcl} \varphi_1 & = & \eta(p') = \{p' \to 1,\, q' \to 0,\, r' \to 0,\, s' \to 0,\, t' \to 0\} \\ \varphi_2 & = & \{p' \to 0,\, q' \to x,\, r' \to 1-x,\, s' \to 0,\, t' \to 0\} \\ \varphi_3 & = & \{p' \to 0,\, q' \to 0,\, r' \to 0,\, s' \to 1,\, t' \to 1\} \\ \alpha_1 & = & \eta(u') = \{u' \to 1,\, v' \to 0,\, w' \to 0,\, w'' \to 0\} \\ \alpha_2 & = & \{u' \to 0,\, v' \to 1,\, w' \to 0,\, w'' \to 0\} \\ \alpha_3 & = & \{u' \to 0,\, v' \to 0,\, w' \to y,\, w'' \to 1-y\}. \end{array}$$

The associated observations are:

$$o_{\varphi_1} = o_{\alpha_1} = o_{\varphi_2} = o_{\alpha_2} = (\emptyset \mapsto 0, \{a\} \mapsto 1), o_{\varphi_3} = o_{\alpha_3} = (\emptyset \mapsto 1, \{a\} \mapsto 0.)$$

The functions φ_2, φ_3, α_2 and α_3 together with their outputs are easily determined based on the operations of the corresponding Moore coalgebra (as depicted in Figure 4.9).
The connection between the behaviour, *i.e.*, ready function of p' (respectively, u') and φ_i (respectively, α_i), for $i \in \{1,2,3\}$, is straightforward. Each of the functions φ_1, φ_2 and φ_3 captures the behaviour of the system starting from p', after executing the traces ε, a and aa, respectively. Note that, for example, the values of the ready function for trace ε and ready sets $\emptyset$ and $\{a\}$, respectively, are in one to one correspondence with the assignments in o_{φ_1}. Similarly for the case of u'.
By following the same approach, the coalgebraic machinery provides an "yes" answer with respect to (maximal) failure equivalence of p' and u' as well. This is also in agreement with the results in [JS90] stating that ready and (maximal) failure equivalence for GPS's have the same distinguishing power. ♠

The equivalence between the coalgebraic and the original definitions of the decorated trace semantics $\mathscr{I} \in \{\mathscr{R}_p, \mathscr{F}_p, \mathscr{MF}_p\}$ in [JS90] consists in showing that, given a GPS (X, δ), $x \in X$, $w \in A^*$ and $S \subseteq A$, it holds that $[\![\eta(x)]\!](w)(S) = I(x)(w, S)$.

4.2.5 Theorem. *Let $(X, \delta : X \to (\mathscr{D}_\omega(X))^A)$ be a GPS and $(\mathscr{D}_\omega(X), \langle o, t\rangle)$ be its associated determinisation as in Figure 4.9. Then, for all $x \in X$, $w \in A^*$ and $S \subseteq A$, it holds*

$$[\![\eta(x)]\!](w)(S) = \mathscr{I}(x)(w, S).$$

Proof. The proof is similar for all $\mathscr{I}$ in $\{\mathscr{R}_p, \mathscr{F}_p, \mathscr{MF}_p\}$, by induction on $w \in A^*$.

- *Base case* – $w = \varepsilon$: $[\![\eta(x)]\!](\varepsilon)(S) = \overline{o}_{\mathscr{I}}(x)(S) = \mathscr{I}(x)(\varepsilon, S)$.

- *Induction step.* Here, we will use the fact that the map into the final coalgebra is also an algebra map and the equality

$$\mathscr{I}(x)(aw, S) = \sum_{y \in Y} \mu_a(x, y) \times \mathscr{I}(x)(w)(S).$$

Consider $aw \in A^*$ and assume $[\![\eta(y)]\!](w)(S) = \mathscr{I}(y)(w,S)$, for all $y \in X$. We want to prove that $[\![\eta(x)]\!](aw)(S) = \mathscr{I}(x)(aw)(S)$, for $a \in A$.

$$\begin{aligned}
[\![\eta(x)]\!](aw)(S) &= [\![\delta(x)(a)]\!](w)(S) \\
&= \sum_{y \in Y} \delta(x)(a)(y) \times [\![\eta(y)]\!](w)(S) && ([\![-]\!] \text{ is an algebra map}) \\
&= \sum_{y \in Y} \delta(x)(a)(y) \times \mathscr{I}(x)(w)(S) && \text{(IH)} \\
&= \sum_{y \in Y} \mu_a(x,y) \times \mathscr{I}(x)(w)(S) && (\mu_a(x,x') = \delta(x)(a)(x')) \\
&= \mathscr{I}(x)(aw)(S)
\end{aligned}$$

□

4.2.2 (Maximal) trace semantics

In this section we provide the coalgebraic modelling of (maximal) trace semantics for GPS's. The approach resembles the one in the previous section: we first recall the aforementioned semantics as introduced in [JS90], and then show how to instantiate the ingredients of Figure 4.9 in order to capture the corresponding behaviours in terms of (final) Moore coalgebras. As a last step, we prove the equivalence between the coalgebraic modellings and the original definitions in [JS90].

4.2.6 Definition ((Maximal) trace equivalence [JS90]). The *trace function* $\mathscr{T}_p : X \to (A^* \to [0,1])$ is given by

$$\mathscr{T}_p(x)(w) = \sum_{y \in X} \mu_w(x,y).$$

The *maximal trace function* $\mathscr{M}\mathscr{T}_p : X \to (A^* \to [0,1])$ is given by

$$\mathscr{M}\mathscr{T}_p(x)(w) = \mu_{w0}(x, \mathbf{0}).$$

We say that $x, x' \in X$ are *trace equivalent* whenever $\mathscr{T}_p(x) = \mathscr{T}_p(x')$. If $\mathscr{M}\mathscr{T}_p(x) = \mathscr{M}\mathscr{T}_p(x')$ holds as well, then we say that x and x' are *maximal trace equivalent*. ♣

From the definition above, it can be easily seen at an intuitive level that trace equivalence identifies processes that can execute with the same probability the same sets of traces $w \in A^*$. Moreover, maximal trace equivalence takes into consideration the probability of not triggering any action after the performance of such w's.
Therefore, we choose the set of observations $B_{\mathscr{I}}$ (where $\mathscr{I} = \mathscr{T}_p$ for trace and $\mathscr{I} = \mathscr{M}\mathscr{T}_p$ for maximal trace semantics) to denote probabilities (of processes to execute $w \in A^*$, or stagnate after triggering such w's) ranging over $[0,1]$.
We define the "decorating" functions, for $\mathscr{I} \in \{\mathscr{T}_p, \mathscr{M}\mathscr{T}_p\}$, $\overline{o}_{\mathscr{I}} : X \to [0,1]$ by

$$\overline{o}_{\mathscr{T}_p}(x) = 1 \qquad \overline{o}_{\mathscr{M}\mathscr{T}_p}(x) = \mu_0(x, \mathbf{0})$$

The (Moore) output function o is given by, for all $\varphi \in \mathscr{D}_\omega(X)$,

$$o(\varphi) = \sum_{x \in supp(\varphi)} \varphi(x) \times \overline{o}_{\mathscr{I}}(x).$$

We can now show the equivalence between the coalgebraic and the original definition of (maximal) trace semantics.

4.2.7 THEOREM. *Let $(X, \delta : X \to (\mathcal{D}_\omega(X))^A)$ be a GPS and $(\mathcal{D}_\omega(X), \langle o, t\rangle)$ be its associated determinisation as in Figure 4.9. Then, for all $x \in X$ and $w \in A^*$:*

$$[\![\eta(x)]\!](w) = \mathscr{I}(x)(w).$$

PROOF. By induction on $w \in A^*$, similar to Theorem 4.2.5. □

Consider, for instance, the systems p' and u' in Example 4.2.4. They are trace equivalent as they both can execute traces ε, a and aa with total probability 1. Consequently, they are maximal trace equivalent as well: for sequences ε and a, their associated maximal trace functions compute value 0, whereas for aa the latter return value 1.
The same answer with respect to (maximal) trace equivalence of p' and u' is obtained by reasoning on bisimilarity of their associated determinisations derived according to the powerset construction. It is easy to check that in the current setting, the Moore automata corresponding to φ_1 and α_1 in Example 4.2.4 output

- for the case of trace semantics:

$$(\forall i \in \{1,2,3\}) \,.\, o_{\varphi_i} = o_{\alpha_i} = 1;$$

- for the case of maximal trace semantics:

$$(\forall i \in \{1,2\}) \,.\, o_{\varphi_i} = o_{\alpha_i} = 0 \text{ and } o_{\varphi_3} = o_{\alpha_3} = 1.$$

Therefore φ_1 and α_1 are bisimilar. Hence, p' and u' are (maximal) trace equivalent.

4.3 Decorated trace semantics in a nutshell

Next we provide a more compact overview on the coalgebraic machinery introduced in Section 4.1 and Section 4.2. This also in order to emphasise on the generality and uniformity of our coalgebraic framework.
Recall that for each of the decorated trace semantics we first instantiate the constituents of Figure 4.2 (summarising the generalised powerset construction). Moreover, for the case of LTS's, the original definitions of the semantics under consideration are provided with equivalent representations in terms of functions $\varphi_Y^{\mathscr{I}}$, paving the way to their interpretation in terms of final Moore coalgebras.
All these are summarised in Figure 4.10, for an arbitrary LTS $(X, \delta : X \to (\mathcal{P}_\omega X)^A)$ and an arbitrary GPS $(X, \delta : X \to (\mathcal{D}_\omega X)^A)$.
Once the ingredients of Figure 4.2 and, for LTS's, functions $\varphi_Y^{\mathscr{I}}$ are defined, we formalise the equivalence between the coalgebraic modelling of $\mathscr{I}$-semantics and its original definition.
For the case of LTS's, for $\mathscr{I}$ ranging over $\mathscr{T}, \mathscr{CT}, \mathscr{F}, \mathscr{R}, \mathscr{PF}, \mathscr{RT}$ and $\mathscr{FT}$, we show that the following result holds:

4.3.1 THEOREM. *Let $(X, \delta : X \to (\mathcal{P}_\omega X)^A)$ be an LTS. For all $x \in X$, $[\![\{x\}]\!] = \varphi_x^{\mathscr{I}} \cong \mathscr{I}(x)$.*

Orthogonally, for the case of GPS's, for $\mathscr{I}$ ranging over $\mathscr{R}_p, \mathscr{F}_p, \mathscr{MF}_p, \mathscr{T}_p$ and $\mathscr{MT}_p$, we prove the following:

4.3.2 THEOREM. *Let $(X, \delta : X \to (\mathcal{D}_\omega X)^A)$ be a GPS. For all $x \in X$, $[\![\eta(x)]\!] = \mathscr{I}(x)$.*

For each of the semantics under consideration, the proofs of Theorem 4.3.1 and Theorem 4.3.2, follow by induction on words over the corresponding action alphabet. For more details see the proof of Theorem 4.1.3 in Section 4.1.1 (for LTS's) and Theorem 4.2.5 in Section 4.2.1 (for GPS's), respectively.

Remark 12 *It is worth observing that by instantiating T with the identity functor, $\mathscr{F}$ with $\mathscr{P}_\omega(-)^A$ and, respectively, $\mathscr{D}_\omega(-)^A$ in (2.7), in Section 2.3, one gets the coalgebraic modelling of the standard notion of bisimilarity for LTS's and, respectively, GPS's.*

Concrete examples on how to use the coalgebraic frameworks are provided for each of the decorated trace semantics. We show how to derive determinisations of LTS's and GPS's in terms of Moore automata, which eventually are used to reason on the corresponding equivalences in terms of Moore bisimulations.

$\mathscr{I}$	$B_{\mathscr{I}}$	$\bar{o}_{\mathscr{I}}: X \to B_{\mathscr{I}}$
$\mathscr{R}$	$\mathscr{P}_\omega(\mathscr{P}_\omega A)$	$\bar{o}_R(x) = \{I(\delta(x))\}$
$\mathscr{F}$	$\mathscr{P}_\omega(\mathscr{P}_\omega A)$	$\bar{o}_{\mathscr{F}}(x) = Fail(\delta(x))$
$\mathscr{T}$	2	$\bar{o}_{\mathscr{T}}(x) = 1$
$\mathscr{C}\mathscr{T}$	2	$\bar{o}_{\mathscr{C}\mathscr{T}}(x) = \begin{cases} 1 & \text{if } I(\delta(x)) = \emptyset \\ 0 & \text{otherwise} \end{cases}$
$\mathscr{P}\mathscr{F}$	$\mathscr{P}(\mathscr{P}A^*)$	$\bar{o}_{\mathscr{P}\mathscr{F}}(x) = \{\mathscr{T}(x)\}$
$\mathscr{R}\mathscr{T}$	$\mathscr{P}_\omega(\mathscr{P}_\omega A)$	$\bar{o}_{\mathscr{R}\mathscr{T}}(x) = \{I(\delta(x))\}$
$\mathscr{F}\mathscr{T}$	$\mathscr{P}_\omega(\mathscr{P}_\omega A)$	$\bar{o}_{\mathscr{F}\mathscr{T}}(x) = Fail(\delta(x))$
$\mathscr{R}_p$	$[0,1]^{\mathscr{P}_\omega(A)}$	$\bar{o}_{\mathscr{R}_p}(x)(I) = \begin{cases} 1 & \text{if } I = I(x) \\ 0 & \text{otherwise} \end{cases}$
$\mathscr{F}_p$	$[0,1]^{\mathscr{P}_\omega(A)}$	$\bar{o}_{\mathscr{F}_p}(x)(Z) = \begin{cases} 1 & \text{if } Z \cap I(x) = \emptyset \\ 0 & \text{otherwise} \end{cases}$
$\mathscr{M}\mathscr{F}_p$	$[0,1]^{\mathscr{P}_\omega(A)}$	$\bar{o}_{\mathscr{M}\mathscr{F}_p}(x)(Z) = \begin{cases} 1 & \text{if } Z = A - I(x) \\ 0 & \text{otherwise} \end{cases}$
$\mathscr{T}_p$	$[0,1]$	$\bar{o}_{\mathscr{T}_p}(x) = 1$
$\mathscr{M}\mathscr{T}_p$	$[0,1]$	$\bar{o}_{\mathscr{M}\mathscr{T}_p}(x) = \mu_0(x, \mathbf{0})$

Figure 4.10: The coalgebraic framework in a nutshell.

4.4 Canonical representatives

Given a *decorated* system $(X, \langle \bar{o}_{\mathscr{I}}, \delta \rangle)$, we showed in the previous sections how to construct a determinisation $(T(X), \langle o, t \rangle)$, with $T = \mathscr{P}_\omega$ for the case of LTS's, and $T = \mathscr{D}_\omega$ for GPS's, respectively. The map $[\![-]\!]: TX \to B_{\mathscr{I}}^{A^*}$ provides us with a *canonical representative* of the behaviour of each state in TX. The image (C, δ') of $(TX, \langle o, t \rangle)$, via the map $[\![-]\!]$, can be viewed as the minimisation with respect to the equivalence $\mathscr{I}$.
Recall that the states of the final Moore coalgebra $(B_{\mathscr{I}}^{A^*}, \langle \epsilon, (-)_a \rangle)$ are functions $\varphi: A^* \to B_{\mathscr{I}}$ and that their decorations and transitions are given by the functions $\epsilon: B_{\mathscr{I}}^{A^*} \to B_{\mathscr{I}}$ and

$(-)_a\colon B_{\mathcal{G}}^{A^*} \to (B_{\mathcal{G}}^{A^*})^A$, defined in Example 2.2.8 in Section 2.2. The states of the canonical representative (C, δ') are also functions $\varphi\colon A^* \to B_{\mathcal{G}}$, i.e., $C \subseteq B_{\mathcal{G}}^{A^*}$. Moreover, the function $\delta'\colon C \to B_{\mathcal{G}} \times C^A$ is simply the restriction of $\langle \epsilon, (-)_a \rangle$ to C, that means $\delta'(\varphi) = \langle \varphi(\epsilon), (\varphi)_a \rangle$ for all $\varphi \in C$.
Finally, it is interesting to observe that for LTS $B_{\mathcal{G}}^{A^*}$ carries a semilattice structure (inherited from $B_{\mathcal{G}}$) and that $[\![-]\!]\colon \mathcal{P}_\omega X \to B_{\mathcal{G}}^{A^*}$ is a semilattice homomorphism. From this observation, it is immediate to conclude that also C is a semilattice, but it is not necessarily freely generated, i.e., it is not necessarily a powerset. Similarly, for GPS $B_{\mathcal{G}}^{A^*}$ carries a positive convex algebra structure (these are the $\mathcal{D}_\omega$-algebras) and $[\![-]\!]\colon \mathcal{D}_\omega X \to B_{\mathcal{G}}^{A^*}$ is a positive convex algebra homomorphism. Again, from this observation, we know that also C is a positive convex algebra (not necessarily freely generated).

4.5 Recovering the spectrum

We will briefly explain how to recover the spectrums from Figure 4.1 from the coalgebraic modelling. First, we recall the following folklore result from coalgebra theory which is the key behind building the spectrum. Let $\mathbf{Coalg}_f(\mathcal{F})$ denote the category of all $\mathcal{F}$-coalgebras with a free carrier (arising from a powerset construction) and $\mathcal{F}$-homomorphisms. That is, the objects are of the form $T(X) \to \mathcal{F}T(X)$. Given two functors $\mathcal{F}$ and $\mathcal{G}$, if one can construct a functor $\sigma\colon \mathbf{Coalg}_f(\mathcal{F}) \to \mathbf{Coalg}_f(\mathcal{G})$ then $\sim_{\mathcal{F}} \subseteq \sim_{\mathcal{G}}$.
In the current setting, we apply this to the category $\mathbf{Coalg}_f(\mathcal{F})$ of all $\mathcal{F}$-coalgebras with a free carrier (arising from a powerset construction) and $\mathcal{F}$-homomorphisms. That is, the objects are of the form $T(X) \to \mathcal{F}T(X)$.
For all the relations in the spectrum we can indeed define such σ. We illustrate here the case for failure and complete trace.

$$(\mathcal{P}_\omega(X) \xrightarrow{\langle o_{\mathcal{F}}, t \rangle} \mathcal{P}_\omega(\mathcal{P}_\omega(A^*)) \times \mathcal{P}_\omega(X)^A\,) \overset{\sigma}{\mapsto} (\mathcal{P}_\omega(X) \xrightarrow{\langle o_{\mathcal{CT}}, t \rangle} 2 \times \mathcal{P}_\omega(X)^A)$$

In order to prove that σ is a functor we need to show that it preserves homomorphisms.

4.5.1 Lemma. *Consider $f\colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(Y)$ such that $o_{\mathcal{F}} = o_{\mathcal{F}} \circ f$. Then $o_{\mathcal{CT}} = o_{\mathcal{CT}} \circ f$.*

Proof.

$$\begin{array}{ll}
 & o_{\mathcal{F}}(S) = o_{\mathcal{F}} \circ f(S) \\
\iff & \{Z \subseteq A \mid Z \cap I(\delta(s)) = \emptyset, s \in S\} = \{Z \subseteq A \mid Z \cap I(\delta(s')) = \emptyset, s' \in f(S)\} \\
\iff & \forall_{s \in S} \exists_{s' \in f(S)}\ Z \cap I(\delta(s)) = \emptyset \iff Z \cap I(\delta(s')) = \emptyset \text{ and vice-versa.} \\
\Rightarrow & \forall_{s \in S} \exists_{s' \in f(S)}\ I(\delta(s)) = \emptyset \iff I(\delta(s')) = \emptyset \text{ and vice-versa.} \\
\iff & \bigvee_{s \in S} (I(\delta(s)) = \emptyset) = \bigvee_{s' \in f(S)} (I(\delta(s')) = \emptyset) \\
\iff & o_{\mathcal{CT}}(S) = o_{\mathcal{CT}} \circ f(S)
\end{array}$$

Note that this is different from the technique used to recover a hierarchy of probabilistic systems in [BSdV04] where injective natural transformations were defined between functor types and then it was shown that bisimilarity was reflected by these transformations. Here, the situation is different and, for several different equivalences, we have the same functor (e.g., for $\mathcal{CT}$ and $\mathcal{T}$).

In the case of the probabilistic spectrum similar proofs can be given. We illustrate it for the case of probabilistic ready and trace semantics.

$$(\mathcal{D}_\omega(X) \xrightarrow{\langle o_{\mathcal{R}_p}, t\rangle} [0,1]^{\mathcal{P}_\omega(A^*)} \times \mathcal{D}_\omega(X)^A) \quad \overset{\sigma}{\mapsto} \quad (\mathcal{D}_\omega(X) \xrightarrow{\langle o_{\mathcal{T}_p}, t\rangle} [0,1] \times \mathcal{D}_\omega(X)^A)$$

Again, in order to prove that σ is a functor we need to show that it preserves homomorphisms.

4.5.2 Lemma. *Consider $f : \mathcal{D}_\omega(X) \to \mathcal{D}_\omega(Y)$ such that $o_{\mathcal{R}_p} = o_{\mathcal{R}_p} \circ f$. Then $o_{\mathcal{T}_p} = o_{\mathcal{T}_p} \circ f$.*

Proof.

$$\begin{aligned}
o_{\mathcal{R}_p}(\varphi) = o_{\mathcal{R}_p} \circ f(\varphi) \quad &\Longleftrightarrow \quad \sum_{\substack{x \in X \\ I = I(x)}} \varphi(x) = \sum_{\substack{y \in Y \\ I = I(y)}} f(\varphi)(y), \quad \text{for all } I \subseteq A. \\
&\Rightarrow \quad \sum_{I \subseteq A} \sum_{\substack{x \in X \\ I = I(x)}} \varphi(x) = \sum_{I \subseteq A} \sum_{\substack{y \in Y \\ I = I(y)}} f(\varphi)(y) \\
&\Longleftrightarrow \quad \sum_{x \in X} \varphi(x) = \sum_{y \in Y} f(\varphi)(y) \\
&\Longleftrightarrow \quad o_{\mathcal{T}_p}(\varphi) = o_{\mathcal{T}_p} \circ f(\varphi)
\end{aligned}$$

4.6 Testing semantics

In this section we show how must and may testing [CH89, DH84, Hen88] can be modelled coalgebraically by exploiting the generalised powerset construction in the context of LTS's with internal behaviour. As we shall see, the modelling of may testing is derived based the coalgebraic characterisation of trace semantics in Section 4.1.2, in a straightforward fashion. The coalgebraic characterisation of must testing follows as an "extension to divergence" of failure semantics in Section 4.1.1.

In our approach we consider LTS's on an alphabet $A + \{\tau\}$, where τ is a special label representing *internal actions*. We write $\overset{\varepsilon}{\Rightarrow}$ to represent $\overset{\tau}{\to}^*$ the reflexive and transitive closure of $\overset{\tau}{\to}$ and, for $a \in A$, by $\overset{a}{\Rightarrow}$ we denote $\overset{\tau}{\to}^* \overset{a}{\to} \overset{\tau}{\to}^*$. For $w \in A^*$, $\overset{w}{\Rightarrow}$ is defined inductively, in the obvious way.

4.6.1 From traces to may testing

In this section we show how may testing semantics can be modelled in the coalgebraic setting.

Intuitively, may testing relates processes in terms of the observable traces (consisting of actions different from τ) they can execute, by ignoring (any number of) occurrences of the internal action τ.

Let $L(p)$ represent the set of observable traces associated with a state p of an LTS with actions in $A \cup \{\tau\}$, referred to as the *language* of p:

$$L(p) = \{w \in (A - \{\tau\})^* \mid (\exists p').\, p \overset{w}{\Rightarrow} p' \}. \tag{4.12}$$

In [CH89], an alternative characterisation may testing semantics is defined as follows.

4.6.1 Definition (May semantics [CH89]). Let x and y be two states of an LTS. We write $x \sqsubseteq_{may} y$ iff $L(x) \subseteq L(y)$. We say that x and y are *may-equivalent* ($x \sim_{may} y$) iff $x \sqsubseteq_{may} y$ and $y \sqsubseteq_{may} x$. ♣

The connection with trace semantics in Section 4.1.2 is rather obvious: both may and trace distinguish processes depending on their languages. Hence, we further provide an extension of the coalgebraic modelling of trace semantics to the context of LTS's with internal behaviour, and show it corresponds precisely the may testing as given in Definition 4.6.3.
To begin with, we model LTS's with *internal behaviour* as coalgebras $(S, t\colon S \to (\mathcal{P}_\omega S)^A)$, such that, for $x \in S$ and $a \in A$:

$$t(x)(a) = \{y \mid x \overset{a}{\Rightarrow} y\}. \tag{4.13}$$

Then, we decorate LTS's by means of a function $o\colon S \to 2$ such that, for all $x \in S$

$$o(x) = 1$$

and apply the generalised powerset construction as depicted in Figure 4.2 in Section 4.1. Similarly to the case of trace semantics, the final Moore coalgebra is 2^{A^*} – the set of languages over A. Therefore, by the definition of the transition function t in (4.13), it immediately follows that the behaviour map $[\![-]\!]$ captures precisely the languages of states in S. Namely, for all $x \in S$:

$$[\![\{x\}]\!] \cong L(x).$$

Note that 2^{A^*} carries a join semilattice structure, where identity is the empty language and join is the union of languages. Consider $\sqsubseteq$ the associated preorder. At this point, the coalgebraic modelling of may testing semantics is straightforward:

4.6.2 Theorem. *Let x and y be two states of an LTS. Then*

$$x \sqsubseteq_{may} y \textit{ iff } [\![\{x\}]\!] \sqsubseteq [\![\{y\}]\!] \textit{ and } x \sim_{may} y \textit{ iff } [\![\{x\}]\!] = [\![\{y\}]\!].$$

4.6.2 From failures to must testing

In what follows we provide a coalgebraic handling of must testing semantics [DH84, Hen88], and show the connection between our approach and the framework used for the corresponding (alternative) modelling in [CH89].
Intuitively, must testing relates processes based on the traces that do not lead to divergent states (*i.e.*, states that can engage into infinite internal computations), and a notion of non-determinism captured in terms of antichains of corresponding ready actions. By exploiting the isomorphism of antichains and downsets introduced in Section 4.1.1, it was easy to observe that must testing coincides failure semantics for LTS's without internal behaviour (as formalised in Proposition 4.6.10 later on in this section). With this intuition in mind, we provide an extension of failure semantics to the context of divergent LTS's and show it coincides with must testing semantics. The aforementioned coincidence is proven by employing a "lifting" of the isomorphism of downsets and antichains encompassing information on both the degree of non-determinism and divergence of processes.

We first recall some notations in [CH89]. The *acceptance set of x after w* is $A(x,w) = \{\{a \in A \mid x' \xrightarrow{a}\} \mid x \overset{w}{\Rightarrow} x' \wedge x' \not\xrightarrow{\tau}\}$. Intuitively, it represents the set of actions that can be fired after "maximal" executions of w from x, those that cannot be extended by some τ-labelled transitions.
The possibility of an LTS to execute τ-actions forever is referred to as *divergence*. We write $x \not\downarrow$ whenever x diverges. Dually, the convergence relation $x{\downarrow}w$ for a state x and a word $w \in A^*$ is inductively defined as follows: $x{\downarrow}\varepsilon$ iff x does not diverge and $x{\downarrow}aw'$ iff (a) $x{\downarrow}\varepsilon$ and (b) if $x \overset{a}{\Rightarrow} x'$, then $x'{\downarrow}w'$.
Given two sets $B, C \in \mathscr{P}_\omega(\mathscr{P}_\omega(A))$, we write $B \subset\subset C$ iff for all $B_i \in B$, there exists $C_i \in C$ such that $C_i \subseteq B_i$.
With these ingredients, it is possible to introduce must preorder and equivalence.

4.6.3 Definition (Must semantics [CH89]). Let x and y be two states of an LTS. We write $x \sqsubseteq_{mst} y$ iff for all words $w \in A^*$, if $x{\downarrow}w$ then $y{\downarrow}w$ and $A(y,w) \subset\subset A(x,w)$. We say that x and y are *must-equivalent* ($x \sim_{mst} y$) iff $x \sqsubseteq_{mst} y$ and $y \sqsubseteq_{mst} x$. ♣

As an example, consider the LTS's depicted below. States x_4, x_5 and y_1 are divergent. All the other states diverge for words containing the letter b and converge for words on a^*. For these words and states x, x_1, x_2, x_3 and y, the corresponding acceptance sets equal $\{\{a,b\}\}$. In particular, note that $A(x_2, \varepsilon)$ is $\{\{a,b\}\}$ and not $\{\{b\},\{a,b\}\}$. It is therefore easy to conclude that x, x_1, x_2, x_3 and y are all must equivalent.

$$\begin{tikzcd}
x \arrow[r, "a"] \arrow[rr, bend left, "b"] \arrow[d, bend right, "a"'] & x_2 \arrow[r, "b"] \arrow[d, "\tau"] & x_4 \arrow[loop right, "\tau"] & & y \arrow[loop left, "a"] \arrow[r, "b"] & y_1 \arrow[loop right, "\tau"] \\
x_1 \arrow[u, bend right, "a"'] \arrow[rr, bend right, "b"'] & x_3 \arrow[l, "a"'] \arrow[r, "b"] & x_5 \arrow[u, "\tau"'] & & &
\end{tikzcd} \tag{4.14}$$

Coalgebraic characterisation of must semantics. In what follows we show how must testing semantics can be captured in terms of coalgebras.
In order to proceed, we have to properly tackle *internal behaviour* and *divergence*. We model LTS's on $A + \{\tau\}$ in terms of coalgebras $(S, t\colon S \to (1 + \mathscr{P}_\omega S)^A)$, where $1 = \{\top\}$ is the singleton set, and for $x \in S$,

$$t(x)(a) = \top, \text{ if } x \not\downarrow a \qquad t(x)(a) = \{y \mid x \overset{a}{\Rightarrow} y\}, \text{ otherwise.} \tag{4.15}$$

Note that we use $x \not\downarrow a$ as a shorthand for $x \not\downarrow a\varepsilon$. Intuitively, a state $x \in S$ that displays divergent behaviour with respect to an action $a \in A$ is mapped to $\top$. Otherwise t computes the set of states that can be reached from x through a (by possibly performing a finite number of τ-transitions).
Similarly to failure equivalence in Section 4.1.1, we decorate the states of the LTS by means of a function $o\colon S \to 1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A))$ defined as follows:

$$o(x) = \begin{cases} \top & \text{if } x \not\downarrow \\ \bigcup_{x \xrightarrow{\tau} x'} o(x') & \text{if } x \xrightarrow{\tau} \\ \mathit{Fail}(t(x)) & \text{otherwise.} \end{cases} \tag{4.16}$$

Note that $(S, \langle o,t\rangle)$ is an FT-coalgebra for the functor $F(S) = (1+\mathscr{P}_\omega(\mathscr{P}_\omega A)) \times S^A$ and the monad $T(S) = 1+\mathscr{P}_\omega S$. T-algebras are semilattice with bottom and an extra element $\top$ acting as *top* (i.e., such that $x \sqcup \top = \top$ for all x). For any set U, $1+\mathscr{P}_\omega(U)$ carries a semilattice with bottom and top: bottom is the empty set; top is the element $\top \in 1$; $X \sqcup Y$ is defined as the union for arbitrary subsets $X, Y \in \mathscr{P}_\omega(U)$ and as $\top$ otherwise. Consequently, $1+\mathscr{P}_\omega(\mathscr{P}_\omega A)$ and $FT(S)$ carry a T-algebra structure as well. This enables the application of the generalised powerset construction (Section 2.3) associating to each FT-coalgebra $(S,\langle o,t\rangle)$ the F-coalgebra $(1+\mathscr{P}_\omega S, \langle o^\sharp, t^\sharp\rangle)$ defined for all $X \in 1+\mathscr{P}_\omega S$ as expected:

$$o^\sharp(X) = \begin{cases} \top & \text{if } X = \top \\ \bigsqcup_{x\in X} o(x) & \text{if } X \in \mathscr{P}_\omega(S) \end{cases} \qquad t^\sharp(X)(a) = \begin{cases} \top & \text{if } X = \top \\ \bigsqcup_{x\in X} t(x)(a) & \text{if } X \in \mathscr{P}_\omega(S) \end{cases}$$

Note that in the above definitions, $\sqcup$ is not simply the union of subsets (as it was the case for failure), but it is the join operation in $1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ and $1+\mathscr{P}_\omega(\mathscr{P}_\omega(S))$. Moreover, $(1+\mathscr{P}_\omega S, \langle o^\sharp, t^\sharp\rangle)$ is a Moore machine with output in $1+\mathscr{P}_\omega(\mathscr{P}_\omega A)$ and, therefore induces a function $[\![-]\!] : (1+\mathscr{P}_\omega(S)) \to (1+\mathscr{P}_\omega(\mathscr{P}_\omega A))^{A^*}$. The semilattice structure of $1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ can be easily lifted to $(1+\mathscr{P}_\omega(\mathscr{P}_\omega A))^{A^*}$: bottom, top and $\sqcup$ are defined pointwise on A^*. We denote by $\sqsubseteq_{\mathscr{M}}$ the preorder on $(1+\mathscr{P}_\omega(\mathscr{P}_\omega A))^{A^*}$ induced by this semilattice.
A result (based on the isomorphism between downsets and antichains) similar to the one for failures, in Section 4.1.1, can also be derived in a modular fashion, for the case of LTS's decorated with outputs in $1+\mathscr{P}_\omega(\mathscr{P}_\omega A)$.
As shown in Section 4.1.1, both the set of downsets $\overline{\mathscr{D}}(\mathscr{P}_\omega(A))$, and the set of antichains $\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ carry join-semillatice structures. It is easy to see that the corresponding extensions to $1+(-)$ are join-semilattices with bottom as 0, top as $\top$ (which, intuitively, plays the role of the greatest element) and $\sqcup$ extended as $\top \sqcup C = \top$ for $C \in 1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A))$, or $C \in 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$, respectively.
The isomorphism $1+i\colon 1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A)) \to 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ follows immediately from the isomorphism $i\colon \overline{\mathscr{D}}(\mathscr{P}_\omega(A)) \to \overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ in (4.4) in Section 4.1.1, by defining

$$(1+i)(\top) = \top \qquad\qquad (1+i)(F) = i(F), \quad F \neq \top.$$

In the sequel, we will exploit $1+i$ to define a "more efficient" characterisation of the function $o^\sharp\colon 1+\mathscr{P}_\omega(S) \to 1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$, also useful to prove the soundness of the coalgebraic modelling of must testing semantics (formalised in Theorem 4.6.7).
As a first step, observe that the function $o\colon S \to 1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ can be restricted to $o\colon S \to 1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A))$ (since if $x\!\downarrow$ then $o(x)$ is a downset and the union of downsets is a downset, otherwise $o(x) = \top$). In analogy with Section 4.1.1, we define $o_2\colon S \to 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$

$$o_2(x) = \begin{cases} \top & \text{if } x \not\downarrow \\ min(\cup_{x \xrightarrow{\tau} x'} \overline{o}(x')) & \text{if } x \xrightarrow{\tau} \\ \{I(t(x))\} & \text{otherwise.} \end{cases}$$

and $o_2^\sharp\colon 1+\mathscr{P}_\omega(S) \to 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ as

$$o_2^\sharp(X) = \begin{cases} o_2(x) & \text{if } X = \{x\} \text{ with } x \in S \\ \top & \text{if } X = \top \\ 0 & \text{if } X = 0 \\ min(o_2(X_1) \sqcup o_2(X_2)) & \text{if } X = X_1 \sqcup X_2 \end{cases}$$

Proposition 4.6.6 states that it is equivalent computing $o^\sharp$ or computing $o_2^\sharp$. To this aim, we need the following lemmas.

4.6.4 LEMMA. $(1+i)\circ o = o_2$

PROOF. If $x \not\downarrow$, then $o_2(x) = \top = (1+i)\circ o(x)$. If $x\downarrow$ and $x \not\xrightarrow{\tau}$, then $o_2(x) = \{I(t(x))\} = (1+i)(\mathit{Fail}(t(x))) = (1+i)\circ o(x)$. If $x\downarrow$ and $x \xrightarrow{\tau}$, then observe that $o(x) = \bigcup\{\mathit{Fail}(t(x')) \mid x \xrightarrow{\tau}{}^{*} x' \text{ and } x' \not\xrightarrow{\tau}\}$ and that

$$o_2(x) = min\left(\bigcup\{I(t(x')) \mid x \xrightarrow{\tau}{}^{*} x' \text{ and } x' \not\xrightarrow{\tau}\}\right).$$

We obtain the conclusion by the previous case and by the fact that i is a homomorphism of semilattices. □

4.6.5 LEMMA. $(1+i)\circ o^\sharp = o_2^\sharp$

PROOF. Follows immediately by Lemma 4.6.4 and the fact that $(1+i)$ is a homomorphism of semilattices. □

4.6.6 PROPOSITION. *For all $X, Y \in \mathscr{P}_\omega(S)$, $o^\sharp(X) = o^\sharp(Y)$ iff $o_2^\sharp(X) = o_2^\sharp(Y)$.*

PROOF. Follows from Lemma 4.6.5 and the fact that $(1+i)$ is an isomorphism of semilattices. □

Remark 13 *Note that the relation $\subset\subset$ used for defining $\sqsubseteq_{mst}$:*

$$B \subset\subset C \;\; \texttt{iff} \;\; (\forall B_i \in B).(\exists C_i \in C).C_i \subseteq B_i \tag{4.17}$$

is the ordering induced by $\sqcup$ in $\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$:

$$\begin{array}{lll} & min(B \sqcup C) = C & \\ \texttt{iff} & min(B \sqcup C) = min(C) & (\text{as } C \in \overline{\mathscr{A}}(\mathscr{P}_\omega(A))) \\ \texttt{iff} & (\forall B_i \in B).(\exists C_i \in C).C_i \subseteq B_i & (\text{by definition of } min) \\ \texttt{iff} & B \subset\subset C & (\text{by definition of } \subset\subset) \end{array}$$

We formalise the coalgebraic modelling of must semantics in the following theorem.

4.6.7 THEOREM. *Let x and y be two states of an LTS. Then*

$$x \sqsubseteq_{mst} y \text{ iff } [\![\{y\}]\!] \sqsubseteq_{\mathscr{M}} [\![\{x\}]\!] \text{ and } x \sim_{mst} y \text{ iff } [\![\{x\}]\!] = [\![\{y\}]\!].$$

The morphism $o_2^\sharp\colon 1+\mathscr{P}_\omega(S) \to 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ is useful to prove that the preorders $\sqsubseteq_{\mathscr{M}}$ and $\sqsubseteq_{mst}$ coincide. Indeed, the Moore machine $(1+\mathscr{P}_\omega S, \langle o_2^\sharp, t^\sharp\rangle)$ induces the morphism $[\![-]\!]_2\colon 1+\mathscr{P}_\omega S \to (1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A)))^{A^*}$ defined for all $X \in 1+\mathscr{P}_\omega(S)$ as

$$[\![X]\!]_2(\varepsilon) = o_2^\sharp(X) \qquad [\![X]\!]_2(aw) = [\![t^\sharp(X)(a)]\!]_2(w).$$

The isomorphism $(1+i)\colon 1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A)) \to 1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A))$ can be extended to the isomorphism $(1+i)^{A^*}\colon (1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A)))^{A^*} \to (1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A)))^{A^*}$, defined for every function $\phi \in (1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A)))^{A^*}$ and word $w \in A^*$ as

$$(1+i)^{A^*}(\phi)(w) = (i+1)(\phi(w)).$$

Note that the function $[\![-]\!]\colon 1+\mathscr{P}_\omega(S) \to (1+\mathscr{P}_\omega\mathscr{P}_\omega(A))^{A^*}$ can be restricted to $[\![-]\!]\colon 1+\mathscr{P}_\omega(S) \to (1+\overline{\mathscr{D}}(\mathscr{P}_\omega(A)))^{A^*}$.

4.6.8 PROPOSITION. $(1+i)^{A^*} \circ [\![-]\!] = [\![-]\!]_2$

PROOF. This can be proved by ordinary induction on words, exploiting Lemma 4.6.5 for the base case. □

The ordering $\sqsubseteq_{\mathcal{M}_2}$ induced by the semilattice structure of $(1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A)))^{A^*}$ is given as follows: for all $\phi, \psi \in (1+\overline{\mathscr{A}}(\mathscr{P}_\omega(A)))^{A^*}$, $\phi \sqsubseteq_{\mathcal{M}_2} \psi$ iff for all $w \in A^*$

1. if $\phi(w) = \top$ then $\psi(w) = \top$ and
2. if $\phi(w) \neq \top$ then either $\psi(w) = \top$ or $\phi(w) \subset\subset \psi(w)$.

Observe that $[\![X]\!]_2(w) = \top$ iff $X \not\downarrow w$. Furthermore, whenever $X = \{x\}$ and $x{\downarrow}w$, $[\![X]\!]_2(w) = A(x,w)$. As a consequence, the following proposition holds.

4.6.9 PROPOSITION. $[\![\{x\}]\!]_2 \sqsubseteq_{\mathcal{M}_2} [\![\{y\}]\!]_2$ *iff* $y \sqsubseteq_{mst} x$

PROOF. Suppose that $[\![\{x\}]\!]_2 \sqsubseteq_{\mathcal{M}_2} [\![\{y\}]\!]_2$ and take one word $w \in A^*$. If $y{\downarrow}w$, then $[\![\{y\}]\!]_2(w) \neq \top$ and also $[\![\{x\}]\!]_2(w) \neq \top$, that is $x{\downarrow}w$. This means that $[\![\{x\}]\!]_2(w) \subset\subset [\![\{y\}]\!]_2(w)$, that is $A(x,w) \subset\subset A(y,w)$. summarising $y \sqsubseteq_{mst} x$.
Now suppose that $y \sqsubseteq_{mst} x$ and take one word $w \in A^*$. If $[\![\{x\}]\!]_2(w) = \top$, then $x \not\downarrow w$. This implies that also $y \not\downarrow w$ (and thus $[\![\{y\}]\!]_2(w) = \top$) because otherwise the hypothesis $y \sqsubseteq_{mst} x$ would be violated. If $[\![\{x\}]\!]_2(w) \neq \top$, then we have two possibilities: (a) $y{\downarrow}w$ or (b) $y \not\downarrow w$. For (a), we have that $A(x,w) \subset\subset A(y,w)$, that is $[\![\{x\}]\!]_2(w) \subset\subset [\![\{y\}]\!]_2(w)$. For (b), we immediately have that $[\![\{y\}]\!]_2 = \top$. □

From the two above propositions, Theorem 4.6.7 follows immediately.
Note that in absence of divergence, the "decorating" function in (4.16) and the transition function in (4.15) correspond precisely to $\overline{o}_{\mathscr{F}}$ and δ in Section 4.1.1, for the case of failure semantics. Hence, by Theorem 4.6.7, Definition 4.6.3 and Remark 13 it follows immediately that must and failure semantics coincide in the context of LTS's without internal behaviour.

4.6.10 PROPOSITION. *Consider two states x, y of an LTS without internal behaviour. Then*

$$x \sqsubseteq_{mst} y \textit{ iff } \mathscr{F}(y) \subseteq \mathscr{F}(x)$$
$$x \sim_{mst} y \textit{ iff } \mathscr{F}(x) = \mathscr{F}(y).$$

Remark 14 *Note that according to the definition of $\sqsubseteq_{\mathcal{M}}$, $[\![\{y\}]\!] \sqsubseteq_{\mathcal{M}} [\![\{x\}]\!]$ iff $[\![\{y\}]\!] \sqcup [\![\{x\}]\!] = [\![\{x\}]\!]$, and since $[\![-]\!]$ is a T-homomorphism (namely it preserves bottom, top and $\sqcup$), the latter equality holds iff $[\![\{y,x\}]\!] = [\![\{x\}]\!]$. Summarising,*

$$x \sqsubseteq_{mst} y \textit{ iff } [\![\{x,y\}]\!] = [\![\{x\}]\!].$$

Consider, once more, the LTS in (4.14). The part of the Moore machine $(1+\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp \rangle)$ which is reachable from $\{x\}$ and $\{y\}$ is depicted below (the output function $o^\sharp$ maps $\top$ to $\top$ and the other states to $\{0\}$).

$$\begin{tikzcd}
\top \arrow[loop above, "{a,b}"] \arrow[d, dash, dashed] & \{x\} \arrow[l, "b"] \arrow[r, "a"] \arrow[d, dash] & \{x_1,x_2,x_3\} \arrow[r, "a"] \arrow[ll, bend right, "b"'] & \{x,x_1\} \arrow[r, "a"] \arrow[lll, bend right, "b"'] & \{x,x_1,x_2,x_3\} \arrow[loop above, "a"] \arrow[llll, bend right, "b"'] \\
\top \arrow[loop below, "{a,b}"] & \{y\} \arrow[l, "b"] \arrow[loop below, "a"] \arrow[ur, dash, dashed] \arrow[urr, dash, dashed] \arrow[urrr, dash, dashed] & & &
\end{tikzcd} \tag{4.18}$$

The relation consisting of dashed lines is a bisimulation proving that $[\![\{x\}]\!] = [\![\{y\}]\!]$, i.e., that $x \sim_{mst} y$.
Our construction is closely related to the one in [CH89], that transforms LTS's into (deterministic) acceptance graphs. We further provide more details on the connection between the coalgebraic machinery for reasoning on must preorder and the corresponding framework in [CH89].

Moore machines and acceptance graphs. As previously introduced in this section, the behaviour of an LTS with divergence $\mathscr{L} = (S, t : S \to (1 + \mathscr{P}_\omega S)^A)$ can be captured in terms of a Moore machine

$$\mathscr{M} = (1 + \mathscr{P}_\omega S, \langle o^\sharp, t^\sharp\rangle : 1 + \mathscr{P}_\omega S \to (1 + \mathscr{P}_\omega(\mathscr{P}_\omega A)) \times (1 + \mathscr{P}_\omega S)^A)$$

derived according to the powerset construction, and that reasoning on must preorder is equivalent to reasoning on the preorder $\sqsubseteq_{\mathscr{M}}$ on the final Moore coalgebra, as stated in Theorem 4.6.7.
In [CH89] must preorder is established in terms of a notion of prebisimulation ($\sqsubseteq_{\langle \Pi, 0\rangle}$) on the so-called "acceptance graphs" generated from such $\mathscr{L}$'s, denoted by $ST(\mathscr{L})$. Intuitively, an acceptance graph $ST(\mathscr{L})$ consists of a set of nodes p of shape $\langle Q, b\rangle \in \mathscr{P}_\omega S \times \{\mathbf{tt}, \mathbf{ff}\}$, where Q is a set of states in S, and b is associated the boolean value **tt** whenever all states in Q converge (written $Q\downarrow$) and **ff** otherwise.
Orthogonally to the Moore machines with output in $1 + \mathscr{P}_\omega(\mathscr{P}_\omega A)$, for a node $p = \langle Q, b\rangle$ in $ST(\mathscr{L})$, the information representing the divergence of (states in) Q is given by $p.closed (= b)$, and the corresponding (minimised) acceptance set consisting of visible actions that can be triggered as a first step from the states in Q is represented by $p.acc$ (defined later on in this section). Moreover, (deterministic) transitions in $ST(\mathscr{L})$ are of shape $\langle Q_1, b_1\rangle \xrightarrow{a} \langle Q_2, b_2\rangle$, where $a \in A$ and Q_2 is the set of a-successors of states in Q_1, computed with respect to $\overset{a}{\Rightarrow}$.
Based on the resemblance between the aforementioned Moore machines and acceptance graphs, we consider worth investigating to what extent these constructions and the corresponding "alternative" semantics used for reasoning on must preorder are connected.
In what follows we recall the formal definition of acceptance graphs as introduced in [CH89], show they are isomorphic (up-to divergent behaviours) with the Moore machines used for the coalgebraic modelling of must semantics.
We proceed by first providing the basic ingredients needed for the definition of acceptance graphs.
Consider $Q \in \mathscr{P}_\omega S$. The *$\varepsilon$-closure* of a Q is $Q^\varepsilon = \{p \mid q \overset{\varepsilon}{\Rightarrow} p \wedge q \in Q\}$. The set of *direct a-successors* of states $q \in Q$ is $D(Q, a) = \{q' \mid q \xrightarrow{a} q' \wedge q \in Q\}$, where $a \in A \cup \{\tau\}$.

4.6.11 Definition (Acceptance graphs [CH89]). Consider $\mathscr{L}$ an LTS with divergence, with state space S and visible actions labelled in A. The corresponding *acceptance graph* $ST(\mathscr{L}) = (T, A \cup \{\tau\}, \to)$ is defined as follows.

1. $T = \{\langle Q, b\rangle \in \mathscr{P}_\omega S \times \{\mathbf{tt}, \mathbf{ff}\} \mid Q = Q^\varepsilon \wedge (b = \mathbf{tt} \Rightarrow Q\downarrow)\}$.
2. For $p = \langle Q, b\rangle \in T$ define $p.closed = b$ and

$$p.acc = \begin{cases} 0 & \text{if } p.closed = \mathbf{ff} \\ min(\{\{a \in A \mid q \xrightarrow{a}\} \mid q \in Q \wedge q \not\xrightarrow{\tau}\}) & \text{otherwise.} \end{cases}$$

(We refer to (4.3) in Section 4.1.1 for the definition of min.)

3. A transition $\langle Q_1, b_1\rangle \xrightarrow{a}_T \langle Q_2, b_2\rangle$ is performed exactly when the following hold: $a \neq \tau$, $D(Q_1,a)^\varepsilon = Q_2$, and $b_1 = \mathbf{tt} \wedge (Q_2{\downarrow} \Rightarrow b_2 = \mathbf{tt})$. ♣

It is worth observing that, according to Definition 4.6.11, acceptance graphs are deterministic, and moreover, there are no outgoing transitions from divergent states (see (c) above). Considering graphs satisfying the latter property comes as a natural consequence of the fact that the must preorder considers divergence catastrophic, as can be inferred from Definition 4.6.3.

Given an LTS with divergence $\mathscr{L}$, and q a state of $\mathscr{L}$, the node in $ST(\mathscr{L})$ corresponding to q is $\langle \{q\}^\varepsilon, q{\downarrow}\varepsilon\rangle$. Orthogonally, the state corresponding to q in the Moore machine derived according to the powerset construction is $\{q\}$.

For an example, consider the following LTS:

$$\begin{tikzcd}
q_4 \arrow[loop below, "\tau"] \arrow[rr, bend right, "c"'] & q_2 \arrow[l, "b"'] \arrow[dr, "c"] & q_1 \arrow[l, "a"'] \arrow[r, "\tau"] \arrow[rr, bend right, "a"] & q_{10} & q_3 \arrow[r, "b"] \arrow[d, "\tau"'] & q_7 \\
 & & q_5 & q_8 & q_6 \arrow[l, "a"'] \arrow[r, "c"] & q_9
\end{tikzcd}$$

The associated Moore determinisation $\mathscr{M}$ when starting from q_1 and the corresponding acceptance graph $ST(\mathscr{L})$, respectively, are illustrated as follows.

$$\begin{tikzcd}
\mathscr{M}: & & \{q_1\} \arrow[r, Rightarrow] \arrow[d, "a"] & {[abc]} \\
 & & \{q_2,q_3,q_6\} \arrow[r, ">"] \arrow[dl, "a"'] \arrow[d, "b"] \arrow[dr, "c"] & {[a]\cup[b]} \\
 & \{q_8\} \arrow[d, Rightarrow] & \top \arrow[loop right, "{a,b,c}"] \arrow[d, Rightarrow] & \{q_5,q_9\} \arrow[d, Rightarrow] \\
 & {[abc]} & \top & {[abc]}
\end{tikzcd}
\qquad
\begin{tikzcd}
ST(\mathscr{L}): & & \langle\{q_1,q_{10}\},\mathbf{tt}\rangle \arrow[r, dashed] \arrow[d, "a"] & 0 \\
 & & \langle\{q_2,q_3,q_6\},\mathbf{tt}\rangle \arrow[r, ">"] \arrow[dl, "a"'] \arrow[d, "b"] \arrow[dr, "c"] & \{\{b,c\},\{a,c\}\} \\
 & \langle\{q_8\},\mathbf{tt}\rangle \arrow[d] & \langle\{q_4,q_7\},\mathbf{ff}\rangle \arrow[d] & \langle\{q_5,q_9\},\mathbf{tt}\rangle \arrow[d] \\
 & 0 & 0 & 0
\end{tikzcd}$$

Recall from Section 4.1.1 that for the simplicity of notation we write, for example, $[abc]$ in order to denote the powerset of $\{a,b,c\}$. In $ST(\mathscr{L})$, the notation $\langle Q,b\rangle \dashrightarrow B$ represents a node $p = \langle Q,b\rangle$ such that $p.acc = B$ and $p.closed = b$.

Observe that: both $\mathscr{M}$ and $ST(\mathscr{L})$ are deterministic, transitions starting from divergent states $\top$ in $\mathscr{M}$ always produce output $\top$, whereas in $ST(\mathscr{L})$ divergent nodes $p = \langle Q^\varepsilon, \mathbf{ff}\rangle$ are deadlock-like and, moreover, $p.acc = 0$.

Given an LTS $\mathscr{L}$ with state space S, the connection between non-divergent nodes Q in the corresponding Moore machine $\mathscr{M} = (1 + \mathscr{P}_\omega S, \langle o^\sharp, t^\sharp\rangle)$ and those in the associated acceptance graph $ST(\mathscr{L})$ is obvious. Each such Moore state Q corresponds to a node $p = \langle Q^\varepsilon, \mathbf{tt}\rangle$ in the acceptance graph such that $p.acc = i(o^\sharp(Q))$, where i is the isomorphism between downsets and antichains defined in Section 4.1.1.

For example, state $\{q_1\}$ in $\mathscr{M}$ is in one to one correspondence with $p = \langle \{q_1^\varepsilon\} = \{q_1, q_{10}\}, \mathbf{tt}\rangle$ in $ST(\mathscr{L})$, and, moreover:

$$\begin{aligned} i(o^\sharp(\{q_1\})) &= i(F = \{0,\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}) \\ &= min(\cup_{F_i \in F}\{A - F_i\}) = 0 = p.acc. \end{aligned}$$

As already hinted, a divergent set of states Q is represented by $\top$ in the Moore machine derived from an LTS, and it corresponds to a node $p = \langle Q^\varepsilon, \mathbf{ff}\rangle$ in the associated acceptance

graph, such that p has no outgoing transitions and $p.acc = 0$. For this case we refer to the states $Q = \{q_4, q_7\}$ in $\mathscr{L}$.
An important remark is that divergent nodes and their successors in the Moore machines can safely be ignored when reasoning on $\sqsubseteq_{\mathscr{M}}$. This follows as a consequence of:

$$\begin{array}{ll} & [\![X]\!] \sqsubseteq_{\mathscr{M}} [\![Y]\!] \\ \text{iff} & (\forall w \in A^*).\, [\![X]\!](w) \sqsubseteq [\![Y]\!](w) \\ \text{iff} & (\forall w \in A^*).\, Y\!\downarrow\, \Rightarrow (X\!\downarrow \wedge\ [\![X]\!](w) \sqsubseteq [\![Y]\!](w)) \\ & (\text{as if } X \not\downarrow w \text{ then } [\![X]\!](w) = \top, \text{ which follows by induction on } w \in A^*) \end{array} \tag{4.19}$$

for all $X, Y \in 1 + \mathscr{P}_\omega S$, where S is the state space of the LTS and A is the corresponding action alphabet.
Hence, the corresponding subsequent transitions $\top \Leftarrow \top \circlearrowleft a$ can be ignored as well, for all $a \in A$.
As a last ingredient in showing the connection between the Moore machine and the acceptance graph associated with an LTS with divergence, we make the following observations. Transitions $o_1 \Leftarrow Q_1 \xrightarrow{a} Q_2 \Rightarrow o_2$ between non-divergent states Q_1, Q_2 correspond to transitions $p_1 = \langle Q_1^\varepsilon, \mathbf{tt}\rangle \xrightarrow{a} \langle Q_2^\varepsilon, \mathbf{tt}\rangle = p_2$ such that $p_i.acc = i(o_i)$, for $i \in \{1,2\}$. Each transition $o_1 \Leftarrow Q_1 \xrightarrow{a} \top \Rightarrow \top$ with Q_1 a non-divergent state matches a transition $p_1 = \langle Q_1^\varepsilon, \mathbf{tt}\rangle \xrightarrow{a} p_2$ such that $p_1.acc = i(o_1)$, $p_2.closed = \mathbf{ff}$ and $p_2.acc = 0$.
At this point we conclude that, given an LTS with divergence $\mathscr{L}$, the Moore machine derived according to the powerset construction and the corresponding acceptance graph $ST(\mathscr{L})$ are isomorphic up-to divergent behaviours.

4.7 Discussion

In this chapter, we have proved that the coalgebraic characterisations of decorated trace semantics for labelled transition systems and generative probabilistic systems, respectively, are equivalent with the corresponding standard definitions in [vG01a] and [JS90]. More precisely, we have shown that for a state x, the coalgebraic canonical representative $[\![\{x\}]\!]$, given by determinisation and finality, coincides with the classical semantics $\mathscr{I}(x)$, for $\mathscr{I}$ ranging over $\mathscr{T}, \mathscr{CT}, \mathscr{R}, \mathscr{F}, \mathscr{PF}, \mathscr{RT}$ and $\mathscr{FT}$, representing the traces, complete traces, ready pairs, failure pairs, possible futures, ready traces and, respectively, failure traces of x in a labelled transition system. Similar equivalences have been proven for $\mathscr{I}$ ranging over $\mathscr{R}_p, \mathscr{F}_p, \mathscr{MF}_p, \mathscr{T}_p$ and $\mathscr{MT}_p$ representing the ready, failure, maximal failure, trace and maximal trace functions for the case of probabilistic systems.
We also showed that the spectrum of decorated trace semantics can be recovered from the coalgebraic modelling.
Moreover, we provided an extension of trace and failure semantics to the context of labelled transition systems with internal behaviour, which further enabled the coalgebraic modelling of may and must testing semantics in [CH89] via the generalised powerset construction. A similar idea of system determinisation was also applied in [CH89], in a non-coalgebraic setting where, in the absence of internal actions and divergence, respectively, may testing coincides with trace and must testing coincides with failure semantics, respectively. The connection with this work is also studied in this chapter.

In addition, we have illustrated how to reason about decorated trace and testing semantics using coinduction, by constructing suitable Moore bisimulations. This is a sound and complete proof technique, and represents an important step towards automated reasoning, as it opens the way for the use of, for instance, coinductive theorem provers such as CIRC [RL09].

Chapter 5

Algorithms for decorated trace and testing semantics

In Chapter 4 we provided a coalgebraic handling of a suite of semantics for different types of systems. These consist of decorated trace semantics for labelled transition systems and generative probabilistic systems, and may/must testing semantics for labelled transition systems with internal behaviour. In this chapter we focus on deriving algorithms for reasoning on failure and must testing, but our considerations hold also for the other decorated trace semantics for LTS's in Chapter 4, and for may testing semantics.

The problem of automatically checking these notions of behavioural equivalence is usually reduced to the problem of checking bisimilarity, as implemented in several tools [CPS93b, CS96, CDLT08, CGK$^+$13] and proposed in [CH89] which introduces a procedure for checking testing equivalences. The idea is the following. First, non-deterministic systems, represented by labelled transition systems (LTS's), are transformed into deterministic "acceptance graphs" with a construction which is reminiscent of the *determinisation* of non-deterministic automata (NDA's). Then, since bisimilarity in acceptance graphs coincides with testing equivalence in the original LTS's, one checks bisimilarity via the so-called *partition refinement* algorithm [KS83, PT87]. Such algorithm, which is the best-known for minimising LTS's with respect to bisimilarity, is analogous to Hopcroft's minimisation algorithm [Hop71] for deterministic automata (DA's) with respect to language equivalence. In both, a partition of the state space is iteratively refined until the largest fixed-point is reached. In a nutshell, the procedure for checking testing semantics adopted in [CH89] is in essence the same as the classical procedure for checking language equivalence of non-deterministic automata: first determinise and then compute a largest fixed-point.

This observation led us to experiment with applying other interesting language equivalence algorithms, not available for bisimilarity, to solve the problem of checking must and failure semantics. In order to achieve this, we took a coalgebraic perspective of the problem at hand, which allowed us to study the constructions and the semantics in a uniform fashion. The abstract coalgebraic framework enabled a unified study of different kinds of state based systems: (a) both the determinisation of NDA's and the construction of acceptance graphs in [CH89] are instances of the generalised powerset construction [CHL03, Len99, SBBR10], and (b) the iterations of both the Hopcroft and the partition refinement algorithms are in one-to-one correspondence with the so-called construction of the terminal sequence [AK95, Wor05]. While (b) is well-known in the community of coalgebras [ABH$^+$12, FME05, Kur00, Sta11], (a) is the key observation of

this work, which enabled us to devise other algorithms for must and failure semantics (introduced in Section 4.1.1 and Section 4.6.2, respectively).
First, we consider *Brzozowski's algorithm* [Brz62] which transforms an NDA into the minimal deterministic automaton accepting the same language: the input automaton is reversed (by swapping final and initial states and reversing its transitions), determinised, reversed and determinised once more. This somewhat intriguing algorithm can be explained in terms of duality and coalgebras [BBRS12, BPK12]. In particular, the approach in [BBRS12] allows us to extend it to Moore machines, which paves the way to adapt Brzozowski's algorithm for checking testing semantics.
Next, we consider several more efficient algorithms that have been recently introduced in a series of papers [ACH$^+$10, BP13, DR10, WDHR06]. These algorithms rely on different kinds of *(bi)simulations up-to*, which are proof techniques originally proposed for process calculi [Mil89, MPW92, San98]. From these algorithms, we choose the one in [BP13] (HKC), which can be easily proved correct using coalgebraic techniques. HKC can be easily adapted to check must testing, once a coalgebraic characterisation of must equivalence is given.
Comparing the efficiency of these three families of algorithms (partition refinement [CH89], Brzozowski and bisimulations up-to) is not a trivial task. Both the problems of checking language and testing equivalence are PSPACE-complete as shown in [MS73] and [KS83], respectively. However, in both cases, the theoretical complexity appears not to be problematic in practice, so that an empirical evaluation is more desirable. In [TV05, Wat95, Wat00], experiments have shown that Brzozowski's algorithm performs better than Hopcroft's one for "high-density" NDA's, while Hopcroft's algorithm is more efficient for generic NDA's. Both algorithms appear to be rather inefficient compared to those of the new generation [ACH$^+$10, BP13, DR10, WDHR06]. It is out of the scope of this work to present an experimental comparison of the adaptation of these algorithms for must equivalence; we confine our results to showing that each approach can be more efficient than the others on concrete examples.
summarising, the main contributions of this chapter are:

- The adaptation of HKC and Brzozowski's algorithm for failure and must semantics. For the latter, this includes an optimisation which avoids an expensive determinisation step. All the observations for failure can be used for various other decorated trace semantics, such as ready and ready trace.
- An interactive applet[1] allowing one to experiment with these algorithms.
- Experiments checking the equivalence of an ideal and a distributed multiway synchronisation protocol [PS96].
- At a more conceptual level, the present work also shows that the coalgebraic analysis of systems yields not only a good mathematical theory of their semantics but also a rich playground to devise algorithms.

Organisation of the chapter. We first recall the word automata, the algorithms we will start with, and their coalgebraic description (Sect. 5.1). We adapt these algorithms to failure semantics (Sections 5.2.1, 5.2.3, 5.2.4, 5.2.5), and then to must semantics (Sections 5.2.2, 5.2.3, 5.2.6, 5.2.7) for finite machines: although failure semantics can be seen as a special case of must semantics, the first generalisation is important for the sake

[1]`http://perso.ens-lyon.fr/damien.pous/brz`

of clarity. We finally give examples illustrating the relative behaviour of the various algorithms (Sections 5.3, 5.4), before concluding (Section 5.5).

5.1 Language equivalence

The core of this chapter is about the problem of checking whether two states in a finite transition system are behavioural equivalent, for a certain notion of equivalence. More explicitly, we will reduce the problem of reasoning on failure and must testing semantics, respectively, to the classical problem of checking language equivalence.

We proceed by first providing a short overview on deterministic automata (DA's), Moore machines and non-deterministic automata (NDA's), and the problem of recovering language semantics of NDA's, in the coalgebraic setting.

We recall again that a *deterministic automaton* over the input alphabet A is a pair $(S, \langle o, t\rangle)$, where S is a set of states and $\langle o, t\rangle : S \to 2 \times S^A$ is a function with two components: o, the output function, determines whether a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each state and each input letter, the next state.

From any DA, there exists a function $[\![-]\!] : S \to 2^{A^*}$ mapping states to formal languages, defined as follows, for all $x \in S$:

$$[\![x]\!](\varepsilon) = o(x) \qquad [\![x]\!](a \cdot w) = [\![t(x)(a)]\!](w) \tag{5.1}$$

The language $[\![x]\!]$ is called the language accepted by x, and it consists of all words $w \in A^*$ which, if executed from x, lead to a final (or accepting) state. Given an automaton $(S, \langle o, t\rangle)$, the states $x, y \in S$ are said to be *language equivalent* iff they accept they same language.

Throughout this chapter, we will use *Moore machines* which are coalgebras for the functor $F(S) = B \times S^A$. These are very similar to DA's, but with outputs in any (fixed) set B. The unique F-homomorphism to the final coalgebra $[\![-]\!] : S \to B^{A^*}$ is defined exactly as for DA's by the equations in (5.1). Note that the behaviours of Moore machines are functions $\varphi : A^* \to B$, rather than subsets of A^*. For each behaviour $\varphi \in B^{A^*}$, there exists a minimal Moore machine realising it.

A *non-deterministic automaton* is similar to a DA but the transition function returns a set of next-states instead of a single state. Thus, an NDA over the input alphabet A is a pair $(S, \langle o, t\rangle)$, where S is a set of states and $\langle o, t\rangle : S \to 2 \times (\mathscr{P}_\omega(S))^A$. An example is depicted below (final states are overlined, labelled edges represent transitions).

$$\begin{tikzcd}
x & z \arrow[l, "a"'] \arrow[r, bend left, "a"] & \overline{y} \arrow[l, bend left, "a"] & & u \arrow[r, bend left, "a"] \arrow[rr, bend right, "a"'] & w \arrow[l, bend left, "a"] & \overline{v} \arrow[l, "a"']
\arrow[from=1-1, to=1-3, bend right, "a"']
\end{tikzcd} \tag{5.2}$$

Classically, in order to recover language semantics of NDA, one uses the *powerset construction* (see Section 2.3 for a reminder), transforming every NDA $(S, \langle o, t\rangle)$ into the DA $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ where $o^\sharp : \mathscr{P}_\omega(S) \to 2$ and $t^\sharp : \mathscr{P}_\omega(S) \to \mathscr{P}_\omega(S)^A$ are defined for all $X \in \mathscr{P}_\omega(S)$ as

$$o^\sharp(X) = \bigsqcup_{x \in X} o(x) \qquad\qquad t^\sharp(X)(a) = \bigsqcup_{x \in X} t(x)(a) \ .$$

Note that we use $\sqcup$ to denote the "Boolean or" in 2, the union of languages in 2^{A^*} and the union of sets in $\mathscr{P}_\omega(S)$.

For instance with the NDA from (5.2), $o^\sharp(\{x,y\}) = 0 \sqcup 1 = 1$ (i.e., the state $\{x,y\}$ is final) and $t^\sharp(\{x,y\})(a) = \{y\} \sqcup \{z\} = \{y,z\}$ (i.e., $\{x,y\} \xrightarrow{a} \{y,z\}$).
Since $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ is a deterministic automaton, we can now apply the language semantics above, yielding a function $[\![-]\!] : \mathscr{P}_\omega(S) \to 2^{A^*}$ mapping *sets* of states to languages. Given two states x and y, we say that they are language equivalent iff $[\![\{x\}]\!] = [\![\{y\}]\!]$. More generally, for two sets of states $X, Y \subseteq S$, we say that X and Y are language equivalent iff $[\![X]\!] = [\![Y]\!]$.
In order to introduce the algorithms in full generality, it is important to recall here that the sets 2, $\mathscr{P}_\omega(S)$, $\mathscr{P}_\omega(S)^A$, $2 \times \mathscr{P}_\omega(S)^A$ and 2^{A^*} carry a semilattice with bottom structure $(X, \sqcup, 0)$ and that the functions $\langle o^\sharp, t^\sharp\rangle : \mathscr{P}_\omega(S) \to 2 \times \mathscr{P}_\omega(S)^A$ and $[\![-]\!] : \mathscr{P}_\omega(S) \to 2^{A^*}$ are homomorphisms of semilattices with bottom. In the rest of the chapter we will indiscriminately use 0 to denote the element $0 \in 2$, the empty language in 2^{A^*} and the empty set in $\mathscr{P}_\omega(S)$.

5.1.1 Language equivalence via bisimulation up-to: HKC

We recall the algorithm HKC from [BP13]. We first provide a notion of bisimulation on sets of states, underlying the notion of progression. Note that this is equivalent to the bisimulation introduced in Section 2.2, but more appropriate for the proofs in this chapter.

5.1.1 Definition (Progression, Bisimulation). Given two relations $R, R' \subseteq \mathscr{P}_\omega(S) \times \mathscr{P}_\omega(S)$, *$R$ progresses to R'*, denoted $R \rightarrowtail R'$, if whenever $X \mathrel{R} Y$ then

$$1.\ o^\sharp(X) = o^\sharp(Y) \quad \text{and} \quad 2.\ \text{for all } a \in A,\ t^\sharp(X)(a) \mathrel{R'} t^\sharp(Y)(a).$$

A *bisimulation* is a relation R such that $R \rightarrowtail R$. ♣

This definition considers the states, the transitions and the outputs of the *determinised* NDA. For this reason, the bisimulation proof technique is sound and complete for language equivalence.
Consequently, the coinduction proof principle is stated as follows.

5.1.2 Proposition (Coinduction). *For all $X, Y \in \mathscr{P}_\omega(S)$, $[\![X]\!] = [\![Y]\!]$ iff there exists a bisimulation that relates X and Y.*

For an example, suppose that we want to prove the equivalence of $\{x\}$ and $\{u\}$ of the NDA in (5.2). The part of the determinised NDA that is reachable from $\{x\}$ and $\{u\}$ is depicted below. The relation consisting of dashed and dotted lines is a bisimulation which proves that $[\![\{x\}]\!] = [\![\{u\}]\!]$.

$$\begin{tikzcd}
\{x\} \arrow[r, "a"] \arrow[d, dash, dashed, "1"'] & \overline{\{y\}} \arrow[r, "a"] \arrow[d, dash, dashed, "2"'] & \{z\} \arrow[r, "a"] \arrow[d, dash, dashed, "3"'] & \overline{\{x,y\}} \arrow[r, "a"] \arrow[d, dash, dotted] & \overline{\{y,z\}} \arrow[r, "a"] \arrow[dl, dash, dotted] & \overline{\{x,y,z\}} \arrow[loop below, "a"] \arrow[dll, dash, dotted] \\
\{u\} \arrow[r, "a"'] & \overline{\{v,w\}} \arrow[r, "a"'] & \{u,w\} \arrow[r, "a"'] & \overline{\{u,v,w\}} \arrow[loop right, "a"] & &
\end{tikzcd} \tag{5.3}$$

The dashed lines (numbered by 1, 2, 3) form a smaller relation which is not a bisimulation, but a *bisimulation up-to congruence*: the equivalence of $\{x,y\}$ and $\{u,v,w\}$ can be immediately deduced from the fact that $\{x\}$ is related to $\{u\}$ and $\{y\}$ to $\{v,w\}$. In order to formally introduce bisimulations up-to congruence, we need to define first the *congruence* *closure* $c(R)$ of a relation $R \subseteq \mathscr{P}_\omega(S) \times \mathscr{P}_\omega(S)$. This is done inductively, by the following rules:

$$\frac{X\,R\,Y}{X\;c(R)\;Y} \quad \frac{}{X\;c(R)\;X} \quad \frac{X\;c(R)\;Y}{Y\;c(R)\;X} \quad \frac{X\;c(R)\;Y \quad Y\;c(R)\;Z}{X\;c(R)\;Z} \quad \frac{X_1\;c(R)\;Y_1 \quad X_2\;c(R)\;Y_2}{X_1 \sqcup X_2\;c(R)\;Y_1 \sqcup Y_2} \tag{5.4}$$

Note that the term "congruence" here is intended with respect to the semilattice structure carried by the state space $\mathscr{P}_\omega(S)$ of the determinised automaton. Intuitively, $c(R)$ is the smallest equivalence relation containing R and which is closed w.r.t $\sqcup$.

5.1.3 Definition (Bisimulation up-to congruence). A relation $R \subseteq \mathscr{P}_\omega(S) \times \mathscr{P}_\omega(S)$ is a *bisimulation up-to c* if $R \rightarrowtail c(R)$, i.e., whenever $X\,R\,Y$ then

$$1.\ o^\sharp(X) = o^\sharp(Y) \quad \text{and} \quad 2.\ \text{for all } a \in A,\ t^\sharp(X)(a)\;c(R)\;t^\sharp(Y)(a).$$ ♣

5.1.4 Theorem ([BP13]). *Any bisimulation up-to c is contained in a bisimulation.*

Figure 5.1 shows the corresponding algorithm, parametric on $o^\sharp$, $t^\sharp$, and c. Starting from an NDA $(S, \langle o, t\rangle)$ and considering the determinised automaton $(S, \langle o^\sharp, t^\sharp\rangle)$, it can be used to check language equivalence of two sets of states X and Y. Starting from the pair (X, Y), the algorithm builds a relation R that, in case of success, is a bisimulation up-to congruence. In order to do that, it employs the set $todo$ which, intuitively, at any step of the execution, contains the pairs (X', Y') that must be checked: if (X', Y') already belongs to $c(R \cup todo)$, then it does not need to be checked. Otherwise, the algorithm checks if X' and Y' have the same outputs. If $o^\sharp(X') \neq o^\sharp(Y')$ then X and Y are different, otherwise the algorithm inserts (X', Y') in R and, for all $a \in A$, the pairs $(t^\sharp(X')(a), t^\sharp(Y')(a))$ in $todo$. The check $(X', Y') \in c(R \cup todo)$ at step 2.2 is done with the rewriting algorithm of [BP13, Section 3.4].

5.1.5 Proposition. *For all* $X, Y \in \mathscr{P}_\omega(S)$, $[\![X]\!] = [\![Y]\!]$ *iff* HKC(X, Y).

The iterations corresponding to the execution of HKC$(\{x\}, \{u\})$ on the NDA in (5.2) are concisely described by the numbered dashed lines in (5.3). Observe that only a small portion of the determinised automaton is explored; this fact usually makes HKC more efficient than the algorithms based on minimisation, that need to build the whole reachable part of the determinised automaton.

5.1.2 Language equivalence via Brzozowski's algorithm

The problem of checking language equivalence of two sets of states X and Y of a non-deterministic finite automaton can be reduced to that of building the minimal DA for $[\![X]\!]$ and $[\![Y]\!]$ and checking whether they are the same (up to isomorphism). The most well-known procedure consists in first determinising the NDA and then minimising it with Hopcroft's algorithm [Hop71]. Another interesting solution is Brzozowski's algorithm [Brz62].

To explain the latter, it is convenient to consider a set of *initial states* I. Given an NDA $(S, \langle o, t\rangle)$ and a set of states I, Brzozowski's algorithm computes the minimal automaton for the language $[\![I]\!]$ by performing the 4 steps in Figure 5.1.

The operation "reverse and determinise" takes as input an NDA $(S, \langle o, t\rangle)$ and returns a DA $(\mathscr{P}_\omega(S), \langle \overline{o}_R, \overline{t}_R\rangle)$ where the functions $\overline{o}_R \colon \mathscr{P}_\omega(S) \to 2$ and $\overline{t}_R \colon \mathscr{P}_\omega(S) \to \mathscr{P}_\omega(S)^A$ are defined for all $X \in \mathscr{P}_\omega(S)$ as

$$\overline{o}_R(X) = 1 \text{ iff } X \cap I \neq 0 \qquad \overline{t}_R(X)(a) = \{x \in S \mid t(x)(a) \cap X \neq 0\}$$

```
HKC($X,Y$):
 (1) $R$ is empty; $todo$ is $\{(X,Y)\}$;
 (2) while $todo$ is not empty, do
   (2.1) extract $(X',Y')$ from $todo$;
   (2.2) if $(X',Y') \in c(R \cup todo)$ then continue;
   (2.3) if $o^\sharp(X') \neq o^\sharp(Y')$ then return false;
   (2.4) for all $a \in A$, insert $(t^\sharp(X')(a), t^\sharp(Y')(a))$ in $todo$;
   (2.5) insert $(X',Y')$ in $R$;
 (3) return true;
```

```
Brzozowski:
 (1) reverse and determinise;
 (2) take the reachable part;
 (3) reverse and determinise;
 (4) take the reachable part.
```

Figure 5.1: Generic HKC algorithm, parametric on $o^\sharp$, $t^\sharp$ and c. Generic Brzozowski's algorithm, parametric on reverse and determinise. Instantiation to language/failure/must equivalence.

and the new initial state is the old set of final states: $\overline{I}_R = \{x \mid o(x) = 1\}$. The second step consists in taking the part of $(\mathscr{P}_\omega(S), \langle \overline{o}_R, \overline{t}_R \rangle)$ which is reachable from $\overline{I}_R$. The third and the fourth steps perform this procedure once more.
As an example, consider the NDA in (5.2) with the set of initial states $I = \{x\}$. Brzozowski's algorithm builds the minimal DA accepting $[\![\{x\}]\!]$ as follows. After the first two steps, it returns the following DA where the initial state is $\{y\}$.

$$\begin{tikzcd}
\{y\} \arrow[r, "a"] & \overline{\{x,z\}} \arrow[r, "a"] & \{z,y\} \arrow[r, "a"] & \overline{\{x,y,z\}} \arrow[loop right, "a"]
\end{tikzcd}$$

After steps 3 and 4, it returns the DA below with initial state $\{\{x,z\}\{x,y,z\}\}$.

$$\begin{tikzcd}
\{\{x,z\}\{x,y,z\}\} \arrow[r, "a"] & \overline{\{\{y\}\{z,y\}\{x,y,z\}\}} \arrow[dl, "a"'] \\
\{\{x,z\}\{z,y\}\{x,y,z\}\} \arrow[r, "a"] & \overline{\{\{y\}\{x,z\}\{z,y\}\{x,y,z\}\}} \arrow[loop right, "a"]
\end{tikzcd}$$

Computing the minimal NDA in (5.2) with the set of initial states $I = \{u\}$ results in an isomorphic automaton, showing the equivalence of x and u.

5.2 Algorithms for failure and must testing semantics

In this section we show how the algorithms HKC and Brzozowski can be adapted for reasoning on failure and must testing semantics. Next we briefly summarise the coalgebraic modelling of these semantics via the generalised powerset construction, as introduced in Chapter 4.

An LTS over the alphabet A is a pair (S,t) with $t\colon S \to \mathcal{P}_\omega(S)^A$. For a function $\varphi \in \mathcal{P}_\omega(S)^A$, $I(\varphi)$ denotes the set of all labels "enabled" by φ, given by $I(\varphi) = \{a \in A \mid \varphi(a) \neq 0\}$, while $\mathit{Fail}(\varphi)$ denotes the set $\{Z \subseteq A \mid Z \cap I(\varphi) = 0\}$. A *failure pair* of a state $x \in S$ is a pair $(w,Z) \in A^* \times \mathcal{P}_\omega(A)$ such that $x \xrightarrow{w} y$ and $Z \in \mathit{Fail}(t(y))$. The set of failure pairs of x is denoted by $\mathcal{F}(x)$. Given two states $x,y \in S$, x is *failure equivalent* to y ($x \sim_{\mathcal{F}} y$) if and only if $\mathcal{F}(x) = \mathcal{F}(y)$.
In short, the coalgebraic modelling of failure semantics in Section 4.1.1 is as follows. First, the states of an LTS (S,t) are decorated by means of the output function $o\colon S \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))$ defined as

$$o(x) = \mathit{Fail}(t(x)). \tag{5.5}$$

Then, the decorated LTS $(S, \langle o,t\rangle)$ is translated, using the generalised powerset construction from Section 2.3, into a Moore machine $(\mathcal{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ with $o^\sharp\colon \mathcal{P}_\omega(S) \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))$ and $t^\sharp\colon \mathcal{P}_\omega(S) \to \mathcal{P}_\omega(S)^A$ defined for all $X \in \mathcal{P}_\omega(S)$ as

$$o^\sharp(X) = \bigsqcup_{x\in X} o(x) \qquad\qquad t^\sharp(X)(a) = \bigsqcup_{x\in X} t(x)(a) \tag{5.6}$$

where, in the left equation, $\sqcup$ denotes the union of subsets in $\mathcal{P}_\omega(\mathcal{P}_\omega(A))$. Note that $(\mathcal{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ is a Moore machine with outputs in $\mathcal{P}_\omega(\mathcal{P}_\omega(A))$. The map into the final Moore coalgebra $[\![-]\!]\colon \mathcal{P}_\omega(S) \to (\mathcal{P}_\omega(\mathcal{P}_\omega(A)))^{A^*}$ associates to a set of states their "behaviours". The latter are in one-to-one correspondence with failure pairs. More explicitly, for all $x \in S$, $Z \in \mathcal{P}_\omega(\mathcal{P}_\omega(A))$ and $w \in A^*$:

$$Z \in [\![\{x\}]\!](w) \text{ iff } (w,Z) \in \mathcal{F}(x). \tag{5.7}$$

Hence,

$$x \sim_{\mathcal{F}} y \text{ iff } [\![\{x\}]\!] = [\![\{y\}]\!]. \tag{5.8}$$

The trace-based characterisation of must testing in [CH89] leads to a similar coalgebraic representation via the generalised powerset construction. In Section 4.6.2 we modelled LTS's with internal behaviour and divergence as coalgebras $(S, t\colon S \to (1 + \mathcal{P}_\omega(S))^A)$ such that, for all $x \in S$ and $a \in A$

$$t(x)(a) = \top, \text{ if } x \not\downarrow a \qquad t(x)(a) = \{y \mid x \overset{a}{\Rightarrow} y\}, \text{ otherwise.} \tag{5.9}$$

Recall that $\overset{a}{\Rightarrow}$ denotes the execution of an action $a \in A$, possibly preceded or followed by (any number of) internal steps τ, $\downarrow$ is the convergence predicate, and $1 = \{\top\}$ is used to coalgebraically "capture" divergent behaviours.
Then, we decorate such LTS's by means of a function $o\colon S \to 1 + \mathcal{P}_\omega(\mathcal{P}_\omega A)$ such that, for all $x \in S$ and $a \in A$

$$o(x) = \begin{cases} \top & \text{if } x \not\downarrow \\ \bigcup_{x \xrightarrow{\tau} x'} o(x') & \text{if } x \xrightarrow{\tau} \\ \mathit{Fail}(t(x)) & \text{otherwise.} \end{cases} \tag{5.10}$$

Finally, we apply the generalised powerset construction and derive a Moore machine $(1 + \mathcal{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ defined for all $x \in 1 + \mathcal{P}_\omega(S)$ and $a \in A$ as

$$o^\sharp(X) = \begin{cases} \top & \text{if } X = \top \\ \bigsqcup_{x\in X} o(x) & \text{if } X \in \mathcal{P}_\omega(S) \end{cases} \qquad t^\sharp(X)(a) = \begin{cases} \top & \text{if } X = \top \\ \bigsqcup_{x\in X} t(x)(a) & \text{if } X \in \mathcal{P}_\omega(S) \end{cases} \tag{5.11}$$

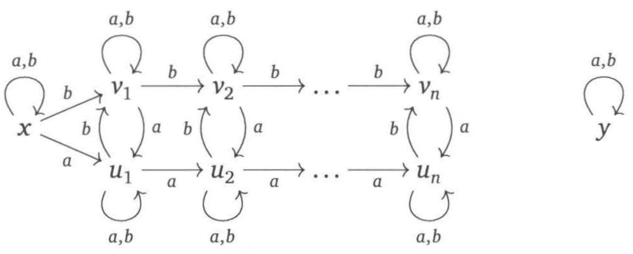
The state space $(1+\mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}$ of the final Moore coalgebra carries the structure of a join semilattice with top, inducing a partial order $\sqsubseteq_{\mathcal{M}}$. This, together with the behaviour map $[\![-]\!] : 1+\mathcal{P}_\omega(S) \to (1+\mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}$ further enabled formalising must testing preorder and equivalence, respectively, as follows:

$$\begin{array}{l} x \sqsubseteq_{mst} y \text{ iff } [\![\{y\}]\!] \sqsubseteq_{\mathcal{M}} [\![\{x\}]\!] \\ x \sim_{mst} y \text{ iff } [\![\{x\}]\!] = [\![\{y\}]\!]. \end{array} \tag{5.12}$$

5.2.1 HKC for failure semantics

The algorithm HKC in Figure 5.1 can be used to check failure equivalence on an LTS (S,t) by taking $o^\sharp$ and $t^\sharp$ as defined in (5.6). Then, the congruence closure c is defined as for language equivalence in (5.4). The analogue of Proposition 5.1.5 can be proved in exactly the same way (check Section 5.2.3): in particular, soundness of bisimulation up-to-congruence is guaranteed from the fact that $(\mathcal{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ is a bialgebra.

We provide an example of using bisimulation up-to congruence for reasoning on failure semantics. Consider the following systems, where n is an arbitrary natural number:



It is easy to see that x and y are bisimilar: intuitively, all the states of the automata depicted above can trigger actions a and b as a first step and, moreover, all their subsequent transitions lead to states with the same behaviour. Therefore x and y are also $\mathcal{F}$-equivalent, according to van Glabbeek's lattice of semantic equivalences [vG01a] (partially) illustrated in Figure 4.1 in Chapter 4.

The coalgebraic machinery provides a "yes" answer with respect to $\mathcal{F}$-equivalence of the two LTS's as well. After determinisation, $\{x\}$ can reach all states of shape: $\{x\}\cup\overline{u}_i$, $\{x\}\cup\overline{v}_i$, $\{x\}\cup\overline{u}_i\cup\overline{v}_i$, for $i\in\{1,\dots,n\}$ and $\{x\}\cup\overline{u}_j\cup\{v_1\}$, $\{x\}\cup\overline{v}_j\cup\{u_1\}$, respectively, for $j\in\{2,\dots,n\}$. (We write, for example, $\overline{u}_i$ in order to represent the set $\{u_1,u_2,\dots,u_i\}$.)

Consequently, the generalised powerset construction associates to x a Moore automaton consisting of $5n-1$ states, whereas the determinisation of y has only one state. Hence, the (Moore) bisimulation relation R including $(\{x\},\{y\})$ consists of $5n-1$ pairs as follows:

$$\begin{array}{rcl} R & = & \{(\{x\},\{y\})\}\cup \\ & & \{(\{x\}\cup\overline{u}_i\cup\{v_1\},\{y\}), (\{x\}\cup\overline{v}_i\cup\{u_1\},\{y\}) \mid i\in\{2,\dots,n\}\}\cup \\ & & \{(\{x\}\cup\overline{u}_i,\{y\}), (\{x\}\cup\overline{v}_i,\{y\}), (\{x\}\cup\overline{u}_i\cup\overline{v}_i,\{y\}) \mid i\in\{1,\dots,n\}\}. \end{array} \tag{5.13}$$

For a better intuition, we illustrate bellow the determinisations starting from x and y, for the case $n = 3$:

$$\begin{tikzcd}
 & & & \{x\} \arrow[dll, "a"'] \arrow[drr, "b"] & & & & \{y\} \arrow[loop right, "{a,b}"] \\
 & \{x,u_1\} \arrow[dl, "a"'] \arrow[drr, "b"] & & & & \{x,v_1\} \arrow[dll, "a"'] \arrow[dr, "b"] & & \\
\{x,u_1,u_2\} \arrow[loop below, "a"] \arrow[ddrrr, bend right, "b"'] & & & \{x,u_1,v_1\} \arrow[dl, "a"'] \arrow[dr, "b"] & & & \{x,v_1,v_2\} \arrow[loop below, "b"] \arrow[ddlll, bend left, "a"] & \\
 & & \{x,u_1,u_2,v_1\} \arrow[loop below, "a"] \arrow[dr, "b"] & & \{x,u_1,v_1,v_2\} \arrow[loop below, "b"] \arrow[dl, "a"'] & & & \\
 & & & \{x,u_1,u_2,v_1,v_2\} \arrow[loop below, "{a,b}"] & & & &
\end{tikzcd}$$

It is easy to see that the bisimulation relating $\{x\}$ and $\{y\}$ consists of all pairs $(X, \{y\})$, with X ranging over the state space of the Moore automaton derived according to the generalised powerset construction, starting with $\{x\}$.
Observe that all the pairs in R in (5.13) can be "generated" from $(\{x\}, \{y\})$, $(\{x\} \cup \overline{u}_i, \{y\})$ and $(\{x\} \cup \overline{v}_i, \{y\})$ by iteratively applying the rules in (5.4). Therefore, for an arbitrary natural number n, the bisimulation up-to congruence stating the equivalence of x and y is:

$$R^c = \{(\{x\}, \{y\})\} \cup \{(\{x\} \cup \overline{u}_i, \{y\}), (\{x\} \cup \overline{v}_i, \{y\}) \mid i \in \{1, \ldots, n\}\}$$

and consists of only $2n + 1$ pairs. The latter represent exactly the states explored by HKC.

5.2.2 HKC for must semantics

The coalgebraic characterisation of must testing guarantees soundness and completeness of bisimulation up-to congruence for the associated equivalence. Bisimulations are now relations $R \subseteq (1 + \mathscr{P}_\omega(S)) \times (1 + \mathscr{P}_\omega(S))$ on the state space $1 + \mathscr{P}_\omega(S)$ where $o^\sharp$ and $t^\sharp$ are defined as in (5.11). Now, the congruence closure $c(R)$ of a relation $R \subseteq (1+\mathscr{P}_\omega(S)) \times (1+\mathscr{P}_\omega(S))$ is defined by the rules in (5.4) where $\sqcup$ is the join in $(1 + \mathscr{P}_\omega(S))$ (rather than the union in $\mathscr{P}_\omega(S)$). By simply redefining $o^\sharp$, $t^\sharp$ and $c(R)$, the algorithm in Figure 5.1 can be used to check must equivalence and preorder (the detailed proof is in Section 5.2.3).
Consider, for an example, the LTS's in Section 4.6.2:

$$\begin{tikzcd}
x \arrow[r, "a"] \arrow[rr, bend left, "b"] \arrow[d, bend left, "a"] & x_2 \arrow[r, "b"] \arrow[d, "\tau"] & x_4 \arrow[loop right, "\tau"] & y \arrow[loop left, "a"] \arrow[r, "b"] & y_1 \arrow[loop right, "\tau"] \\
x_1 \arrow[u, bend left, "a"] \arrow[rr, bend right, "b"'] & x_3 \arrow[l, "a"'] \arrow[r, "b"] & x_5 \arrow[u, "\tau"'] & &
\end{tikzcd} \tag{5.14}$$

In Section 4.6.2 we showed that the states x and y are must equivalent, by identifying a bisimulation relating $\{x\}$ and $\{y\}$. This time however, we depict by the dashed lines in (5.15) the relation $R = \{(\{x\}, \{y\}), (\{x_1, x_2, x_3\}, \{y\})\}$ which is not a bisimulation, but a bisimulation up-to congruence, since both $(\top, \top) \in c(R)$ and $(\{x, x_1\}, \{y\}) \in c(R)$. For the latter, observe that

$$\{x, x_1\} \; c(R) \; \{y, x_1\} \; c(R) \; \{x_1, x_2, x_3\} \; c(R) \; \{y\}.$$

$$\begin{tikzcd}
\top \arrow[loop above, "{a,b}"] & \{x\} \arrow[l, "b"'] \arrow[r, "a"] \arrow[d, dash] & \{x_1,x_2,x_3\} \arrow[r, "a"] \arrow[ll, bend right, "b"'] \arrow[dl, dashed, dash] & \{x,x_1\} \arrow[r, "a"] \arrow[lll, bend right, "b"'] & \{x,x_1,x_2,x_3\} \arrow[loop above, "a"] \arrow[llll, bend right, "b"'] \\
\top \arrow[loop below, "{a,b}"] & \{y\} \arrow[l, "b"] \arrow[loop below, "a"] & & &
\end{tikzcd} \tag{5.15}$$

It is important to remark here that HKC computes this relation without the need of exploring all the reachable part of the Moore machine $(1+\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$. So, amongst all the states in (5.15), HKC only explores $\{x\}$, $\{y\}$ and $\{x_1, x_2, x_3\}$.

5.2.3 Correctness of HKC

We provide a uniform proof of correctness of HKC in Figure 5.1 for language, failure and must semantics (Proposition 5.1.5). The key step is (the analogue) of Theorem 5.1.4 stating that bisimulation up-to congruence is a sound proof technique. This holds for any bialgebra (see e.g. Corollary 6.6 in [RBR13]) and, in particular, for $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ (or $(1+\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$) which is guaranteed to be a bialgebra by the generalised powerset construction (we refer the interested reader to [Kli11] for a nice introduction on this topic).
We first observe that if HKC(X,Y) returns `true` then the relation R that is built before arriving to step 3 is a bisimulation up-to congruence. Indeed, the following proposition is an invariant for the loop corresponding to step 2:

$$R \rightarrowtail c(R \cup todo)$$

This invariant is preserved since at any iteration of the algorithm, a pair (X', Y') is removed from $todo$ and inserted in R after checking that $o^\sharp(X') = o^\sharp(Y')$ and adding $(t^\sharp(X')(a), t^\sharp(Y')(a))$ for all $a \in A$ in $todo$. Since $todo$ is empty at the end of the loop, we eventually have $R \rightarrowtail c(R)$, i.e., R is a bisimulation up-to congruence.
We now prove that if HKC(X,Y) returns `false`, then $[\![X]\!] \neq [\![Y]\!]$. Note that for all (X', Y') inserted in $todo$, there exists a word $w \in A^\star$ such that, in the determinised NDA, $X \xrightarrow{w} X'$ and $Y \xrightarrow{w} Y'$. Since $o^\sharp(X') \neq o^\sharp(Y')$, then $[\![X]\!](w) = o^\sharp(X') \neq o^\sharp(Y') = [\![Y]\!](w)$.

5.2.4 Brzozowski's algorithm for failure semantics

A variation of Brzozowski's algorithm for Moore machines is given in [BBRS12]. We could apply such algorithm to the Moore machine $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ which is induced by a decorated LTS $(S, \langle o, t\rangle)$, with o defined as in (5.5). However, we propose a more efficient variation that skips the first determinisation from $(S, \langle o, t\rangle)$ to $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$.
The novel algorithm consists of the four steps described in Section 5.1.2, where the procedure "reverse and determinise" is modified as follows: $(S, \langle o, t\rangle)$ with initial state I is transformed into $(\mathscr{P}_\omega(\mathscr{P}_\omega(A))^S, \overline{o}_R, \overline{t}_R)$ where

$$\overline{o}_R \colon \mathscr{P}_\omega(\mathscr{P}_\omega(A))^S \to \mathscr{P}_\omega(\mathscr{P}_\omega(A))$$

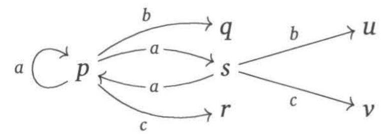
and

$$\overline{t}_R \colon \mathscr{P}_\omega(\mathscr{P}_\omega(A))^S \to (\mathscr{P}_\omega(\mathscr{P}_\omega(A))^S)^A$$

are defined for all functions $\psi \in \mathscr{P}_\omega(\mathscr{P}_\omega(A))^S$ as

$$\overline{o}_R(\psi) = \bigsqcup_{x \in I} \psi(x) \qquad\qquad \overline{t}_R(\psi)(a)(x) = \bigsqcup_{y \in t(x)(a)} \psi(y) \tag{5.16}$$

and the new initial state is $\overline{I}_R = o$.
Note that the result of this procedure is a Moore machine. Brzozowski's algorithm in Figure 5.1 transforms an NDA $(S, \langle o, t\rangle)$ with initial state I into the minimal DA for $[\![I]\!]$. Analogously, our novel algorithm transforms an LTS into the minimal Moore machine for $[\![I]\!]$.
Let us illustrate the minimisation procedure by means of an example. Consider the LTS (S,t) on the alphabet $A = \{a,b,c\}$ depicted below.



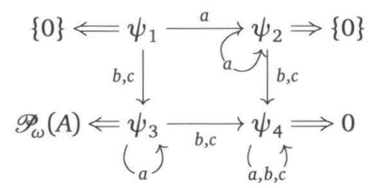
$$\begin{tikzcd}
 & q & & u \\
p \arrow[loop left, "a"] \arrow[ur, "b"] \arrow[r, bend left, "a"] \arrow[dr, "c"'] & s \arrow[l, bend left, "a"] \arrow[urr, "b"] \arrow[drr, "c"'] & & \\
 & r & & v
\end{tikzcd}$$

$$\begin{array}{ll}
o(p) = \{0\} & o(s) = \{0\} \\
o(q) = \mathscr{P}_\omega(A) & o(u) = \mathscr{P}_\omega(A) \\
o(r) = \mathscr{P}_\omega(A) & o(v) = \mathscr{P}_\omega(A)
\end{array}$$

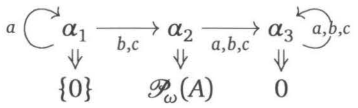
The function $o\colon S \to \mathscr{P}_\omega(\mathscr{P}_\omega(A))$ assigning to each state x the set $\mathit{Fail}(t(x))$ is given on the right. Suppose we want to build the minimal Moore machine for the behaviour $[\![\{p\}]\!]$, i.e., the set of failure pairs of p:

$$\mathscr{F}(p) = \{(a^*, \{0\}), (a^*b, \mathscr{P}_\omega(A)), (a^*c, \mathscr{P}_\omega(A))\}.$$

By applying our algorithm to the decorated LTS $(S, \langle o, t\rangle)$, we first obtain the intermediate Moore machine on the left below, where a double arrow $\psi \Rightarrow Z$ means that the output of ψ is the set Z. The new initial state is $\psi_1 \colon S \to \mathscr{P}_\omega(\mathscr{P}_\omega A)$ which, by definition, is the output function of the original LTS mapping p, s to $\{0\}$ and q, r, u and v to $\mathscr{P}_\omega(A)$. The explicit definitions of the other functions ψ_i can be easily computed according to the definition of $\overline{t}_R$ (5.16).



$$\begin{tikzcd}
\{0\} & \psi_1 \arrow[l, Rightarrow] \arrow[r, "a"] \arrow[d, "b{,}c"'] & \psi_2 \arrow[r, Rightarrow] \arrow[loop above, "a"] \arrow[d, "b{,}c"] & \{0\} \\
\mathscr{P}_\omega(A) & \psi_3 \arrow[l, Rightarrow] \arrow[r, "b{,}c"'] \arrow[loop below, "a"] & \psi_4 \arrow[r, Rightarrow] \arrow[loop below, "a{,}b{,}c"] & 0
\end{tikzcd}$$



$$\begin{tikzcd}
\alpha_1 \arrow[loop left, "a"] \arrow[r, "b{,}c"'] \arrow[d, Rightarrow] & \alpha_2 \arrow[r, "a{,}b{,}c"'] \arrow[d, Rightarrow] & \alpha_3 \arrow[loop right, "a{,}b{,}c"] \arrow[d, Rightarrow] \\
\{0\} & \mathscr{P}_\omega(A) & 0
\end{tikzcd}$$

Observe that $[\![\psi_1]\!]$ is the "reverse" of $[\![\{p\}]\!]$. For instance, triggering a sequence in the language denoted by ba^* from ψ_1 leads to ψ_3 with output $\mathscr{P}_\omega(A)$; this is the same output we get by executing a^*b from p, according to $\mathscr{F}(p)$. Executing "reverse and determinise" once more (step 3) and taking the reachable part (step 4), we obtain the minimal Moore machine depicted on the right, with initial state α_1.
The correctness of this algorithm is established in Section 5.2.5; it builds on the coalgebraic perspective on Brzozowski's algorithm given in [BBRS12].

5.2.5 Correctness of Brzozowski for failure semantics

The main intuition behind Brzozowski algorithm is that the procedure `reverse and determinise` transforms a system into one having the "reversed" behaviour. Moreover, if the original system is *reachable* (that is all the states are reachable from the initial state), then the resulting system is *observable* (that is, all the states have different behaviours). Therefore, after performing the first two steps of Brzozowski's algorithm, one obtains a system which is reachable and that has a "reversed" behaviour. After the third step, the system has the original behaviour and, moreover, it is observable. After the fourth step, it is observable and reachable, that is, it is *minimal*.

There are two key steps for our proof: (a) showing that the procedure `reverse and determinise` introduced in Section 5.2.4 transforms a decorated LTS into a Moore machine having "reversed" behaviour; (b) showing that at the third step the algorithm transforms a reachable Moore machine into an observable one.

Point (b) follows immediately from Section 5 in [BBRS12], where a variation of Brzozowski's algorithm for Moore machines is introduced: when restricted to Moore machines, the operations of reversing and determinising in the our algorithm coincide with those in [BBRS12].

In the sequel, we prove (a) by relying on [BBRS12]. Let (S,t) be an LTS with the initial set of initial states $i\colon 1 \to \mathscr{P}_\omega(S)$ (we prefer to use this functional notation, rather than $I \in \mathscr{P}_\omega(S)$, because it is more convenient for the proof). Let $(\mathscr{P}_\omega(S), \langle o^\sharp, t^\sharp\rangle)$ be the corresponding Moore machine (as defined in (5.6)) and let $[\![-]\!] : \mathscr{P}_\omega(S) \to (\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{A^*}$ be the induced semantics map.

By reversing and determinising as in [BBRS12], we obtain the Moore machine

$$(\mathscr{P}_\omega(\mathscr{P}_\omega(A))^{\mathscr{P}_\omega S}, \langle o^\sharp_R t^\sharp_R\rangle)$$

with initial states i_R, defined as

$$i_R = o^\sharp \qquad o^\sharp_R(\varphi) = \varphi \circ i \qquad t^\sharp_R(\varphi)(a)(X) = \varphi(t^\sharp(X)(a)). \tag{5.17}$$

According to [BBRS12], we know that this machine has "reversed" behaviour, *i.e*,

$$(\forall w \in A^*) \,.\, [\![i_R]\!]_1(w) = [\![i]\!](w^R) \tag{5.18}$$

where

$$[\![-]\!]_1 \colon \mathscr{P}_\omega(\mathscr{P}_\omega(A))^{\mathscr{P}_\omega S} \to (\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{A^*}$$

is the semantic map, and w^R denotes the reverse word w inductively defined as $\varepsilon^R = \varepsilon$ and $(aw')^R = w'^R a$.

Our algorithm performs the determinisation and the "reverse and determinise" at once. For a Moore machine defined as in (5.16,) the map to the final coalgebra

$$[\![-]\!]_2 \colon \mathscr{P}_\omega(\mathscr{P}_\omega(A))^S \to (\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{A^*}$$

satisfies the following lemma.

5.2.1 Lemma. *Let $\psi \in (\mathscr{P}_\omega(\mathscr{P}_\omega A))^S$ and $\varphi \in (\mathscr{P}_\omega(\mathscr{P}_\omega A))^{\mathscr{P}_\omega S}$ be such that, for all $X \in \mathscr{P}_\omega S$*

$$\varphi(X) = \sqcup_{x \in X} \psi(x). \qquad (\star)$$

Then, $[\![\psi]\!]_2 = [\![\varphi]\!]_1$.

PROOF. The proof is by induction on $w \in A^*$. For the base case, $w = \varepsilon$, we have:

$$[\![\psi]\!]_2(\varepsilon) = \overline{o}_R(\psi) = \bigsqcup_{x \in i} \psi(x) \overset{(\star)}{=} \varphi(i) = o_R^\sharp(\varphi) = [\![\varphi]\!]_1(\varepsilon)$$

For the inductive step, consider $w \in A^*$ and assume that $[\![\psi]\!]_2(w) = [\![\varphi]\!]_1(w)$ holds for all ψ, φ satisfying $(\star)$.
We want to prove that $[\![\psi]\!]_2(aw) = [\![\varphi]\!]_1(aw)$ holds for $a \in A$. We first define $\overline{\varphi}_a(X) = \varphi(t^\sharp(X)(a))$ and $\overline{\psi}_a(x) = \sqcup_{y \in t(x)(a)} \psi(y)$, where $X \in \mathscr{P}_\omega S$ and $x \in S$ (which, as an intuition, will further be used when applying the induction hypothesis in our proof).
Note that $(\star)$ is satisfied by $\overline{\varphi}_a$ and $\overline{\psi}_a$: $\overline{\varphi}_a(X) = \bigsqcup\limits_{x \in X} \overline{\psi}_a(x)$, because

$$\overline{\varphi}_a(X) = \varphi(t^\sharp(X)(a)) = \varphi(\bigsqcup_{x \in X} t(x)(a))t \overset{(\star)}{=} \bigsqcup_{x \in X} \bigsqcup_{y \in t(x)(a)} \psi(y) = \bigsqcup_{x \in X} \overline{\psi}_a(x).$$

At this point it is easy to see that $[\![\psi]\!]_2(aw) = [\![\varphi]\!]_1(aw)$:

$$\begin{aligned}
[\![\varphi]\!]_1(aw) &= [\![\lambda X\,.\,\varphi(t^\sharp(X)(a))]\!]_1(w) && \text{(by definition of } t_R^\sharp\text{)}\\
&= [\![\lambda X\,.\,\overline{\varphi}_a(X)]\!]_1(w)\\
&= [\![\lambda x\,.\,\overline{\psi}_a(x)]\!]_2(w) && \text{(by the induction hypothesis)}\\
&= [\![\lambda x\,.\,\sqcup_{y \in t(x)(a)} \psi(y)]\!]_2(w)\\
&= [\![\overline{t}_R(\psi)(a)]\!]_2(w) = [\![\psi]\!]_2(aw).
\end{aligned}$$

□

In particular, if we take $\psi = \overline{I}_R$ and $\varphi = o^\sharp$, we have that $[\![\overline{I}_R]\!]_2 = [\![o^\sharp]\!]_1$. By (5.18) and the fact that and $i_R = o^\sharp$ the following holds:

$$(\forall w \in A^*)\,.\,[\![o^\sharp]\!]_1(w) = [\![i]\!](w^R).$$

summarising, for all $w \in A^*$, $[\![\overline{I}_R]\!]_2(w) = [\![i]\!](w^R)$.
For an example of this fact, observe that p and ψ_1 in Section 5.2.4 have reversed behaviours.

5.2.6 Brzozowski's algorithm for must semantics

The Brzozowski algorithm introduced in Section 5.2.4 for failure equivalence can be used also for checking $\sim_{mst}$ and $\sqsubseteq_{mst}$. Now, the procedure "reverse and determinise" returns the Moore machine $((1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S, \overline{o}_R, \overline{t}_R)$. The initial state $\overline{I}_R$, the outputs

$$\overline{o}_R \colon (1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S \to 1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A))$$

and the transitions

$$\overline{t}_R \colon (1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S \to ((1 + \mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S)^A$$

are defined as in (5.16), plus the case

$$\overline{t}_R(\psi)(a)(x) = \top \text{ if } t(x)(a) = \top,$$

by replacing o and t with those defined in (5.10) and (5.9), and by considering the join operation $\sqcup$ in $1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ (rather than in $\mathscr{P}_\omega(\mathscr{P}_\omega(A))$).
In what follows, we illustrate Brzozowski's algorithm for must testing, by means of an example. Consider the divergent LTS (S,t) below:

$$\begin{tikzcd}
x_1 \arrow[dr, "a"] \arrow[dd, bend left, "a"] & & & \\
& x_3 \arrow[r, "b"] & x_4 \arrow[r, "{a,b}"] & x_5 \arrow[loop right, "\tau"] \\
x_2 \arrow[ur, "a"'] \arrow[uu, bend left, "a"] & & &
\end{tikzcd}$$

and $o\colon S \to 1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ the decoration function

$$\begin{array}{rcl}
o(x_1) & = & o(x_2) \quad = \quad \{b\} \\
o(x_3) & = & \{a\} \\
o(x_4) & = & \{0\} \\
o(x_5) & = & \top.
\end{array}$$

Assume we want to build the minimal Moore machine for the behaviour of x_1, which is must testing equivalent with x_2. By applying our algorithm to the decorated LTS $(S,\langle o,t\rangle)$ we obtain the following intermediate Moore machine:

$$\begin{tikzcd}
\{0,\{b\}\} & \{0,\{a\},\{b\}\} & 0 \\
\psi_1 \arrow[u, Rightarrow] \arrow[r, "a"] \arrow[d, "b"'] & \psi_2 \arrow[u, Rightarrow] \arrow[r, "b"] \arrow[loop below, "a"] & \psi_4 \arrow[u, Rightarrow] \arrow[loop right, "b"] \arrow[d, "a"'] \arrow[d, bend left, "b"] \\
\psi_3 \arrow[loop left, "a"] \arrow[rru, bend right, "b"'] \arrow[d, Rightarrow] & & \psi_5 \arrow[loop right, "a"] \arrow[d, Rightarrow] \\
0 & & \top
\end{tikzcd}$$

Observe that $[\![\psi_1]\!]$ is the "reverse" of $[\![\{x_1\}]\!]$. For instance, each sequence w in the language denoted by ba^* determines, when triggered from ψ_1, the output 0, which coincides with the (empty) set of actions that the automaton can fail to execute after performing w. Finally, we execute `reverse` and `determinise` and get the following minimal Moore automaton (with initial state α_1):

$$\begin{tikzcd}
\{0,\{b\}\} & \{0,\{a\},\{b\}\} & 0 \\
\alpha_1 \arrow[u, Rightarrow] \arrow[r, "a"] \arrow[d, "b"'] & \alpha_2 \arrow[u, Rightarrow] \arrow[r, "b"] \arrow[loop below, "a"] & \alpha_4 \arrow[u, Rightarrow] \arrow[d, "{a,b}"] \\
\alpha_3 \arrow[loop left, "{a,b}"] \arrow[d, Rightarrow] & & \alpha_5 \arrow[loop right, "{a,b}"] \arrow[d, Rightarrow] \\
0 & & \top
\end{tikzcd}$$

Remark that the behaviours of the must equivalent states x_1 and x_2 have been "collapsed" into α_1.

5.2.7 Correctness of Brzozowski's algorithm for must semantics

In this section we show the correctness of Brzozowski's algorithm for must equivalence. The approach is similar to the one described in Section 5.2.5; the slight differences which are consequences of the divergence-sensitive nature of must semantics are summarised as follows.
Consider an LTS with divergence $(S, t: S \to (1+\mathscr{P}_\omega S)^A)$, with the initial set of initial states $i: 1 \to \mathscr{P}_\omega(S)$. As recalled in the beginning of this chapter, the corresponding coalgebraic ingredients are extended to $1+\mathscr{P}_\omega(-)$ (see 5.11): the associated Moore machine has the state space in $1+\mathscr{P}_\omega S$ and observations in $1+\mathscr{P}_\omega(\mathscr{P}_\omega A)$, whereas the induced semantic map becomes $[\![-]\!]: 1+\mathscr{P}_\omega(S) \to (1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{A^*}$. Consequently, the current approach considers the join operation $\sqcup$ in $1+\mathscr{P}_\omega(-)$, rather than in $\mathscr{P}_\omega(-)$, as for failure semantics.

By reversing and determinising as in [BBRS12], we obtain the Moore machine

$$\mathscr{M}_R = ((1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{1+\mathscr{P}_\omega S}, \langle o_R^\sharp t_R^\sharp\rangle)$$

for which the initial set of states i_R, $o_R^\sharp$ and $t_R^\sharp$ are defined as in (5.17), in Section 5.2.5. Equivalently to the statement in (5.18), this machine has the "reversed" behaviour of the initial LTS.
The novel algorithm performing the determinisation and the "reverse and determinise" at once returns, for the case of must semantics, the Moore machine

$$\overline{\mathscr{M}}_R = ((1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S, \overline{o}_R, \overline{t}_R)$$

for which the corresponding initial state $\overline{I}_R$, the outputs $\overline{o}_R: (1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S \to 1+\mathscr{P}_\omega(\mathscr{P}_\omega(A))$ and the transitions $\overline{t}_R: (1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S \to ((1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S)^A$ are defined as in (5.16) (plus the case $\overline{t}_R(\psi)(a)(x) = \top$ if $t(x)(a) = \top$), by replacing o and t with those defined in (5.10) and (5.9), in the beginning of this chapter.
The fact that $\overline{\mathscr{M}}_R$ has the reverse behaviour of the original LTS follows according to a statement similar to the one in Lemma 5.2.1, by taking $[\![-]\!]_2: (1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^S \to (1+\mathscr{P}_\omega(\mathscr{P}_\omega(A)))^{A^*}$, $\psi = \overline{I}_R$ and $\varphi = o^\sharp$ (satisfying $(\star)$), and the fact that $\mathscr{M}_R$ has reversed behaviour:

$$(\forall w \in A^*). [\![i_R]\!]_1(w) = [\![i]\!](w^R).$$

To conclude, the soundness of our algorithm follows by:

$$(\forall w \in A^*). [\![\overline{I}_R]\!]_2(w) = [\![i]\!](w^R).$$

5.3 Three families of examples

As discussed in the beginning of this chapter, the theoretical complexity is not informative about the behaviour of these algorithms on concrete cases. In this section, we compare HKC, Brzozowski and partition refinement [CH89] on three families of examples. First, we need some tools to measure their behaviours. For HKC, we take $|R|$, the size of the produced relation R: indeed cycle 2 of HKC is repeated at most $1+|A|\cdot|R|$ times (where $|A|$ is the size of the alphabet). For [CH89], we consider the size n of the reachable part of determinised system: the main loop of the partition refinement is iterated at most n times. Finally, the cost of Brzozowski algorithm is related to the size of both the intermediate Moore machine (built after steps 1,2) and the minimal one (built after steps 3,4).

First consider the following LTS, where n is an arbitrary natural number. After the determinisation, $\{x\}$ can reach all the states of the shape $\{x\} \cup X_N$, where $X_N = \{x_i \mid i \in N\}$ for any $N \subseteq \{1, \dots, n\}$. More precisely, a trace $w \in \{a,b\}^*$ of length k which leads $\{x\}$ to $\{x\} \cup X_N$ can be generally defined as a word whose $k-i+1$'st letter is b if and only if $i \in N$. For instance for $n = 2$, $\{x\} \overset{aa}{\to} \{x\}$, $\{x\} \overset{ab}{\to} \{x, x_1\}$, $\{x\} \overset{ba}{\to} \{x, x_2\}$ and $\{x\} \overset{bb}{\to} \{x, x_1, x_2\}$. All those states are distinguished by must testing; for instance, $[\![\{x, x1, x2\}]\!](a) = \{a\}$ while $[\![\{x, x2\}]\!](a) = \{0\}$. Therefore, the minimal Moore machine for $[\![\{x\}]\!]$ has at least 2^n states.

$$\begin{tikzcd}
x \arrow[loop left, "{a,b}"] \arrow[r, "b"] & x_1 \arrow[r, "{a,b}"] & \dots \arrow[r, "{a,b}"] & x_n \arrow[r, "b"] & u \arrow[loop right, "\tau"] \\
y \arrow[loop left, "{a,b}"] \arrow[r, "b"] \arrow[dr, "{a,b}"] & y_1 \arrow[r, "{a,b}"] & \dots \arrow[r, "{a,b}"] & y_n \arrow[r, "b"] & v \arrow[loop right, "\tau"] \\
& z \arrow[u, bend right, "b"'] \arrow[ul, bend left, "a"] & & &
\end{tikzcd}$$

One can prove that x and y are must equivalent by showing that relation $R =$

$$\{(\{x\},\{y\}), (\{x\},\{y,z\}), (\top,\top)\} \cup \{(\{x\} \cup X_N, \{y,z\} \cup Y_N) \mid N \subseteq \{1, \dots, n\}\}$$

is a bisimulation (here $Y_N = \{y_i \mid i \in N\}$). Note that R contains $2^n + 3$ pairs.
In order to check $[\![\{x\}]\!] = [\![\{y\}]\!]$, HKC builds the following relation,

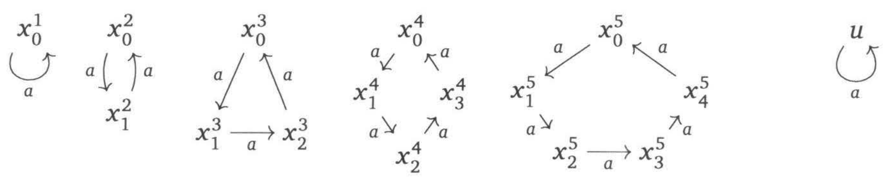
$$R' = \{(\{x\},\{y\}), (\{x\},\{y,z\})\} \cup \{(\{x, x_i\}, \{y, z, y_i\}) \mid i \in \{1, \dots, n\}\}$$

which is a bisimulation up-to and which contains only $n+2$ pairs. It is worth to observe that R' is like a "basis" of R: all the pairs $(X,Y) \in R$ can be generated by those in R' by iteratively applying the rules in (5.4). Therefore, HKC proves $[\![\{x\}]\!] = [\![\{y\}]\!]$ in polynomial time, while minimisation-based algorithms (such as [CH89] or Brzozowski's algorithm) require exponential time.

For the following family of LTS's, the algorithm from [CH89] is efficient (the LTS is already deterministic) while Brzozowski's algorithm is not: the intermediate Moore machine built after steps 1,2 has exponentially many states (for similar reasons as in the previous example, the automaton being reversed first).

$$\begin{tikzcd}
x_n \arrow[r, "{a,b}"] & \dots \arrow[r, "{a,b}"] & x_1 \arrow[r, "b"] & x \arrow[loop right, "{a,b}"]
\end{tikzcd}$$

Finally consider the family of LTS's on $A = \{a\}$, consisting in n disjoint cycles of increasing lengths. The case $n = 5$ is depicted on the left below. Suppose that we want to show that the superposition of states $x_0^1, \dots, x_0^n$ is equivalent to u given on the right.



The states reachable from the set $\{x_0^1, \dots x_0^n\}$ in the determinised system are of the shape $X_k = \{x^i_{k \bmod i} \mid i \leq n\}$. There are p such sets, where $p = lcm[1..n]$ is the least common multiple of the first n natural numbers (this number is greater than 2^n for $n \geq 8$).

With [CH89], one would start by constructing all those sets, and one can show that HKC actually produces a relation of size p. Therefore, those two methods need exponentially many steps. On the other hand, Brzozowski's algorithm is extremely efficient on this family of examples: the output of any state is always $\{0\}$, so that the only reachable state in the intermediate Moore machines (built after steps 1 and 2) is the function mapping all the states to $\{0\}$. Therefore we obtain the minimal realisation immediately.

5.4 Concrete tests on a synchronisation protocol

We implemented the presented algorithms (Brzozowski minimisation and HKC) for ready, failure, and must semantics. Moreover, we tested our implementation and compared the various algorithms, by analyzing some instances of a multiway synchronisation protocol (MSP) due to Parrow and Sjödin [PS96].
The scenario is the following: there are several clients, denoted by $1, 2, \ldots$, trying to synchronise on communication channels, denoted by $a, b \ldots$. Each channel comes with a fixed subset of clients, all of which must agree to participate for the action to take place. For instance, in a configuration denoted by $a(1,2), b(1,2,3)$, with three clients and two channels; clients 1 and 2 have to synchronise to perform action a, and the three clients have to synchronise to perform action b. Parrow and Sjödin study protocols allowing to schedule clients requests, so as to enforce the synchronisation constraints. They propose an ideal and centralized scheduler as a specification, and a distributed and more realistic scheduler. They prove them equivalent, using a notion of equivalence called "cs-equivalence" which entails must-testing equivalence in the considered case. Both schedulers are presented as finite LTS.
We computed those LTS for some small configurations, checked them for must equivalence, and minimised the ideal scheduler with respect to must semantics. For each configuration, we give various size indications in Figure 5.2: the first column is the configuration; the second one gives the number of states of the minimal Moore machine; the third and fourth column give the number of states of the ideal and distributed schedulers, respectively. One can notice that the ideal schedulers are almost minimal, while the distributed ones are huge, comparatively. The fifth column gives the number of reachable states, after determinisation along weak transitions (i.e., the number of states one would start with with a partition-refinement algorithm); this number is usually smaller than the size of the distributed scheduler since the later contains lots of intermediate states that are removed by determinisation. The sixth column gives the size of the intermediate automaton, after performing half of Brzozowski's minimisation algorithm; notice that this intermediate automaton is usually much smaller than the distributed scheduler, but also much larger than the ideal and minimal ones. The last column gives the number of pairs required by HKC to prove the equivalence between the ideal and the distributed scheduler; it is systematically much less than the size of the determinised automaton.

5.5 Discussion

In Chapter 4 we have introduced coalgebraic characterisations of decorated trace and must testing semantics by means of the *generalised powerset construction* [SBBR10]. This allowed us to adapt proof techniques and algorithms that have been developed for lan-

config.	min.	ideal	distr.	determ.	interm.	HKC
a(1,2)	9	9	34	12	88	12
a(1,2,3)	27	27	304	84	1110	82
a(1,2),b(1)	15	18	6089	1074	189	294
a(1,2),b(3)	17	27	1057	303	436	225
a(1,2),b(1,2)	28	34	101532	18608	389	2236
a(1,2),b(1,3)	49	54	38288	11024	2568	5462
a(1,2),b(3,4)	65	81	8666	3230	7570	1806
a(1,2,3),b(1)	45	54	54090	8644	2207	2207
a(1,2,3),b(4)	53	81	12053	3330	5546	2116
a(1,2,3),b(1,4)	-	162	259890	-	-	-
a(1),b(2),c(3)	9	27	5917	1594	126	830
a(1),b(1),c(2)	9	18	37380	7984	66	2351
a(1),b(1),c(1)	9	11	149267	41444	34	2685
a(1,2),b(3),c(4)	33	81	50844	20526	2176	6642

Figure 5.2: Concrete tests.

guage equivalence to must semantics. In particular, in this chapter, we showed that *bisimulations up-to congruence* (that were recently introduced in [BP13] for NDA's) are sound also for must semantics. This fact guarantees the correctness of a generalisation of HKC [BP13] for checking must equivalence and preorder and suggests that the *antichains*-based algorithms [ACH$^+$10, DR10, WDHR06] can be adapted in a similar way. We have also proposed a variation of Brzozowski's algorithm [Brz62] to check must semantics, by exploiting the abstract theory in [BBRS12]. Our contribution is not a simple instantiation of [BBRS12], but developing our algorithm has required some ingenuity to avoid the preliminary determinisation that would be needed to directly apply [BBRS12]. We implemented these algorithms together with an interactive applet available online.
Beyond must semantics, one can use such algorithms to check the *decorated trace equivalences* [vG01a] that have been studied in [BBC$^+$12]: like failure, these are obtained by decorating the states of an LTS with a function $o\colon S \to B$. The key of our approach is that B needs to be a semi-lattice with bottom (for must, a semi-lattice with bottom and top); this is required by the generalised powerset construction so that decorated LTS's can be determinised into Moore machines.

Chapter 6

Future work

We provide an overview of the possible theoretical and practical further developments of the work in this thesis.

With respect to the contributions on generalised regular expressions modelling non-deterministic coalgebras introduced in Chapter 3, we consider:

Extensions to quantitative coalgebras. In the future, we would like to extend the class of systems to include quantitative coalgebras. In [SBBR11], the approach for handling non-deterministic coalgebras was extended to a large class of quantitative systems encompassing weighted automata, simple Segala, stratified and Pnueli-Zuck systems, by considering a functor type that allows the transitions of systems to take values in a monoid structure of quantitative values.
The challenge in this respect arises from the fact that computing bisimulation relations in a quantitative setting will involve matrix manipulations, hence requiring linear algebra techniques of which it is not clear how to implement in CIRC.

Tool enhancements and complexity studies. To improve usability, building a graphical interface for the tool is an obvious next step. The graphical interface should ideally allow the specification of expressions by means of systems of equations (which are then solved internally) or even by means of an automaton, which would then be translated to an expression using Kleene's theorem.
We also would like to explore how adding more axioms than ACI to the prover (that is, each step of the bisimulation checking is performed modulo more equations) improves the performance of the tool. Our experience so far shows that by adding the axioms describing the interplay between $\underline{\emptyset}$ and the other constructs, *i.e.* $\underline{\emptyset} \oplus \varepsilon = \varepsilon$, the prover works significantly faster.
We have not yet studied complexity bounds for the algorithms presented in this paper. We conjecture however that the bounds will be very similar to the already known ones for classical regular expressions [Koz06, Wor08].

In connection with the coalgebraic handling of decorated trace, may and must testing semantics in Chapter 4 and Chapter 5, we consider:

Coalgebraic handling of other semantics. In the future, we want to derive a new representation of possible-futures semantics. This is motivated by the current drawback

of storing for each state of the LTS's the corresponding set of traces. In this context, it might be more appropriate considering the definition of possible-futures semantics given in terms of nested bisimulations [HM85], rather than the set-theoretic one in [vG01a].
Moreover, we aim at providing coalgebraic modellings for the remaining semantics of the spectrum in [vG01a]. Amongst these, we mention possible-worlds semantics, whose path-based characterisation shifts the problem of reasoning on the corresponding equivalence to a setting close to possible-futures semantics. The coalgebraic modelling of possible-futures semantics still requires an efficient handling of the traces associated with a process, as mentioned above. Orthogonally, the challenge in deriving a straightforward modelling of simulation semantics via the generalised powerset construction [SBBR13] originates from the absence of an equivalent trace-based definition.
We would also like to understand how our approach can be combined with the results in [BG06] to obtain a coinductive approach to denotational (linear-time) semantics of different kinds of processes calculi. The work in [BG06] presents a fully abstract model of must testing for CSP by turning the set of processes into a (partial) Moore automaton with output on a certain semiring K and input from a set of actions A. The final semantics of this automaton is then given as a powerserie in K^{A^*}. The approach can be easily extended to trace equivalence and other calculi, such as CCS, but no other decorated trace equivalences are further considered. Our work is similar in spirit of the above as we also construct a Moore automaton from a transition system but, in general, we do not need a semiring structure, making the entire framework much simpler. For example, for the must testing, our Moore automata have outputs in the set $1+\mathcal{P}_\omega(\mathcal{P}_\omega(A))$. The framework is even simpler for the case of trace semantics, where our Moore automata have outputs in the two elements set 2.
Furthermore, we think it is promising to investigate whether our approach can be extended to the testing semantics of probabilistic and non-deterministic processes [DvGHM11, YL92, Seg96].

More algorithms. An interesting topic to investigate in the future is adapting the Brzozowski and HKC algorithms to check *fair testing* [RV07]. In [RV07], fair testing is defined in terms of the so-called failure trees. While the corresponding coalgebraic modelling can be easily derived via the powerset construction, we do not know how to model fair testing equivalence and preorder.
We would also like to study whether Brzozowski and HKC can be adapted and effectively applied to reason on decorated trace semantics of generative probabilistic systems.

Rule formats for compositionality. In the future we consider worth studying to what extent the modal characterisations of decorated trace semantics in [vG01a] can be exploited in order to develop a systematic study of their compositionality for languages defined by SOS-like rules [Plo04] satisfying specific formats.
In this respect, we refer to the work in [Kli09], where both the rule formats and decorated trace equivalences are "massaged" into a bialgebraic setting, by means of logical distributive laws defined in terms of notions of syntax and logical formulae. However, applying the machinery in [Kli09] requires a certain amount of ingenuity for identifying the right logical behaviour. Therefore, one of the challenges (also mentioned as pointer to future work in [Kli09]) consists in (partially) automating the whole procedure or, at least, in gaining more insight on how this could be achieved in a rather algorithmic fashion.

Bibliography

[ABH$^+$12] J. Adámek, F. Bonchi, M. Hülsbusch, B. König, S. Milius, and A. Silva. A coalgebraic perspective on minimization and determinization. In *FoSSaCS*, 58–73, 2012. **Cited** on page 87.

[ABV94] L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS rules into equations. *Inf. Comput.*, 111:1–52, May 1994. **Cited** on page 2.

[ACEII11] L. Aceto, G. Caltais, E. I. Goriac, and A. Ingólfsdóttir. Axiomatizing GSOS with Predicates. In *Proc. SOS 2011*, EPTCS 62, 1–15, 2011. **Cited** on page 2.

[ACGI11] L. Aceto, G. Caltais, E. I. Goriac, and A. Ingólfsdóttir. Preg axiomatizer - a ground bisimilarity checker for GSOS with predicates. In *CALCO*, 378–385, 2011. **Cited** on page 2.

[ACH$^+$10] P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. TACAS*, volume 6015, 158–174, 2010. **Cited** on pages 8, 88 and 104.

[AFV99] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, 197–292, 1999. **Cited** on page 60.

[AH92] L. Aceto and M. Hennessy. Termination, deadlock, and divergence. *JACM*, 39:147–187, January 1992. **Cited** on page 42.

[AK95] J. Adámek and V. Koubek. On the greatest fixed point of a set functor. *Theor. Comput. Sci.*, 150(1):57–75, 1995. **Cited** on page 87.

[AM89] P. Aczel and N. P. Mendler. A final coalgebra theorem. In *Cat. Theor. and Comp. Sci.*, 357–365, 1989. **Cited** on page 13.

[Awo10] S. Awodey. *Category theory*. Oxford Logic Guides, 2010. **Cited** on page 11.

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14:25–59, 1987. **Cited** on page 5.

[BBC$^+$12] F. Bonchi, M. M. Bonsangue, G. Caltais, J. J. M. M. Rutten, and A. Silva. Final semantics for decorated traces. *Electr. Notes Theor. Comput. Sci.*, 286:73–86, 2012. **Cited** on pages 8 and 104.

[BBRS12] F. Bonchi, M. Bonsangue, J. Rutten, and A. Silva. Brzozowski's algorithm (co)algebraically. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, volume 7230 of *Lect. Notes in Comput. Sci.*, 12–23, 2012. **Cited** on pages 6, 10, 88, 96, 97, 98, 101 and 104.

[BCG$^+$11] M. Bonsangue, G. Caltais, E. I. Goriac, D. Lucanu, J. Rutten, and A. Silva. A decision procedure for bisimilarity of generalized regular expressions. In *Proc. of the 13th Brazilian conference on Formal methods: foundations and applications*, SBMF'10, 226–241, 2011. **Cited** on page 7.

[BCG$^+$13] M. M. Bonsangue, G. Caltais, E. I. Goriac, D. Lucanu, J. J. M. M. Rutten, and A. Silva. Automatic equivalence proofs for non-deterministic coalgebras. *Sci. Comput. Program.*, 78(9):1324–1345, 2013. **Cited** on page 7.

[BdV04] J. C. M. Baeten and E. P. de Vink. axiomatizing GSOS with termination. *J. Log. Algebr. Program.*, 60-61:323–351, 2004. **Cited** on page 2.

[BFvG04] B. Bloom, W. Fokkink, and R. J. van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Trans. Comput. Logic*, 5:26–78, January 2004. **Cited** on page 2.

[BG06] M. Boreale and F. Gadducci. Processes as formal power series: A coinductive approach to denotational semantics. *Theor. Comput. Sci.*, 360(1-3):440–458, 2006. **Cited** on pages 10 and 106.

[BIM95] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *JACM*, 42:232–268, 1995. **Cited** on page 2.

[BJM00] A. Bouhoula, J. P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000. **Cited** on page 29.

[BP13] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, 457–468. ACM, 2013. **Cited** on pages 3, 5, 8, 10, 88, 90, 91 and 104.

[BPK12] N. Bezhanishvili, P. Panangaden, and C. Kupke. Minimization via duality. In *WoLLIC' 12*, 191-205, 2012. **Cited** on page 88.

[BRS09] M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. An algebra for Kripke polynomial coalgebras. In *Logic in Comput. Sci.*, 49–58, 2009. **Cited** on page 4.

[Brz62] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathem. Theory of Aut.*, volume 12(6), 529–561. **Cited** on pages 6, 10, 88, 91 and 104.

[BS12] S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra*. Dover Publications, Incorporated, 2012. **Cited** on page 1.

[BSdV04] F. Bartels, A. Sokolova, and E. P. de Vink. A hierarchy of probabilistic system types. *Theor. Comput. Sci.*, 327(1-2):3–22, 2004. **Cited** on pages 69 and 76.

[CDE$^+$07] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All about Maude - a high-performance logical framework: how to specify, program and verify systems in rewriting logic*. Springer-Verlag, Berlin, Heidelberg, 2007. **Cited** on pages 19, 26 and 44.

[CDLT08] F. Calzolai, R. De Nicola, M. Loreti, and F. Tiezzi. TAPAs: A tool for the analysis of process algebras. *T. Petri Nets and Other Models of Concurrency*, 5100:54–70, 2008. **Cited** on pages 5, 10 and 87.

[CGK$^+$13] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse. An overview of the μCRL2 toolset and its recent advances. In *TACAS*, 199–213, 2013. **Cited** on pages 3 and 87.

[CH89] R. Cleaveland and M. Hennessy. Testing equivalence as a bisimulation equivalence. *Lect. Notes in Comput. Sci.*, vol. 407, 11–23, 1989. **Cited** on pages 1, 4, 5, 6, 7, 9, 10, 49, 52, 77, 78, 79, 83, 85, 87, 88, 93, 101, 102 and 103.

[CHL03] D. Cancila, F. Honsell, and M. Lenisa. Generalized coiteration schemata. *Electr. Notes in Theor. Comput. Sci.*, 82(1), 2003. **Cited** on pages 10, 15 and 87.

[CPS93a] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, January 1993. **Cited** on page 3.

[CPS93b] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst. (TOPLAS)*, 15(1):36–72, 1993. **Cited** on pages 5, 10 and 87.

[CS96] R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *CAV*, volume 1102 of *Lect. Notes in Comput. Sci.*, 394–397, 1996. **Cited** on pages 5, 10 and 87.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984. **Cited** on pages 1, 4, 5, 7, 10, 52, 77 and 78.

[Dob08] E. E. Doberkat. Erratum and addendum: Eilenberg-Moore algebras for stochastic relations. *Inf. Comput.*, 206(12):1476–1484, 2008. **Cited** on page 69.

[DR10] L. Doyen and J. F. Raskin. Antichain Algorithms for Finite Automata. In *Proc. TACAS*, volume 6015, 2010. **Cited** on pages 88 and 104.

[DvGHM11] Y. Deng, R. J. van Glabbeek, M. Hennessy, and C. Morgan. Real-reward testing for probabilistic processes (extended abstract). *Electr. Proc. Theor. Comput. Sci.*, 61–73, 2011. **Cited** on page 106.

[Eng85] J. Engelfriet. Determinacy - (observation equivalence = trace equivalence). *Theor. Comput. Sci.*, 36:21–25, 1985. **Cited** on page 18.

[FME05] G.L. Ferrari, U. Montanari, and E.Tuosto. Coalgebraic minimization of HD-automata for the pi-calculus using polymorphic types. *TCS*, 331(2–3):325–365, 2005. **Cited** on page 87.

[GLMS11] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In *TACAS*, 372–387, 2011. **Cited** on page 3.

[GLR00] J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *ASE '00: Proc. of the 15th IEEE international conference on Automated software engineering*, 123–132, 2000. **Cited** on pages 4, 9, 19 and 26.

[GLR10] E. . Goriac, D. Lucanu, and G. Roşu. Automating coinduction with case analysis. In *Proc. of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10, 220–236, 2010. **Cited** on page 34.

[GM92] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992. **Cited** on page 26.

[GS02] H. P. Gumm and T. Schröder. Coalgebras of bounded type. *Mathem. Struct. in Comput. Sci.*, 12(5):565–578, 2002. **Cited** on page 12.

[Hen88] M. Hennessy. *Algebraic theory of processes*. MIT Press, Cambridge, MA, USA, 1988. **Cited** on pages 1, 4, 5, 7, 52, 77 and 78.

[HJ98] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. Comput.*, 145(2):107–152, 1998. **Cited** on page 13.

[HJS07] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Meth. in Comput. Sci.*, 3(4), 2007. **Cited** on page 10.

[HK71] J. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report, Dept. of Computer Science, Cornell Univ., December 1971. **Cited** on page 8.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *JACM*, 32(1):137–161, January 1985. **Cited** on page 106.

[HMS05] D. Hausmann, T. Mossakowski, and L. Schröder. Iterative Circular Coinduction for CoCasl in Isabelle/HOL. In *Lect. Notes in Comput. Sci.*, vol. 3442, 341–356, 2005. **Cited** on page 9.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. **Cited** on page 60.

[Hop71] J. E. Hopcroft. An n log n algorithm for minimizing in a finite automaton. In *Proc. International Symposium of Theory of Machines and Computations*, 189–196, 1971. **Cited** on pages 10, 87 and 91.

[HP85] D. Harel and A. Pnueli. Logics and models of concurrent systems. On the development of reactive systems, 477–498, 1985. **Cited** on page 1.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:62–222, 1997. **Cited** on pages 1, 3 and 4.

[JSS12] B. Jacobs, A. Silva, and A. Sokolova. Trace semantics via determinization. In *CMCS*, 109–129, 2012. **Cited** on page 18.

[JS90] C. C. Jou and S. A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. *CONCUR '90 Theories of Concurrency: Unification and Extension, Lect. Notes in Comp. Sci.*, vol. 458, 367–383, 1990. **Cited** on pages 4, 7, 9, 49, 51, 68, 69, 70, 72, 73 and 85.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, July 1976. **Cited** on pages 2 and 7.

[Kle56] S. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 3–42, 1956. **Cited** on page 9.

[Kli04] B. Klin. A coalgebraic approach to process equivalence and a coinduction principle for traces. *Electr. Notes Theor. Comput. Sci.*, 106:201–218, 2004. **Cited** on page 10.

[Kli09] B. Klin. Bialgebraic methods and modal logic in structural operational semantics. *Inf. Comput.*, 207:237–257, February 2009. **Cited** on pages 3 and 106.

[Kli11] B. Klin. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011. **Cited** on page 96.

[Koz91] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *LICS*, 214–225, 1991. **Cited** on pages 9 and 20.

[Koz01] D. Kozen. Myhill-Nerode relations on automatic systems and the completeness of Kleene algebra. In *STACS. Lect. Notes in Comput. Sci.*, vol. 2010, 27–38, 2001. **Cited** on page 9.

[Koz06] D. Kozen. On the representation of Kleene algebras with tests. In *MFCS*, 73–83, 2006. **Cited** on page 105.

[KS83] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. In *Proc. of the second annual ACM symposium on Principles of distributed computing*, PODC '83, 228–240, 1983. **Cited** on pages 10, 87 and 88.

[Kur00] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Ludwigs-Maximilians-Universität München, 2000. **Cited** on page 87.

[Len99] M. Lenisa. From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. *Electr. Notes Theor. Comput. Sci.*, 19:2–22, 1999. **Cited** on pages 10, 15 and 87.

[LGCR09] D. Lucanu, E. I. Goriac, G. Caltais, and G. Roşu. CIRC: a behavioral verification tool based on circular coinduction. In *Proc. of the 3rd international conference on Algebra and coalgebra in computer science*, CALCO'09, 433–442, 2009. **Cited** on pages 33 and 47.

[LPW00] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electr. Notes Theor. Comput. Sci.*, 33:230–260, 2000. **Cited** on page 10.

[Mea55] G. H. Mealy. A method to synthesizing sequential circuits. *Bell System Technical Journal*, 1045–1079, 1955. **Cited** on page 2.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *J. Comput. System Sci.*, 28(3):439–466, 1984. **Cited** on page 9.

[Mil89] R. Milner. *Communication and concurrency*. Prentice Hall, 1989. **Cited** on pages 1, 3, 4, 13 and 88.

[Mon08] L. Monteiro. A coalgebraic characterization of behaviours in the linear time - branching time spectrum. In *WADT*, 251–265, 2008. **Cited** on page 10.

[Moo56] E. F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, 129–153, 1956. **Cited** on page 2.

[MP81] D. Michael and R. Park. Concurrency and automata on infinite sequences. In *Theor. Comput. Sci.*, 167–183, 1981. **Cited** on pages 1, 3, 4 and 13.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I/II. 100(1):1–77, 1992. **Cited** on page 88.

[MS73] A. R. Meyer and L. J. Stockmeyer. Word problems requiring exponential time. In *Proc. STOC*, 1–9, 1973. **Cited** on page 88.

[Nic87] R. de Nicola. Extensional equivalences for transition systems. *Acta Inf.*, 24(2):211–237, 1987. **Cited** on page 6.

[OH86] E. R. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Inf.*, 23(1):9–66, 1986. **Cited** on page 55.

[ORT09] S. Owens, J. H. Reppy, and A. Turon. Regular-expression derivatives reexamined. *J. Funct. Program.*, 19(2):173–190, 2009. **Cited** on page 45.

[Phi87] I. Phillips. Refusal testing. *Theor. Comput. Sci.*, 50:241–284, 1987. **Cited** on page 65.

[Plo04] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004. **Cited** on pages 2 and 106.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *ICALP*, 15–32, 1985. **Cited** on page 64.

[PS96] J. Parrow and P. Sjödin. Designing a multiway synchronization protocol. *Comput. Communic.*, 19(14):1151–1160, 1996. **Cited** on pages 88 and 103.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. **Cited** on pages 10 and 87.

[Rab80] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput.*, 9(2):273–280, 1980. **Cited** on page 2.

[RB81] W. C. Rounds and S. D. Brookes. Possible futures, acceptances, refusals, and communicating processes. In *FOCS*, 140–149, 1981. **Cited** on page 62.

[RBR13] J. Rot, M. M. Bonsangue, and J. J. M. M. Rutten. Coalgebraic bisimulation-up-to. In *SOFSEM*, 369–381, 2013. **Cited** on pages 8 and 96.

[RL09] G. Roşu and D. Lucanu. Circular coinduction: a proof theoretical foundation. In *Proc. of the 3rd international conference on Algebra and coalgebra in computer science*, CALCO'09, 127–144, 2009. **Cited** on pages 3, 4, 9, 19, 26, 29, 30, 34, 38, 47 and 86.

[Ros00] G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000. **Cited** on page 9.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959. **Cited** on pages 2, 5 and 14.

[RTJ01] J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *J. UCS*, 7(2):175–193, 2001. **Cited** on page 9.

[Rut98] J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In *CONCUR*, 194–218, 1998. **Cited** on page 14.

[Rut00] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. **Cited** on pages 1, 3, 11, 12, 13, 19 and 54.

[Rut03] J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003. (Not cited.)

[Rut05] J. J. M. M. Rutten. A coinductive calculus of streams. *Mathem. Struct. in Comput. Sci.*, 15(1):93–147, 2005. **Cited** on page 2.

[RV07] A. Rensink and W. Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, February 2007. **Cited** on page 106.

[Sal66] A. Salomaa. Two complete axiom systems for the algebra of regular events. *JACM*, 13(1):158–169, 1966. **Cited** on page 9.

[San98] D. Sangiorgi. On the bisimulation proof method. 8:447–479, 1998. **Cited** on pages 5, 8, 10 and 88.

[SBBR10] A. Silva, F. Bonchi, M. M. Bonsangue, and J. J. M. M. Rutten. Generalizing the powerset construction, coalgebraically. In *FSTTCS 2010*, vol. 8 of *LIPIcs*, 272–283, 2010. **Cited** on pages 3, 5, 10, 15, 87 and 103.

[SBBR11] A. Silva, F. Bonchi, M. Bonsangue, and J. Rutten. Quantitative Kleene coalgebras. *Inform. and Comput.*, 209(5):822–849, 2011. **Cited** on page 105.

[SBBR13] A. Silva, F. Bonchi, M. M. Bonsangue, and J. J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Meth. in Comput. Sci.*, 9(1), 2013. **Cited** on pages 5, 7, 10, 17, 49, 52, 53, 54 and 106.

[SBR10] A. Silva, M. M. Bonsangue, and J. J. M. M. Rutten. Non-deterministic Kleene coalgebras. *Logical Meth. in Comput. Sci.*, 6(3), 2010. **Cited** on pages 4, 7, 9, 19, 20, 21, 22, 23, 26, 28, 33, 38, 43, 46 and 47.

[Seg96] R. Segala. Testing probabilistic automata. In *CONCUR*, volume 1119 of *Lect. Notes in Comput. Sci.*, 299–314, 1996. **Cited** on page 106.

[SR11] D. Sangiorgi and J. Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge Tracts in Theoretical Computer Science, 2011. **Cited** on pages 3, 5, 8 and 10.

[Sta11] S. Staton. Relating coalgebraic notions of bisimulation. *LMCS*, 7(1), 2011. **Cited** on page 87.

[TP97] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proc. 12th LICS Conf.*, 280–291, 1997. **Cited** on page 3.

[TV05] D. Tabakov and M. Vardi. Experimental evaluation of classical automata constructions. In *Proc. LPAR*, vol. 3835, 396–411, 2005. **Cited** on page 88.

[Val95] A. Valmari. Failure-based equivalences are faster than many believe. In *Struct. in Conc. Theory*, 326–340, 1995. **Cited** on page 5.

[vG01a] R. J. van Glabbeek. The linear time - branching time spectrum I. The semantics of concrete, sequential processes. In *Handbook of Process Algebra*, 3–99, 2001. **Cited** on pages 1, 2, 3, 4, 5, 7, 9, 10, 49, 53, 54, 55, 60, 62, 64, 85, 94, 104 and 106.

[vG01b] R. J. van Glabbeek. Current trends in theoretical computer science. chapter What is branching time semantics and why to use it?, 469–479, 2001. **Cited** on page 4.

[Wat95] B. W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Eindhoven University of Technology, the Netherlands, 1995. **Cited** on page 88.

[Wat00] B. W. Watson. Directly constructing minimal DFAs: Combining two algorithms by Brzozowski. In *CIAA*, 311–317, 2000. **Cited** on page 88.

[WDHR06] M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV*, volume 4144, 17–30, 2006. **Cited** on pages 8, 10, 88 and 104.

[Wor05] J. Worrell. On the final sequence of a finitary set functor. *TCS*, 338(1-3):184–199, 2005. **Cited** on page 87.

[Wor08] J. Worthington. Automatic proof generation in Kleene algebra. In *Lect. Notes in Comput. Sci.*, vol. 4988, 382–396, 2008. **Cited** on page 105.

[YL92] W. Yi and K. G. Larsen. Testing probabilistic and nondeterministic processes. In *PSTV*, volume C-8 of *IFIP Transactions*, 47–61, 1992. **Cited** on page 106.

Samenvatting (Dutch summary)

Het bestuderen van de semantiek van reactieve systemen (*reactive systems*) is een belangrijke richting binnen de informatica. Reactieve systemen voeren berekeningen uit middels interactie met hun omgeving, en zijn over het algemeen samengesteld uit meerdere parallelle componenten die simultaan taken uitvoeren en met elkaar communiceren. Toepassingen bevinden zich in relatief simpele systemen als rekenmachines en verkoopautomaten, tot programma's die mechanische apparaten zoals auto's, metro's of ruimtevaartuigen aansturen. Aangezien dit soort systemen veel gebruikt worden, en vaak erg complex zijn, is het gebruik van rigoureuze methodes voor specificatie, ontwikkeling, en redenatie over het gedrag van deze systemen een grote uitdaging. Een mogelijke aanpak om reactieve systemen formeel te beschouwen is het gebruik van een gemeenschappelijke taal voor de beschrijving van zowel de implementatie als de specificatie. In dit geval correspondeert verificatie van de implementatie met betrekking tot de specificatie van een reactief systeem met het bewijzen van een vorm van equivalentie/ordening tussen de beschrijvingen in de formele taal.

De doelstelling van dit proefschrift is het benutten van de krachten van een algebraïsch–coalgebraïsch raamwerk voor het modelleren van reactieve systemen en het redeneren over verschillende soorten bijbehorende semantieken op een formele wijze. Daarnaast richt dit proefschrift zich op het afleiden van een aantal verificatie algoritmes die geschikt zijn voor implementatie in geautomatiseerde systemen.

In Hoofdstuk 3 presenteren wij een beslissingsprocedure voor bisimilariteit van een klasse van expressies die oneindige rijen (*streams*), Mealy automaten, en gelabelde transitie systemen, kan beschrijven. Deze procedure is geïmplementeerd in de automatische stellingbewijzer CIRC. Hoofdstuk 4 beschrijft een uniforme coalgebraïsche aanpak voor een collectie van semantieken voor transitiesystemen. Hiervoor gebruiken we een uitbreiding van de klassieke machtsverzameling constructie. In het bijzonder beschouwen we "decorated trace" equivalenties voor gelabelde transitie-, en probabilistische systemen, en (de zogenaamde "must" en "may") "testing"-semantieken voor divergente niet deterministische systemen. De coalgebraïsche aanpak stelt ons in staat te redeneren over de eerdergenoemde begrippen van gedrag equivalentie/ordening in termen van bisimulaties. Verder faciliteert ons raamwerk de constructie van geverifieerde algoritmes die niet aanwezig zijn voor bisimularitei, zoals beschreven in Hoofdstuk 5. In dit hoofdstuk beschrijven we een variatie van Brzozowski's algoritme om eindige automaten te minimaliseren, en een optimalisatie van Hopcroft en Karp's algoritme voor taal semantieken. Beide algoritmes zijn succesvol toegepast voor het redeneren over "decorated trace" en "testing" semantieken. De bijbehorende implementaties kunnen online uitgeprobeerd worden: `http://perso.ens-lyon.fr/damien.pous/brz/`

Summary

One of the research areas of great importance in Computer Science is the study of the semantics of concurrent reactive systems. These are systems that compute by interacting with their environment, and typically consist of several parallel components, which execute simultaneously and potentially communicate with each other. Examples of such systems range from rather simple devices such as calculators and vending machines, to programs controlling mechanical devices such as cars, subways or spaceships. In light of their widespread deployment and complexity, the application of rigorous methods for the specification, design and reasoning on the behaviour of reactive systems has always been a great challenge. One possible approach to formally handle reactive systems is to use a “common language" for describing both the actual implementations and their specifications. When following this technique, verifying whether an implementation and its specification describe the same behaviour reduces to proving some notion of equivalence/preorder between their corresponding descriptions over the chosen language.
The aim of this thesis is to exploit the strengths of a (co)algebraic framework in modelling reactive systems and reasoning on several types of associated semantics, in a uniform fashion. Moreover, we derive a suite of corresponding verification algorithms suitable for implementation in automated tools.
In Chapter 3 we present a decision procedure for bisimilarity of a class of expressions defining systems such as infinite streams, deterministic automata, Mealy machines and labelled transition systems. The procedure is implemented in the automatic theorem prover CIRC. Chapter 4 provides a uniform coalgebraic handling of a series of semantics on transition systems. This is achieved by employing a generalisation of the classical powerset construction for determinising non-deterministic automata. In particular, we deal with decorated trace equivalences for labelled transition systems and probabilistic systems and, (the so-called “must” and “may”) testing semantics for divergent non-deterministic systems. The coalgebraic approach enabled reasoning on the aforementioned notions of behavioural equivalence/preorder in terms of bisimulations. Moreover, our framework facilitated the construction of verification algorithms which are not available for bisimilarity, as shown in Chapter 5. There we provide a variation of Brzozowski's algorithm to minimise finite automata and an optimisation of Hopcroft and Karp's algorithm for language semantics. Both algorithms were successfully applied to reason on decorated trace and testing semantics. The corresponding implementations can be tested online at: `http://perso.ens-lyon.fr/damien.pous/brz/`.

Curriculum vitae

1984 Born on 20 April, Suceava, Romania

2003 – 2007 BSc in Computer Science, Alexandru Ioan Cuza University, Iaşi, Romania
Final thesis title: "The Implementation of a Programming Language in Maude, using Denotational Semantics", supervised by prof. dr. Dorel Lucanu

2007 – 2009 MSc in Computer Science, Alexandru Ioan Cuza University, Iaşi, Romania
Final thesis title: "CIRC: A Behavioural Verification Tool Based on Circular Coinduction – extensions –", supervised by prof. dr. Dorel Lucanu

2010 – 2013 PhD student at Reykjavik University, Iceland, and Radboud University, Nijmegen, the Netherlands
Final thesis title: "Coalgebraic Tools for Bisimilarity and Decorated Trace Semantics", (co-)supervised by prof. dr. Jan Rutten, prof. dr. Anna Ingólfsdóttir, dr. Alexandra Silva and dr. Marcello Bonsangue