Enhanced Coinduction

Jurriaan Rot

Open-access: `https://openaccess.leidenuniv.nl`
Typeset using LaTeX, diagrams generated using XY-PIC
Printed by Ridderprint B.V.
ISBN 978-94-6299-174-3

The author was funded by the Netherlands Organization for Scientific Research (NWO) as part of the project Coinductive Calculi for Regular Expressions (CoRE). The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at Leiden University, and also used facilities of Centrum Wiskunde en Informatica (CWI).

Enhanced Coinduction

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op donderdag 15 oktober 2015
klokke 15:00 uur

door

Jurriaan Cornelis Rot
geboren te Amsterdam
in 1987

Promotores:	prof. dr. J.J.M.M. Rutten (CWI & Radboud Universiteit) prof. dr. F.S. de Boer
Copromotor:	dr. M.M. Bonsangue
Promotiecommissie:	dr. H.H. Hansen (Technische Universiteit Delft) prof. dr. B.P.F. Jacobs (Radboud Universiteit) dr. B. Klin (University of Warsaw) prof. dr. U. Montanari (University of Pisa) prof. dr. F. Arbab (secretaris) prof. dr. J.N. Kok (voorzitter)

Contents

Chapter 1

Introduction

Induction is a proof and definition principle which is standard in mathematics and computer science. Coinduction, its dual, is particularly suitable for defining infinite and circular objects and proving properties about them. It is becoming increasingly clear that coinduction provides a foundation for many infinite structures arising in computer science, and in recent years it has been the subject of intense research activity. Coinductive techniques have been used to reason about process calculi and their behavioural properties (e.g., [Mil89, Par81, AFV01, TP97]), data structures such as a streams or infinite trees [HJ97, Rut03, APTS13], languages and automata [BP13, Rut98a, Jac06a], recursive types [BH98, AC14], potentially infinite data structures in functional languages and theorem provers [BPT15, APTS13, HNDV13, LGCR09], and much more [San12a, SR12, KS14].

The broad spectrum of coinduction is unified by the theory of coalgebras, which is a general approach to state-based systems and infinite behaviour. In this introductory chapter, we discuss the notions of coalgebra and coinduction, and describe the contents of this thesis. First, we give the basic intuition of coinduction through an example, in Section 1.1. Then we provide a short background on coalgebras, in Section 1.2. The main aims and contributions of this thesis are stated in Section 1.3, related work in Section 1.4 and the outline in Section 1.5.

1.1 Coinductive reasoning

Induction is most commonly known as a proof principle involving the natural numbers: to prove that a property holds for all natural numbers, one proves (a) that it holds for 0, and (b) that if it holds for n, then it also holds for the successor $n + 1$. The validity of this proof principle arises from the construction of natural numbers as the *least* set that contains 0 and is closed under the successor function. In general, induction concerns the *least* object satisfying some property, whereas coinduction concerns the *greatest* object satisfying some property (in a suitable universe).

Consider, for instance, the set of finite lists of integers. This set is defined inductively as follows: $[]$ is a list (the empty list), and if n is an integer and l is a list then their concatenation $n : l$ is again a list. The set of lists is by definition the *least* set that contains the empty list and is closed under concatenation. In contrast, the set of streams (infinite sequences) of integers is defined coinductively. A stream s decomposes as $s_0 : s'$ where s_0 is an integer (the head of the stream) and s' is again a stream (the tail). The set of streams is defined as the *greatest* set of which each element decomposes in this way as a head and a tail.

Lists and streams are examples of inductively and coinductively defined objects. We now turn to proof principles. Suppose we have two functions f, g on finite lists that we would like to prove equal. To do so we may prove:

1. $f([]) = g([])$, and
2. for any list l and any natural number n: if $f(l) = g(l)$ then $f(n : l) = g(n : l)$.

If these two conditions are satisfied then $f(l) = g(l)$ for any list l, by the induction proof principle. Intuitively, this proof principle is valid since every list is constructed by concatenating a finite number of elements to the empty list. More precisely, the above two conditions ensure that the set $\{l \mid f(l) = g(l)\}$ of lists on which f and g agree, contains the empty list and is closed under concatenation. Therefore, it contains every list, i.e., $f(l) = g(l)$ for every list l.

Now, suppose f, g are functions on streams. Then the above inductive proof principle does not apply, since streams are not constructed from the empty list. Instead, to prove equality of arbitrary streams s and t, we use the decomposition of streams into heads and tails, and observe that $s = t$ precisely if the heads s_0 and t_0 are equal and their tails s' and t' are again equal. This observation does not help very much yet, but the point is that equality of streams is the *largest* relation R on streams such that for every $(s, t) \in R$:

1. $s_0 = t_0$ (the heads are equal), and
2. $(s', t') \in R$ (the tails are again related).

A relation on streams that satisfies the above two properties is called a *(stream) bisimulation*. The fact that the greatest bisimulation is the equality relation is called the *coinduction proof principle*. By the coinduction proof principle, to prove that any two streams are equal, it suffices to construct a bisimulation that relates them. In particular, if we manage to construct a bisimulation that relates $f(s) = g(s)$ for any stream s, then $f = g$. The coinduction proof principle turns out to be a powerful tool for reasoning about streams (e.g., [Rut03, NR11]).

Bisimulations were first introduced in concurrency theory by Milner and Park, as a behavioural equivalence between processes [Mil80, Par81], providing the foundation for much of the work on concurrency theory that followed. In fact, bisimulations are of interest well beyond the study of processes, as a general coinductively defined equivalence between systems with infinite behaviour, that comes with a suitable proof principle. Indeed, bisimulations are also of interest to reason about streams, automata, and many other models of computation. The general applicability of bisimulation and coinduction is based on the theory of coalgebras, which we explain next.

1.2 Coalgebras

Inspired by Milner's work on concurrent processes, Aczel applied bisimulations in set theory, to define equality between non-wellfounded sets [Acz88]. Aczel's work is based on a coalgebraic presentation of transition systems, and he showed that *final coalgebras* provide models of non-wellfounded sets as well as a mathematical interpretation of Milner's process calculi. Aczel and Mendler then proposed a coalgebraic generalization of bisimulations in terms of homomorphisms between coalgebras, and used this to formulate a general coinductive proof principle for behavioural equivalence [AM89].

The abstract coalgebraic definition of bisimulations in [AM89] and the associated coinductive proof principle formed the start of the development of coalgebra as a mathematical theory of state-based systems. Coalgebras uniformly capture a large class of models of interest, including various kinds of transition systems but also automata and infinite or circular data structures. The common properties of all these models are studied in *universal coalgebra*, as developed systematically by Rutten in [Rut00].

The basic idea is that the type of a coalgebra is given by a functor, which describes the observations and dynamics. From a given functor that models the system type of interest, one canonically obtains an associated notion of homomorphism and bisimulation. Moreover, under mild conditions on the functor there exists a final coalgebra, which provides a canonical domain of behaviours. Every coalgebra has a unique homomorphism into this final coalgebra. The unique existence of such a homomorphism is conceptually identified with a coinductive definition and proof principle [JR12]. Given a coalgebra, the homomorphism into the final coalgebra assigns a semantics to it, which allows for coinductive definitions. The fact that this homomorphism is unique gives rise to a proof method.

As an example, we consider *stream systems*, which are coalgebras for a functor that maps a set X to the product $\mathbb{N} \times X$ with the natural numbers. A stream system consists of a set X of states, an output function $o\colon X \to \mathbb{N}$, and a next state function $t\colon X \to X$. The final coalgebra for the functor under consideration consists of all streams over the natural numbers, together with the functions head and tail. The coalgebraic notions of bisimulation and coinduction for this functor instantiate to stream bisimulations and the associated coinductive proof principle, as described in the previous section. The coinductive definition principle, which states that every stream system has a unique map into the final coalgebra of streams, allows to define streams and operations on them by constructing suitable stream systems [Rut03, HKR14].

All in all, coalgebra allows us to understand and prove properties of models of computation at a high level of abstraction, and instantiate these results to a wide variety of concrete systems. Indeed, the theory of coalgebras is a lively research area, with new perspectives and results for such diverse areas as modal logic, operational semantics, probabilistic systems, infinite data structures and automata theory (see, e.g., [Jac12, Mis15, Sil15, JNRS11] for a recent overview).

1.2.1 Classical and coalgebraic coinduction

A standard formalization of coinduction, which we call *classical* coinduction, is in terms of complete lattices rather than coalgebras [San12a]. We briefly comment on its relation with coalgebras; more details are in Chapter 3 of this thesis. Classical coinduction is based on Knaster-Tarski's theorem, which states that every monotone function f on a complete lattice has a greatest fixed point $\mathsf{gfp}(f)$. The existence of $\mathsf{gfp}(f)$ is a definition principle, the fact that it is the greatest post-fixed point a proof principle. For instance, the bisimilarity relation of a given transition system is the greatest fixed point of a certain function on the complete lattice of relations on the state space. The proof principle then states that bisimilarity is the greatest bisimulation. By varying the function f one obtains different coinductive predicates, such as similarity, weak or branching bisimilarity, divergence of processes, increasing streams, language inclusion of automata, and so on.

Classical coinduction is very general in the kind of coinductive predicates that can be defined, but it is specifically suitable for speaking about properties of a fixed system. In contrast, coinduction as finality of coalgebras which model the systems of interest (e.g., transition systems) carries a different intuition: it yields a structural account of the specific coinductive predicate of bisimilarity and of behavioural equivalence, which is however uniform over all systems of the given type.

The classical approach can be rephrased as a special case of coalgebraic coinduction, by the observation that any preorder, and thus in particular any complete lattice, forms a category. Post-fixed points in the lattice correspond to coalgebras in the associated category, and a greatest fixed point corresponds to a final coalgebra. In this sense, coinductive predicates on a given system are themselves coalgebras, which live in a category of predicates.

To define coinductive predicates in this way as coalgebras in a category of predicates, we need a way of speaking about *properties* or *predicates* on systems. Such a structure of predicates can be given by the categorical notion of *fibrations*. As observed by Hermida and Jacobs [HJ98] and further developed by Hasuo et al. [HCKJ13], fibrations provide the basic infrastructure to define coinductive predicates on coalgebras systematically and uniformly, in terms of a lifting of the functor whose coalgebras are the systems of interest to a category of predicates.

1.3 Enhanced coinduction

The aim of this thesis is to develop methods that simplify and enhance coinductive reasoning, with coalgebra as the framework of choice to obtain generally applicable techniques. Our results are divided into two parts: the first part concerns the coinductive proof method, and the second part concerns coinductive definition techniques.

1.3.1 Coinductive proofs

To prove that two processes are bisimilar, it suffices to construct a bisimulation. However, this can be rather difficult in concrete instances. Already in the early days of bisimulations, Milner proposed a simplified method of proving bisimilarity, which he called *bisimulation up to bisimilarity* [Mil83]. This idea was further developed in the work of Sangiorgi [San98], who proposed several new enhancements of the bisimulation proof method, including *bisimulation up to context*, a powerful technique for reasoning about systems with algebraic structure, such as models of process calculi. The gains of using up-to techniques to prove bisimilarity can be spectacular, sometimes allowing to use proofs based on a singleton rather than an infinite set. Indeed, up-to techniques have been extensively applied and are by now standard in concurrency theory [PS12].

Enhancements of the bisimulation proof method are interesting not only in concurrency theory. As an example, the coalgebraic study of automata [Rut98a] led to a general view on determinization constructions [SBBR10] which has been combined with up-to techniques, culminating in a novel, efficient algorithm for language equivalence of non-deterministic automata [BP13, BP15], a problem that is long known and has been studied extensively. Other examples of up-to techniques outside concurrency theory are their use in stream calculus [Rut05, NR11], theorem proving [EHB13], and decidability of weighted language equivalence [Win15]. Further, in Chapter 2 of this thesis we show how to apply up-to techniques for deterministic automata to reason about calculi on languages.

In this thesis, we introduce a coalgebraic framework of up-to techniques for coinductive predicates, generalizing the enhancements of the proof method for bisimilarity of processes to a wide range of coinductive predicates and a wide range of state-based systems. We prove the soundness of enhancements such as bisimulation up to context, bisimulation up to transitivity and bisimulation up to bisimilarity, at this abstract level. Building on the work of Pous and Sangiorgi [San98, Pou07, PS12], we obtain a modular framework in which up-to techniques can safely be combined to obtain new sound enhancements. To cover not only bisimilarity but also other coinductive predicates, we base our approach on functor liftings in the setting of a fibration, as pioneered by Hermida and Jacobs. We show how to instantiate these results to obtain enhanced proof principles for bisimilarity of weighted automata, streams and deterministic automata, and also for other coinductive predicates such as divergence of processes, language inclusion of weighted automata and similarity of processes.

1.3.2 Coinductive definitions

Coalgebras provide the means for studying the behaviour of state-based systems, and to define and reason about operations on these systems. They yield a natural setting to define the operational semantics of languages and calculi for a wide range of computational models. In this context, the structure or syntax of a language is modelled by *algebras*, whereas the observable behaviour is modelled by *coalgebras* [RT93]. The semantics of the operators of a language is specified in terms of the *interplay* between algebra and coalgebra.

As an example, the terms of a typical process calculus, such as CCS, form a (free) algebra, and the behaviour is given in terms of transition systems. This behaviour is defined by inductively turning the terms into a coalgebra, according to the specification of each of the operators. Often, such specifications are presented in the language of structural operational semantics [AFV01]. For example, the parallel composition operator in CCS is defined by the following rules:

$$\frac{x \xrightarrow{a} x'}{x|y \xrightarrow{a} x'|y} \qquad \frac{y \xrightarrow{a} y'}{x|y \xrightarrow{a} x|y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{\overline{a}} y'}{x|y \xrightarrow{\tau} x'|y'} \tag{1.1}$$

The first rule states that if a process x makes an a-transition to x' then the parallel composition $x|y$ with any process y makes an a-transition to $x'|y$, and the second rule is its converse. The third rule states how processes x and y can synchronize. The above rules specify how the behaviour of the parallel composition operation is determined from the behaviour of its arguments. Such rules define a coalgebra (transition system) on terms, by induction. The semantics of the operator then arises coinductively, as the homomorphism from this coalgebra on terms into the final coalgebra.

It was observed by Turi and Plotkin that the interplay between algebra and coalgebra can be captured elegantly and systematically through the categorical concept of a distributive law [TP97]. In particular, they showed that distributive laws can be presented by *abstract GSOS specifications*, providing a specification format for languages and calculi which is parametric in the type of behaviour and the type of syntax, in which every specification induces a compositional semantics. As a special case, this can be instantiated to the celebrated GSOS format, which is a particular variant of structural operational semantics [TP97, Bar04, BIM95]. However, abstract GSOS has also been instantiated to obtain formats for probabilistic systems [Bar04], weighted systems [Kli09], streams [HKR14, Kli11], and more [Kli11]. Moreover, distributive laws have been used to devise coalgebraic determinization procedures [SBBR10, JSS12, Bar04], for solving recursive equations (e.g., [Jac06b, MMS13]), and they play a crucial role in the enhancements of coinductive proof methods proposed in this thesis.

In the second part of this thesis, we integrate distributive laws with equations. We extend Turi and Plotkin's framework with recursive assignment rules. This allows, for instance, to define the replication operator $!x$ in CCS by the rule

$$\frac{!x|x \xrightarrow{a} t}{!x \xrightarrow{a} t} \tag{1.2}$$

which does not fit the GSOS format, since a GSOS rule can not have complex terms such as $!x|x$ in the premise. Subsequently we show that, using assignment rules, we can express the syntactic format for structural congruences proposed by Mousavi and Reniers [MR05]. Structural congruences are a method to combine transition system specifications with equations. We thus integrate (abstract) GSOS specifications with equations, which allows, for instance, to define the replication operator $!x$ in CCS by the equation $!x = !x|x$, and to replace the two symmetric rules in (1.1) by a single rule and the equation $x|y = y|x$. Our main result is that the interpretation of specifications extended with assignment rules (or equations in Mousavi and Reniers' format) is well-behaved, in the sense that bisimilarity is a congruence and that bisimulation up-to techniques are sound. We thus provide a systematic account of combining distributive laws with structural congruences, which was mentioned as an open problem by Bartels [Bar04, page 166] and Klin [Kli07].

While distributive laws can be useful tools to understand the interaction between algebra and coalgebra, they can also be rather hard to describe. Typically, one tries to apply a general method to obtain them, for example by presenting them using abstract GSOS specifications, or using pointwise liftings of the functor that models the type of behaviour [Jac06b, SBBR13]. However, these approaches do not apply if the algebraic structure is modelled by a monad that is not free and the semantics of interest does not arise from a pointwise lifting. This is the case, for instance, in the coalgebraic presentation of context-free grammars proposed in [WBR13].

We show how to present distributive laws for a monad with an equational presentation as the quotient of a distributive law for the underlying free monad, which can in turn be conveniently described using an abstract GSOS specification. The quotient exists under the condition that the original distributive law preserves the equations of the monad, which essentially means that the congruences generated by the equations are bisimulations. We demonstrate our approach by presenting distributive laws for operations on streams and for context-free grammars in a simple manner.

1.4 Related work

The use of up-to techniques to enhance the bisimulation proof method for transition systems goes back to Milner [Mil83, Mil89]. The first systematic study of soundness of up-to techniques for bisimilarity between processes is due to Sangiorgi [San98]. Sangiorgi's approach to modularly construct sound up-to techniques was then generalized to the setting of coinduction in complete lattices by Pous [Pou07, PS12].

At the coalgebraic level, bisimulation up-to techniques were first studied by Lenisa [Len99, LPW00] and by Bartels [Bar04]. They proved the soundness of the specific technique of bisimulation up to context, under certain hypotheses. Both Lenisa and Bartels explicitly mention techniques such as bisimulation up to bisimilarity as an open problem, and Bartels conjectures that the combination of up-to techniques can be achieved by finding a suitable abstract framework and an associated generalization of Sangiorgi's methods to combine sound up-to techniques (see [Bar04, page 166-167],[Len99, page 18]). In this thesis, we provide precisely such a framework, which covers a wide range of enhancements including up-to bisimilarity, but also many other techniques, and their combinations.

Another coalgebraic approach is due to Luo [Luo06], who adapts Sangiorgi's framework of up-to techniques to prove soundness of several up-to techniques. Further, [ZLL$^+$10] introduces bisimulation up-to where the notion of bisimulation is based on a specification language for polynomial functors. All of the previous works on up-to techniques for coalgebras focus on bisimulations; in contrast, the results in this thesis are developed for general coinductive predicates.

Coinductive definition principles through bialgebraic methods have been an active area of research since the work of Turi and Plotkin. The combination of recursive constructs with bialgebraic semantics was suggested by Plotkin [Plo01] and developed by Klin [Kli04], based on bialgebras in an order-enriched setting. Instead, we only assume an order on the behaviour functor of interest, which allows us to combine abstract GSOS specifications with recursive equations. This combination is the basis of our concrete approach to structural congruences in the bialgebraic setting. Our results on structural congruences build on the work of Mousavi and Reniers [MR05]. While structural congruences are standard and widely used in concurrency theory, Mousavi and Reniers provided the only systematic study of structural congruences so far.

The construction of distributive laws as quotients, which we propose in this thesis, yields an instance of a morphism of distributive laws in the sense of [Wat02]. Quotients of distributive laws are studied in [MM07], with a different aim: they study distributive laws of one monad over another in order to compose these monads. Further, [LPW04] introduces several constructions on distributive laws, including a certain kind of quotient. Our main new contributions are an associated proof principle that ensures that a quotient distributive law exists, a self-contained presentation of all the necessary ingredients, and the application to stream calculus and the coalgebraic approach to context-free languages proposed in [WBR13].

1.5 Outline

In Chapter 2 we prove the soundness of up-to techniques for language equivalence and inclusion, and explain how to use these techniques through a wide range of examples. This chapter requires little background knowledge, and it serves as a self-contained introduction to bisimulation and bisimulation up-to. Chapter 2 is based on:

- [RBR13b] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Coinductive proof techniques for language equivalence. LATA 2013.

- [RBR15] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Proving language inclusion and equivalence by coinduction. To appear in Information and Computation, 2015. (Extended version of [RBR13b].)

Chapter 3 contains preliminaries on coalgebras, coinduction, fibrations, algebras and distributive laws, that form the technical background of the subsequent chapters. It is not necessary to understand all of the preliminaries to proceed with the other chapters. In particular, much of the development of Chapter 4 can be understood without knowledge of distributive laws, and the background material on fibrations is only required in Chapter 5.

In Chapter 4 we introduce bisimulation up-to techniques for coalgebras. We prove the soundness of techniques such as up-to-context, up-to-bisimilarity and up-to-equivalence, and their combinations. To illustrate this theory, we show how to use bisimulation up-to techniques to reason about streams, weighted automata and (non)deterministic automata. The soundness of bisimulation up-to techniques for deterministic automata of Chapter 2 is a special case. Chapter 4 is based on:

- [RBB^{+}15] Jurriaan Rot, Filippo Bonchi, Marcello Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced coalgebraic bisimulation. To appear in Mathematical Structures in Computer Science, 2015.

To a smaller extent, Chapter 4 is based on the predecessor of the above paper:

- [RBR13a] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Coalgebraic bisimulation-up-to. SOFSEM 2013.

In Chapter 5 we generalize the results of Chapter 4 to arbitrary coinductive predicates, based on a fibrational approach to coinductive predicates. Our results in this chapter provide a flexible approach to defining general up-to techniques for coinductive predicates and proving their soundness in a modular way. We instantiate this abstract framework to prove the soundness of up-to techniques for similarity of transition systems, language inclusion of weighted automata, and divergence of processes. Chapter 5 is based on:

- [BPPR14] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. CSL-LICS 2014.

Chapter 6 integrates Turi and Plotkin's approach to abstract GSOS with equations. We show how to interpret recursive equations in this context, and prove that they can be encoded by constructing new specifications. We use this to show how abstract GSOS can be combined with structural congruences, for a particular format of equations. Chapter 6 is based on:

- [RB14] Jurriaan Rot and Marcello Bonsangue. Combining bialgebraic semantics and equations. FoSSaCS 2014.

Further, Chapter 6 is based on an extended version [RB15] which is currently under review.

In Chapter 7 we study distributive laws of monads over functors. We show how to present such distributive laws as quotients of distributive laws involving a free monad, which can in turn be given more easily through abstract GSOS specifications. Chapter 7 is based on:

- [BHKR13] Marcello Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot. Presenting distributive laws. CALCO 2013.
- [BHKR15] Marcello Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot. Presenting Distributive Laws. Logical Methods in Computer Science, 2015.

The papers [BHKR13, BHKR15, BPPR14] are a joint effort between the authors. For the other papers mentioned above [RBR13b, RBR15, RBB+15, RBR13a, RB14], the author of this thesis is responsible for the main ideas, technical development and most of the writing.

Chapter 2

Coinduction for languages

The set of all languages over a given alphabet can be turned into an (infinite) deterministic automaton. The proof principle of *coinduction* asserts that two languages are equal if they are bisimilar in this automaton. Thus, to show equality of languages it suffices to construct a suitable bisimulation. Bisimilarity and coinduction are the basis of a practical proof method for language equality [Rut98a], which has, for example, been used in effective procedures for deciding language equivalence of regular expressions (e.g., [KN12, Rut98a, LGCR09, CS11]).

In the current chapter, we enhance this coinductive proof method using up-to techniques. These techniques allow to prove bisimilarity, and thus language equality, by means of *bisimulations up-to*, which are often smaller and easier to construct than actual bisimulations. We show how to apply up-to techniques through a number of examples, including new proofs of classical results such as Arden's rule [HU79].

The up-to techniques introduced in this chapter are particularly suitable for reasoning about operations and calculi on languages. To achieve a general picture of sound up-to techniques, we consider behavioural differential equations [Rut03], which give a syntactic format for specifying operations in terms of language derivatives. We show that bisimulation up-to can be used for reasoning about any operation defined in this format, and use this to prove properties of the shuffle operator and of languages defined by Boolean grammars.

Deterministic automata and their notion of bisimulation are instances of more general concepts from the theory of coalgebras. Indeed, the current exposition is based on the coalgebraic treatment of automata that was initiated in [Rut98a], and the main results of this chapter can be obtained by instantiating the abstract coalgebraic theory of coinduction up-to developed in subsequent chapters of this thesis. Thereby, the current chapter provides a concrete, self-contained introduction to coinduction and up-to techniques, that requires little background knowledge.

Outline. The next section contains preliminaries on languages and bisimulations. In Section 2.2, we motivate and introduce bisimulation up-to for regular operations. Then in Section 2.3, this is generalized to soundness results for operations given by behavioural differential equations, and applied to several other examples. Further, we give an equivalent, semantic characterization of the class of operations that are definable by behavioural differential equations. In Section 2.4, we treat simulation up-to, to reason about language inclusion. In Section 2.5, we discuss related work.

2.1 Bisimulations and coinduction

Throughout this chapter we assume a fixed alphabet A. The set of words is denoted by A^*, the empty word by ε and the concatenation of words w and v by wv. We let $2 = \{0, 1\}$ be the set of Boolean values, where $0 \leq 1$. A language is a set of words, which we represent as a function from A^* to 2; the set of languages is denoted by 2^{A^*}. We abuse notation and use 0 and 1 to denote the empty language and the language containing only the empty word respectively. Further we let any alphabet letter $a \in A$ denote the language that contains only the letter itself.

A *(deterministic) automaton (DA)* over A is a triple (X, o, t) where X is a set of states, $o\colon X \to 2$ is an output function, and $t\colon X \to X^A$ is a transition function. A state $x \in X$ is *final* or *accepting* if $o(x) = 1$. We do not require X to be finite and fix no initial states.

The classical definition of *bisimulation* applies to labelled transition systems, which, in contrast to deterministic automata, do not have output and may feature non-deterministic branching behaviour[1].

Definition 2.1.1. Let (X, o, t) be a deterministic automaton. A relation $R \subseteq X \times X$ is a *bisimulation* if for any $(x, y) \in R$:

1. $o(x) = o(y)$, and
2. for all $a \in A\colon (t(x)(a), t(y)(a)) \in R$.

The largest bisimulation is denoted by $\sim$ and called *bisimilarity*; if $x \sim y$ then we say x is *bisimilar* to y.

We will instantiate the notion of bisimulation to a special deterministic automaton, whose state space is given by the set of languages 2^{A^*}. The output of a language L is simply $L(\varepsilon)$, that is, a language is an accepting state precisely if it contains the empty word. For any $a \in A$, the a-transition from a language L is given by *language derivative* L_a, defined as follows for all $w \in A^*$:

$$L_a(w) = L(aw)\,.$$

[1] Bisimulation-like techniques have been used earlier in the setting of automata. In fact, the standard reference [Par81] introduces bisimulations for automata rather than transition systems, and Theorem 2.1.2 appears already there. For a historical account of bisimulation and coinduction, see [San12b].

Spelling out Definition 2.1.1, a relation R on languages is a bisimulation on this automaton if for any $(L, K) \in R$:

$$L(\varepsilon) = K(\varepsilon) \text{ and for all } a \in A\colon (L_a, K_a) \in R\,.$$

It turns out that bisimilarity of languages is a characterization of language equality. This is called the *coinductive proof principle*, or simply coinduction [Rut98a]. A more general account of coinduction is given in the next chapter.

Theorem 2.1.2 (Coinduction)**.** *For any two languages L, K:*

$$L \sim K \text{ iff } L = K\,.$$

Proof. For the implication from right to left, one shows that the diagonal relation $\{(L, L) \mid L \in 2^{A^*}\}$ is a bisimulation. For the converse, one can prove that for any languages L, K and any word w: if $L \sim K$ then $L(w) = K(w)$, by (structural) induction on w. □

The above coinduction principle is a concrete proof method: to show that two languages L, K are equal, it suffices to construct a bisimulation containing (L, K).

2.1.1 Regular operations

Consider the regular operations of union $L + K$, concatenation $L \cdot K$ (often written as LK) and Kleene star L^*. These are defined, for all w, as usual: $(L + K)(w) = L(w) \vee K(w)$, $(L \cdot K)(w) = 1$ iff there are v, u such that $w = vu$ and $L(v) = 1 = K(u)$, and $L^* = \sum_{i \geq 0} L^i$, where $L^0 = 1$ and $L^{i+1} = L \cdot L^i$. In order to prove equivalence of expressions involving the above operations, we may use bisimulations, but this requires a characterization of the output (acceptance of the empty word) and the derivatives of operations in terms of their arguments. Such a characterization was given for regular expressions by Brzozowski [Brz64]; we formulate this in terms of languages (see, e.g., [Con71, page 41]).

Lemma 2.1.3. *For any two languages L, K and for any $a, b \in A$:*

$$\begin{array}{ll}
0(\varepsilon) = 0 & 0_a = 0 \\
1(\varepsilon) = 1 & 1_a = 0 \\
b(\varepsilon) = 0 & b_a = \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases} \\
(L + K)(\varepsilon) = L(\varepsilon) \vee K(\varepsilon) & (L + K)_a = L_a + K_a \\
(L \cdot K)(\varepsilon) = L(\varepsilon) \wedge K(\varepsilon) & (L \cdot K)_a = L_a \cdot K + L(\varepsilon) \cdot K_a \\
L^*(\varepsilon) = 1 & (L^*)_a = L_a \cdot L^*
\end{array}$$

Remark 2.1.4. This characterization in terms of output and derivative can equivalently be taken as the *definition* of the operations. This is achieved by constructing a deterministic automaton on the state space of *expressions* over languages. Such definition techniques, which are standard in the theory of coalgebras, are discussed in more detail in Chapter 3.

Example 2.1.5. Let $A = \{a, b\}$. We prove that $(a + b)^* = (a^*b^*)^*$. To this end, we start with the relation $R = \{((a + b)^*, (a^*b^*)^*)\}$ and try to show that it is a bisimulation. So we must show that the outputs of $(a+b)^*$ and $(a^*b^*)^*$ coincide, and that their a-derivatives and their b-derivatives are related by R. Using Lemma 2.1.3, we see that $(a + b)^*(\varepsilon) = 1 = (a^*b^*)^*(\varepsilon)$. Moreover, again using Lemma 2.1.3, we have $((a + b)^*)_a = (a + b)_a(a + b)^* = (1 + 0)(a + b)^* = (a + b)^*$ and $((a^*b^*)^*)_a = (a^*b^*)_a(a^*b^*)^* = ((a^*)_ab^* + a^*(\varepsilon) \cdot (b^*)_a)(a^*b^*)^* = (a^*b^* + 0)(a^*b^*)^* = (a^*b^*)^*$, so the a-derivatives are again related (notice that apart from Lemma 2.1.3, we have used some basic facts about union and concatenation). The b-derivative of $(a + b)^*$ is $(a + b)^*$ itself; however, the b-derivative of $(a^*b^*)^*$ is $b^*(a^*b^*)^*$. But $b^*(a^*b^*)^*$ is equal to $(a^*b^*)^*$, so we are done. For an alternative proof that does not use the latter equality, consider the relation $R' = R \cup \{((a+b)^*, b^*(a^*b^*)^*)\}$. As it turns out, the pair $((a + b)^*, b^*(a^*b^*)^*)$ satisfies the necessary conditions as well, turning R' into a bisimulation. We leave the details as an exercise for the reader, and conclude $(a + b)^* = (a^*b^*)^*$ by Theorem 2.1.2.

Constructing a bisimulation (by hand) often follows the above pattern of using Lemma 2.1.3 to compute outputs and derivatives, extending the relation when necessary, and showing that the outputs are equal and the derivatives related. In the remainder of this chapter, we frequently use Lemma 2.1.3 without further reference to it.

If one restricts to regular languages, then the technique of constructing bisimulations in this manner gives rise to an effective algorithm for checking equivalence (cf. [Brz64, Con71, Rut98a]). However, the above coinductive proof method applies to equations over arbitrary languages, not only to regular ones, and in the next sections we consider many such instances of equations.

2.2 Bisimulation up-to for regular operations

In this section, we introduce an enhancement of the bisimulation proof method for equality of languages. We first illustrate the need for such an enhancement with a few examples. Consider the property $LL^* + 1 = L^*$. In order to prove this identity coinductively, we may try to show that

$$R = \{(LL^* + 1, L^*) \mid L \in 2^{A^*}\}$$

is a bisimulation. Using Lemma 2.1.3, it is easy to see that $(LL^* + 1)(\varepsilon) = L^*(\varepsilon)$ for any language L. Further, for any $a \in A$:

$$(LL^* + 1)_a = L_aL^* + L(\varepsilon)L_aL^* + 0 = L_aL^* = (L^*)_a$$

where the leftmost and rightmost equality are by Lemma 2.1.3, and in the second step we use some standard identities. Now we have shown that the derivatives are *equal*; this does not show that R is a bisimulation, since for that, the derivatives need to be *related by* R. The solution, however, is straightforward. If we augment the relation R as follows:

$$R' = R \cup \{(L, L) \mid L \in 2^{A^*}\}$$

then the derivatives of $LL^* + 1$ and L^* are related by R'; moreover, the diagonal is easily seen to satisfy the properties of a bisimulation as well. This solves the problem, but it is arguably somewhat inconvenient that additional work is required to deal with derivatives that are already equal.

As another motivating example, we consider the relation

$$R = \{(L^*L + 1, L^*) \mid L \in 2^{A^*}\}\,.$$

The derivatives are (using Lemma 2.1.3):

$$(L^*L + 1)_a = L_a L^* L + L_a + 0 = L_a(L^*L + 1) \qquad \text{and} \qquad (L^*)_a = L_a L^*$$

Clearly $L_a L^*$ can be obtained from $L_a(L^*L + 1)$ by replacing $L^*L + 1$ by L^*, and indeed the latter two languages are related by R. However, since these derivatives are not related *directly* by R, this argument does not show R to be a bisimulation. Extending R to an actual bisimulation is possible but requires a bit of work that one would rather skip.

To deal with examples such as the above in a more natural and easy way, we introduce the notion of *bisimulation up to congruence*. This requires the definition of congruence closure.

Definition 2.2.1. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *congruence closure* of R (with respect to $+$, $\cdot$ and *) as the least relation $\equiv$ satisfying the following rules:

$$\frac{L\,R\,K}{L \equiv K} \qquad \frac{}{L \equiv L} \qquad \frac{L \equiv K}{K \equiv L} \qquad \frac{L \equiv K \qquad K \equiv M}{L \equiv M}$$

$$\frac{L_1 \equiv K_1 \qquad L_2 \equiv K_2}{L_1 + L_2 \equiv K_1 + K_2} \qquad \frac{L_1 \equiv K_1 \qquad L_2 \equiv K_2}{L_1 \cdot L_2 \equiv K_1 \cdot K_2} \qquad \frac{L \equiv K}{L^* \equiv K^*}$$

In the sequel, we denote the congruence closure of a given relation R on languages by $\equiv_R$, or $\equiv$ if R is clear from the context.

The first rule ensures that $R \subseteq\, \equiv_R$. The three rules on the right in the first row turn $\equiv_R$ into an equivalence relation. The three rules on the second row ensure that $\equiv_R$ is closed under the operations under consideration, which in particular means that $\equiv_R$ relates languages obtained by (syntactic) substitution of languages related by R. For example, if $(L^*L+1, L^*) \in R$, then we can derive from the above rules that $L_a(L^*L + 1) \equiv_R L_a L^*$.

Definition 2.2.2. A relation $R \subseteq 2^{A^*} \times 2^{A^*}$ is a *bisimulation up to congruence*, or simply a *bisimulation up-to*, if for any pair $(L, K) \in R$:

1. $L(\varepsilon) = K(\varepsilon)$, and
2. for all $a \in A$: $L_a \equiv_R K_a$.

In a bisimulation up to congruence, the derivatives can be related by the congruence $\equiv_R$ rather than the relation R itself, and therefore, bisimulations up-to may be easier to construct than bisimulations. Indeed, to prove that R is a bisimulation up-to, the derivatives can be related by familiar equational reasoning.

A bisimulation up-to is, in general, not a bisimulation. However, as we show below, it *represents* one, in the following sense: if R is a bisimulation up-to, then $\equiv_R$ is a bisimulation.

Theorem 2.2.3. *If R is a bisimulation up to congruence then for any $(L, K) \in R$, we have $L = K$.*

Proof. Let $\equiv$ be the congruence closure of R. We show that any pair (L, K) in $\equiv$ satisfies the properties

1. $L(\varepsilon) = K(\varepsilon)$ and
2. for any $a \in A$: $L_a \equiv K_a$

of a bisimulation, by rule induction on $(L, K) \in \equiv$. This amounts to showing that $\equiv$ is closed under the inference rules of Definition 2.2.2. For the base cases:

1. for the pairs contained in R, the conditions are satisfied by the assumption that R is a bisimulation up-to;
2. the case $L \equiv L$ is trivial.

Now assume languages L, K, M, N such that $L \equiv K$, $M \equiv N$, $L(\varepsilon) = K(\varepsilon)$, $M(\varepsilon) = N(\varepsilon)$ and for all $a \in A$: $L_a \equiv K_a$ and $M_a \equiv N_a$. We need to prove that $(L + M, K + N), (LM, KN), (L^*, K^*)$, (K, L) and (L, N) (if $K = M$) again satisfy the properties of a bisimulation, i.e., $(L + M)(\varepsilon) = (K + N)(\varepsilon)$ and for all $a \in A$: $(L + M)_a \equiv (K + N)_a$, and similarly for the other operations. We treat the case of union: $(L + M)(\varepsilon) = L(\varepsilon) \vee M(\varepsilon) = K(\varepsilon) \vee N(\varepsilon) = (K + N)(\varepsilon)$; moreover by assumption and closure of $\equiv$ under $+$ we have $L_a + M_a \equiv K_a + N_a$, and so $(L + M)_a = L_a + M_a \equiv K_a + N_a = (K + N)_a$.

Concatenation and Kleene star are treated in a similar manner, and symmetry and transitivity are not difficult either. Thus, by induction, $\equiv$ is a bisimulation, so by Theorem 2.1.2 we have $L = K$ for any $L \equiv K$ and for any $(L, K) \in R$, in particular. □

Any bisimulation is also a bisimulation up-to, so Theorem 2.2.3 is a generalization of Theorem 2.1.2 for the case of languages. Consequently, its converse holds as well.

We proceed with a number of proofs based on bisimulation up-to.

Example 2.2.4. Recall the relation $R = \{(L^*L + 1, L^*) \mid L \in 2^{A^*}\}$ from the beginning of this section. As we have seen, the a-derivatives are $L_a(L^*L + 1)$ and L_aL^*, which are not related by R; however they *are* related by $\equiv_R$. So R is a bisimulation up-to, and consequently $L^*L + 1 = L^*$, by Theorem 2.2.3. Moreover, the relation $\{(LL^* + 1, L^*) \mid L \in 2^{A^*}\}$ from the beginning of this section is a bisimulation up-to as well; there, the derivatives are equal and thus related by $\equiv_R$.

Example 2.2.5. In order to prove $M + KL \subseteq L \Rightarrow K^*M \subseteq L$, we use that $K^*M \subseteq L$ if and only if $K^*M + L = L$, and try to prove that the relation

$$R = \{(K^*M + L, L) \mid M + KL \subseteq L;\ L, K, M \in 2^{A^*}\}$$

is a bisimulation up-to. Let L, K, M be such languages and note that $M + KL + L = L$. Since $(M + KL + L)(\varepsilon) = L(\varepsilon)$ it follows that $(M + L)(\varepsilon) = L(\varepsilon)$, so $(K^*M + L)(\varepsilon) = L(\varepsilon)$. For any alphabet letter a we have

$$\begin{aligned}
(K^*M + L)_a &= K_aK^*M + M_a + L_a \\
&= K_aK^*M + M_a + (M + KL + L)_a \\
&= K_aK^*M + M_a + M_a + K_aL + K(\varepsilon)L_a + L_a \\
&= K_a(K^*M + L) + M_a + K(\varepsilon)L_a + L_a \\
&\equiv_R K_aL + M_a + K(\varepsilon)L_a + L_a \\
&= (M + KL + L)_a \\
&= L_a\,.
\end{aligned}$$

In conclusion, R is a bisimulation up-to, proving $M + KL \subseteq L \Rightarrow K^*M + L = L$.

The above approach of dealing with language inclusion by reducing it to equality is, in general, not the most efficient one. In Section 2.4, we introduce *simulation up-to* which allows to deal with inequality more directly, and reprove the above example in a shorter way.

Example 2.2.6. Arden's rule, in a special case[2], states that if $L = KL + M$ for some languages L, K and M, and K does not contain the empty word, then $L = K^*M$. In order to prove its validity coinductively, let L, K, M be languages such that $K(\varepsilon) = 0$ and $L = KL + M$, and let $R = \{(L, K^*M)\}$. Using that $K(\varepsilon) = 0$, we have $L(\varepsilon) = (KL + M)(\varepsilon) = (K(\varepsilon) \wedge L(\varepsilon)) \vee M(\varepsilon) = (0 \wedge L(\varepsilon)) \vee M(\varepsilon) = M(\varepsilon) = 1 \wedge M(\varepsilon) = K^*(\varepsilon) \wedge M(\varepsilon) = K^*M(\varepsilon)$. Further,

$$\begin{aligned}
L_a = (KL + M)_a &= K_aL + K(\varepsilon) \cdot L_a + M_a \\
&= K_aL + M_a \equiv_R K_aK^*M + M_a = (K^*M)_a
\end{aligned}$$

for any $a \in A$. So R is a bisimulation up-to, proving Arden's rule.

While Arden's rule is not extremely difficult to prove without using bisimulations, the textbook proofs are longer and arguably more involved than the above proof, which is not much more than taking derivatives combined with a bit of algebraic reasoning, and does not require much ingenuity. Nevertheless, this coinductive proof is completely precise. Giving a formal proof without using these methods seems non-trivial; see [FS12] for the discussion of a proof within the theorem prover Isabelle.

[2] We consider a more general version of Arden's rule in Section 2.4.

In fact, [Rut98a] already contains a coinductive proof of Arden's rule. However, this is based on a bisimulation, in contrast to our proof, which is based on a bisimulation up-to. Indeed, in [Rut98a] the infinite relation $\{(NL + O, NK^*M + O) \mid N, O \in 2^{A^*}\}$ is used, requiring more work in checking the bisimulation conditions. In that case, one essentially closes the relation $\{(L, K^*M)\}$ under (certain) contexts manually—this happens in a general and systematic fashion in the proof of Theorem 2.2.3.

Example 2.2.7. We prove that for any language L: $LL = 1 \Rightarrow L = 1$ (this property was used in [Koz90] to show that the universal Horn theory of Kleene algebra does not coincide with that of the regular sets). Assume $LL = 1$ and let $R = \{(L, 1)\}$. Since $(LL)(\varepsilon) = 1(\varepsilon) = 1$ also $L(\varepsilon) = 1 = 1(\varepsilon)$. We show that the derivatives of L and 1 are equal, turning R into a bisimulation up-to. First, for any $a \in A$: $L_aL + L_a = L_aL + L(\varepsilon)L_a = (LL)_a = 1_a = 0$. Now, one easily proves that this implies $L_a = 0$ (for example by showing that $\{(K, 0) \mid L + K = 0\}$ is a bisimulation). Thus $L_a = 0 = 1_a$, so $L_a \equiv_R 1_a$.

Example 2.2.8. We prove that $LL = L \Rightarrow L^* = 1 + L$, by establishing a bisimulation up-to (in fact, this example can also easily be proved by induction). To this end, let L be a language with $LL = L$ and consider the relation $R = \{(L^*, 1 + L)\}$. Indeed, $L^*(\varepsilon) = 1 = (1 + L)(\varepsilon)$, and for any $a \in A$: $(L^*)_a = L_aL^* \equiv_R L_a(1 + L) = L_a + L_aL = L_a + L(\varepsilon)L_a + L_aL = L_a + (LL)_a = L_a + L_a = L_a$.

The last example of this section concerns context-free languages. These can be expressed in terms of language equations [GR62]. For example, the language $\{a^nb^n \mid n \in \mathbb{N}\}$ is the unique language L such that $L = aLb + 1$.

Example 2.2.9. Let L, K, M be languages such that $L = aKMb+1$, $K = aMLb+1$ and $M = aLKb + 1$. Without thinking of what possible concrete descriptions of L, K and M can be, we show that $L = K = M$. To this end, let $R = \{(L, K), (K, M)\}$. Obviously $L(\varepsilon) = K(\varepsilon)$ and $K(\varepsilon) = M(\varepsilon)$. Moreover for any alphabet letter b other than a, we have $L_b = 0 = K_b$ and $K_b = 0 = M_b$. For the a-derivatives we have $L_a = KMb \equiv_R MKb \equiv_R MLb = K_a$ and similarly for (M, K); so R is a bisimulation up-to, proving that $L = K = M$.

2.3 Sound operations for bisimulation up-to

In the previous sections, we considered the regular operations on languages, and how their coinductive characterization can be used to prove equalities using bisimulations up-to. Next, we introduce a general syntactic format of operations on languages, and prove that a corresponding notion of bisimulation up-to is sound for any operation that can be characterized within this format. This format consists of a well-defined class of *behavioural differential equations* (BDEs) [Rut98a, Rut03]. More precisely, it is a variant of the *stream GSOS* format given in [HKR14] for stream systems. In fact, our format is a special case of *abstract GSOS* [TP97], a categorical specification format at the level of coalgebras. In the following chapters, we recall abstract GSOS and prove soundness results for up-to techniques at this level, to obtain proof techniques not only for deterministic automata but for arbitrary coalgebras.

After introducing the general soundness results, we show several examples involving language equations with Boolean connectives, and the shuffle product. This section is concluded with a discussion of *causal* functions, which turn out to give a semantic characterization of operations that can be defined by behavioural differential equations [KNR11, HKR14].

A *signature* Σ is a countable set of operator names $\sigma \in \Sigma$ with associated arities[3] $|\sigma| \in \mathbb{N}$. A *language interpretation* of a signature Σ is a set of functions

$$\{\hat{\sigma}\colon (2^{A^*})^{|\sigma|} \to 2^{A^*}\}_{\sigma\in\Sigma}\,.$$

In the sequel, every language interpretation for a signature is of the above type (on languages), and so we simply speak about an *interpretation* and write $\{\hat{\sigma}\}_{\sigma\in\Sigma}$.

Definition 2.3.1. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *congruence closure* $\equiv^{\Sigma}_{R}$ of R w.r.t. an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ as the least relation $\equiv$ satisfying the following rules:

$$\frac{L\,R\,K}{L \equiv K} \qquad \frac{}{L \equiv L} \qquad \frac{L \equiv K}{K \equiv L} \qquad \frac{L \equiv K \qquad K \equiv M}{L \equiv M}$$

$$\frac{L_1 \equiv K_1 \qquad \dots \qquad L_n \equiv K_n}{\hat{\sigma}(L_1,\dots,L_n) \equiv \hat{\sigma}(K_1,\dots,K_n)} \qquad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

R is a *bisimulation up-to (w.r.t.* $\{\hat{\sigma}\}_{\sigma\in\Sigma}$*)*, if for any $(L,K) \in R$:

1. $L(\varepsilon) = K(\varepsilon)$, and
2. for all $a \in A$: $L_a \equiv^{\Sigma}_{R} K_a$.

Bisimulation up-to for the regular operators (Definition 2.2.1) is a special case of the above definition. While Theorem 2.2.3 asserts that bisimulation up-to is a sound proof technique in the case of union, concatenation and Kleene star, this is not the case in general for other operations. This is illustrated by the following example, adapted from [PS12].

Example 2.3.2. Assume for simplicity a singleton alphabet $\{a\}$. Consider the signature that only contains a unary operator h, whose interpretation is defined as follows:

$$\hat{h}(L) = \begin{cases} 0 & \text{if } L = 0 \\ 1 & \text{otherwise} \end{cases}$$

Now notice that $0_a = 0 = \hat{h}(0)$, $a_a = 1 = \hat{h}(a)$, and $0(\varepsilon) = 0 = a(\varepsilon)$. Consequently the relation $R = \{(0,a)\}$ is a bisimulation up-to w.r.t. $\{\hat{h}\}$, whereas $0 \neq a$, so bisimulation up-to with respect to $\{\hat{h}\}$ is not sound.

[3] For notational convenience we assume that all operations have finite arity, but all the results hold for non-finitary operations—such as the infinite sum—as well.

We introduce a condition that guarantees soundness, based on characterizing the operations in terms of BDEs [Rut03]. Informally, this means that one specifies the output of an operation in terms of the outputs of the arguments, and the derivatives as an expression involving the arguments, their derivatives and their outputs. The equations in Lemma 2.1.3 form an example. Indeed, behavioural differential equations are best explained through concrete examples. To prove our soundness theorem, however, we need a precise characterization.

Define the set of *terms* $\Sigma^*(V)$ over a signature Σ and a set of variables V by the grammar

$$t ::= v \mid \sigma(t_1, \ldots, t_n)$$

where v ranges over V, σ ranges over Σ and $n = |\sigma|$. Given an interpretation $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ we define a function

$$I \colon \Sigma^*(2^{A^*}) \to 2^{A^*}$$

by induction: $I(L) = L$ and $I(\sigma(t_1, \ldots, t_n)) = \hat{\sigma}(I(t_1), \ldots, I(t_n))$. Substitution in t of a term u for a variable x is denoted by $t[x := u]$.

Definition 2.3.3. A *(syntactic) behavioural differential equation (BDE)* over a signature Σ for an operator $\sigma \in \Sigma$ of arity n consists of a pair (o, d) of functions of the form

$$o \colon 2^n \to 2 \qquad \text{and} \qquad d \colon A \to \Sigma^*(V_n)$$

where V_n is a set consisting of variables

- $x_1, \ldots, x_n$,
- $x_1^\varepsilon, \ldots, x_n^\varepsilon$ and
- for each $a \in A$ and each $i \leq n$ a variable x_i^a,

all of which are pairwise distinct.

The function o specifies the output of the operation given the output of the arguments, and the function d specifies, for each alphabet letter, the derivative. This derivative is given as a term; intuitively a variable x_i represents the i-th argument of the operation, a variable x_i^ε represents its output, and a variable x_i^a represents the a-derivative of the i-th argument. For instance, the equations for language concatenation in Lemma 2.1.3 would be presented as a behavioural differential equation (o, d), where $o \colon 2 \times 2 \to 2$ is conjunction, and $d(a) = x_1^a \cdot x_2 + x_1^\varepsilon \cdot x_2^a$ for all $a \in A$.

To formalize the intuition that syntactic behavioural differential equations define actual equations on languages, we define for each n a function

$$\begin{aligned} &\rho_n \colon \Sigma^*(V_n) \to ((2^{A^*})^n \to \Sigma^*(2^{A^*})) \\ &\rho_n(t)(L_1, \ldots, L_n) = t[x_i := L_i]_{i \leq n}[x_i^a := (L_i)_a]_{i \leq n, a \in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i \leq n} \end{aligned} \tag{2.1}$$

which substitutes each x_i by L_i, x_i^a by the a-derivative $(L_i)_a$ and x_i^ε by $L(\varepsilon)$. Now, given a function $\hat{\sigma}\colon (2^{A^*}) \to 2^{A^*}$, a BDE (o, d) for σ defines an *equation* for each $a \in A$:

$$\hat{\sigma}(L_1, \ldots, L_n)_a = I(\rho_n(d(a))(L_1, \ldots, L_n)) \tag{2.2}$$

which states in a precise manner that the a-derivative of $\hat{\sigma}(L_1, \ldots, L_n)$ behaves according to the syntactic presentation $d(a)$. For instance, if $\hat{\sigma}$ is language composition and $d(a) = x_1^a \cdot x_2 + x_1^\varepsilon \cdot x_2^a$ then the equation corresponds to

$$(L_1 \cdot L_2)_a = (L_1)_a \cdot L_2 + L_1(\varepsilon) \cdot (L_2)_a \,.$$

If, for an arbitrary operation $\hat{\sigma}$ and BDE (o, d) the equation 2.2 holds and, moreover, the output of $\hat{\sigma}(L_1, \ldots, L_n)$ is given by o applied to the output of its arguments, then we say $\hat{\sigma}$ is *given by* (o, d). This is captured formally by the following definition.

Definition 2.3.4. We say an interpretation $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ *can be given by BDEs* if for each σ (with arity n) there is a BDE (o, d) over Σ such that for all languages $L_1, \ldots, L_n$:

$$\begin{aligned} \hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\ \hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n)) \end{aligned}$$

where ρ^n is defined as in (2.1).

Remark 2.3.5. A behavioural differential equation (o, d) as in Definition 2.3.3 induces for each set X a function

$$d'_X\colon A \to (X^n \times (X^A)^n \times 2^n \to \Sigma^*(X))$$

which is *natural* in X, informally meaning that $d'_X(a)$ is defined uniformly over every set X. This view allows for a neater formalization of presentation by BDEs (Definition 2.3.4) with respect to operations on an *arbitrary* deterministic automaton, which concides with Definition 2.3.4 for the case of the automaton of languages. Since we aim here to use as few technical notions as possible we postpone such a treatment to Section 3.5. There, we recall a general approach to define and study operations and calculi based on the theory of algebras and coalgebras, with behavioural differential equations as a special case.

Lemma 2.1.3 states that the regular operations are captured by BDEs. So the following theorem generalizes the proof principle of Theorem 2.2.3.

Theorem 2.3.6. *If $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ can be given by BDEs, then for any relation R which is a bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma \in \Sigma}$: if $(L, K) \in R$ then $L = K$.*

Proof. Similarly to the proof of Theorem 2.2.3, we show that the congruence closure $\equiv$ of R is a bisimulation, by proving by rule induction that

- $L(\varepsilon) = K(\varepsilon)$ and

- for any $a \in A$: $L_a \equiv K_a$

holds for any $(L, K) \in \equiv$. The base cases, i.e., if $L = K$ or $(L, K) \in R$, are the same as in Theorem 2.2.3.

The rules for symmetry and transitivity are not difficult. We treat the rule for an operator $\sigma \in \Sigma$ of arity $n = |\sigma|$. Let o and d be the functions from Definition 2.3.3 associated to σ which exist since $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ can be given by BDEs, and suppose we have languages $L_1, \ldots, L_n$ and $K_1, \ldots, K_n$ such that for all i:

$$L_i \equiv K_i,\ L_i(\varepsilon) = K_i(\varepsilon) \text{ and for all } a \in A\colon (L_i)_a \equiv (K_i)_a\,. \tag{2.3}$$

Then we have

$$\begin{aligned}\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\ &= o(K_1(\varepsilon), \ldots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)\end{aligned}$$

and for any $a \in A$:

$$\begin{aligned}\hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n)) \\ &= I(d(a)[x_i := L_i]_{i \leq n}[x_i^a := (L_i)_a]_{i \leq n, a \in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i \leq n}) \\ &\equiv I(d(a)[x_i := K_i]_{i \leq n}[x_i^a := (K_i)_a]_{i \leq n, a \in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i \leq n}) \\ &= I(\rho_n(d(a))(K_1, \ldots, K_n)) \\ &= \hat{\sigma}(K_1, \ldots, K_n)_a\end{aligned}$$

where the third step (relation by $\equiv$) holds by the induction hypothesis (2.3). □

Bisimulation up-to with respect to the function $\hat{h}$ of Example 2.3.2 is not sound, as we have seen. Indeed $\hat{h}$ cannot be given by BDEs, since the output $\hat{h}(L)(\varepsilon)$ depends not only on $L(\varepsilon)$ but on the entire language L.

2.3.1 Language equations with complement and intersection

Language complement $\overline{L}$ and intersection $L \wedge K$ are defined as $\overline{L}(w) = \neg(L(w))$ and $(L \wedge K)(w) = L(w) \wedge K(w)$ respectively. Language equations including these additional operations can be used to give semantics to *conjunctive* and *Boolean grammars*, which extend context-free grammars with conjunction and complement [Okh13]. Complement and intersection have a known characterization in terms of outputs and derivatives as well [Brz64]:

Lemma 2.3.7. *For any two languages L, K and for any $a \in A$:*

$$\begin{aligned}&\overline{L}(\varepsilon) = \neg(L(\varepsilon)) && (\overline{L})_a = \overline{(L_a)} \\ &(L \wedge K)(\varepsilon) = L(\varepsilon) \wedge K(\varepsilon) && (L \wedge K)_a = L_a \wedge K_a\end{aligned}$$

The above characterization is in terms of BDEs, so by Theorem 2.3.6 we obtain the soundness of bisimulation up-to.

Example 2.3.8. There are unique languages L and K such that

$$L = aLa + bLb + a + b + 1 \qquad\qquad K = aKa + bKb + aA^*b + bA^*a$$

L is the language of *palindromes*, i.e., words which are equal to their own reverse. We claim that K is the language of all *non*-palindromes, and prove this formally by showing that the relation $R = \{(\overline{L}, K)\}$ is a bisimulation up-to. The outputs are easily seen to be equal: $\overline{L}(\varepsilon) = \neg(L(\varepsilon)) = \neg(1(\varepsilon)) = 0 = K(\varepsilon)$.

$$\begin{aligned}\overline{L}_a &= \overline{L_a} = \overline{La+1} = \overline{La} \wedge \overline{1} = (\overline{L}a + A^*b + 1) \wedge \overline{1} \\ &\equiv_R (Ka + A^*b + 1) \wedge \overline{1} = Ka \wedge \overline{1} + A^*b \wedge \overline{1} + 1 \wedge \overline{1} = Ka + A^*b = K_a\end{aligned}$$

In the fourth step, we unfold the complement $\overline{La}$, the validity of which is itself a nice exercise in bisimulation up-to. Further, the case of b-derivatives is of course similar to the above. So R is a bisimulation up-to, proving that K indeed is the complement of L.

2.3.2 Shuffle (closure)

The *shuffle* operation is defined on words w, v inductively as follows: $w \otimes \varepsilon = \varepsilon \otimes w = w$ and $aw \otimes bv = a(w \otimes bv) + b(aw \otimes v)$ for any alphabet letters a, b. This is extended to languages L, K as $L \otimes K = \sum_{w \in L, v \in K} w \otimes v$. The *shuffle closure* is defined as

$$L^{\circledast} = \sum_{i=0}^{\infty} L^{\otimes i}$$

where $L^{\otimes i}$ is defined inductively by $L^{\otimes 0} = 1$ and $L^{\otimes i+1} = L \otimes L^{\otimes i}$. Notice that the shuffle closure is very similar to the Kleene star; the difference is that here shuffle is used instead of concatenation. Both shuffle and shuffle closure can be characterized in terms of BDEs, as stated by the following lemma.

Lemma 2.3.9. *For any two languages L, K and for any $a, b \in A$:*

$$\begin{array}{ll} (L \otimes K)(\varepsilon) = L(\varepsilon) \wedge K(\varepsilon) & (L \otimes K)_a = L_a \otimes K + L \otimes K_a \\ L^{\circledast}(\varepsilon) = 1 & (L^{\circledast})_a = L_a \otimes L^{\circledast} \end{array}$$

As an example of a proof using bisimulation up-to that involves the shuffle operator, we treat the unfolding of the shuffle closure.

Example 2.3.10. Let L be any language; then $L^{\circledast} = L \otimes L^{\circledast} + 1$. To show this, let $R = \{(L^{\circledast}, L \otimes L^{\circledast} + 1\}$. Then $L^{\circledast}(\varepsilon) = 1 = (L \otimes L^{\circledast} + 1)(\varepsilon)$. Moreover for any alphabet letter a:

$$\begin{aligned}(L^{\circledast})_a &= L_a \otimes L^{\circledast} = L_a \otimes (L^{\circledast} + L^{\circledast}) \\ &\equiv_R L_a \otimes (L^{\circledast} + L \otimes L^{\circledast} + 1) = L_a \otimes (L^{\circledast} + L \otimes L^{\circledast}) \\ &= L_a \otimes L^{\circledast} + L_a \otimes L \otimes L^{\circledast} = L_a \otimes L^{\circledast} + L \otimes L_a \otimes L^{\circledast} \\ &= (L \otimes L^{\circledast} + 1)_a\end{aligned}$$

using Lemma 2.3.9, Lemma 2.1.3 and some basic identities. Thus R is a bisimulation up-to, proving $L^{\circledast} = L \otimes L^{\circledast} + 1$.

2.3.3 Causal functions

The format of BDEs defined in this section is a straightforward extension of the one for streams, given in [KNR11, HKR14]. There, it is shown that functions that can be given by BDEs are exactly those that are *causal*, and vice versa. This result can be extended to the case of languages. As a consequence of Theorem 2.3.6, we then obtain causality of functions as an equivalent, *semantic* condition for soundness of up-to techniques. Here, we assume the alphabet A to be finite.

For any language L and any $k \in \mathbb{N}$ we define $L|_k \in 2^{A^*}$ by

$$L|_k(w) = \begin{cases} L(w) & \text{if } |w| \leq k \\ 0 & \text{otherwise} \end{cases}$$

where $|w|$ is the length of a word w. Define the relation $\approx_k$ as follows:

$$L \approx_k K \text{ iff } L|_k = K|_k \, .$$

A function $\hat{\sigma} \colon (2^{A^*})^n \to 2^{A^*}$ is *causal* if for all languages $L_1, \ldots, L_n$, $K_1, \ldots, K_n$ and for any $k \in \mathbb{N}$:

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \text{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n) \, .$$

Causality means that equality up to length k is a congruence, for any k. In other words, membership in $\hat{\sigma}(L_1, \ldots, L_n)$ of words of length less than k depends only on the words in $L_1, \ldots, L_n$ of length less than k. For example, the function $\hat{h}$ from Example 2.3.2 is not causal: whether or not $\hat{h}(L)$ contains the empty word depends on the entire language L.

Lemma 2.3.11. *The set of all causal functions can be given by BDEs.*

Proof. The core of the proof is that the derivatives of causal functions can be expressed in terms of causal functions again. We only show how this works for a unary function $\hat{\sigma} \colon 2^{A^*} \to 2^{A^*}$; the extension to other arities is straightforward. Let $A = \{a_1, \ldots, a_l\}$ be a finite alphabet. Consider, for an alphabet letter $a \in A$, the function

$$\tilde{\sigma}_a \colon (2^{A^*})^{l+1} \to 2^{A^*}$$

defined as $\tilde{\sigma}_a(M, K_1, \ldots, K_l) = (\hat{\sigma}(M(\varepsilon) + a_1 K_1 + \ldots + a_l K_l))_a$. Then $\tilde{\sigma}_a$ is causal, and it follows that

$$\hat{\sigma}(L)_a = \tilde{\sigma}_a(L(\varepsilon), L_{a_1}, \ldots, L_{a_l}) \, .$$

We have thus expressed the derivative $\hat{\sigma}(L)_a$ in terms of another causal function, which takes the output and derivatives of L as arguments. Further, since $\hat{\sigma}$ is causal, the output $\hat{\sigma}(L)(\varepsilon)$ depends only on $L(\varepsilon)$. □

In order to prove the converse, that is, any operation that can be given by BDEs is causal, we need the following.

Lemma 2.3.12. *Let $k \in \mathbb{N}$. Suppose that for all $\hat{\sigma}$ in some set $\{\hat{\sigma}\}_{\sigma \in \Sigma}$, and for all languages $L_1, \ldots, L_n, K_1, \ldots, K_n$ (where $n = |\sigma|$) we have*

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \textit{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n)\,. \tag{2.4}$$

Then for any list of variables $x_1, \ldots, x_m$, any term $t \in \Sigma^*(x_1, \ldots, x_m)$ over operators in Σ, and any languages $L_1, \ldots, L_m$:

$$L_1 \approx_k K_1, \ldots, L_m \approx_k K_m \quad \textit{implies} \quad I(t[x_i := L_i]_{i \le m}) \approx_k I(t[x_i := K_i]_{i \le m})\,.$$

Proof. Let $L_1, \ldots, L_n, K_1, \ldots, K_n$ be languages such that $L_1 \approx_k K_1, \ldots, L_m \approx_k K_m$, and suppose that (2.4) holds. We prove that $I(t[x_i := L_i]_{i \le m}) \approx_k I(t[x_i := K_i]_{i \le m})$ by structural induction on t.

For the base case, if t is a variable x_j then $I(t[x_i := L_i]_{i \le m}]) = L_j$ and $I(t[x_i := K_i]_{i \le m}) = K_j$, and we need to prove $L_j \approx_k K_j$ which trivially follows from our assumption.

Suppose $I(t_j[x_i := L_i]_{i \le m}) \approx_k I(t_j[x_i := K_i]_{i \le m})$ for all $j \le n$. Then

$$\begin{aligned}
&I(\sigma(t_1, \ldots, t_n)[x_i := L_i]_{i \le m}) \\
&= I(\sigma(t_1[x_i := L_i]_{i \le m}, \ldots, t_n[x_i := L_i]_{i \le m})) \\
&= \hat{\sigma}(I(t_1[x_i := L_i]_{i \le m}), \ldots, I(t_n[x_i := L_i]_{i \le m})) \\
&\approx_k \hat{\sigma}(I(t_1[x_i := K_i]_{i \le m}), \ldots, I(t_n[x_i := K_i]_{i \le m})) \\
&= I(\sigma(t_1[x_i := K_i]_{i \le m}, \ldots, t_n[x_i := K_i]_{i \le m})) \\
&= I(\sigma(t_1, \ldots, t_n)[x_i := K_i]_{i \le m})
\end{aligned}$$

where the second step (relating by $\approx_k$) follows by the induction hypothesis and the assumption (2.4). □

Theorem 2.3.13. *A function $\hat{\sigma} \colon (2^{A^*})^n \to 2^{A^*}$ is causal if and only if it is contained in a set of functions which can be given by BDEs.*

Proof. From left to right, the result follows from Lemma 2.3.11. For the other direction, assume a set of functions given by BDEs. We prove that

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \text{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n) \tag{2.5}$$

for every $\hat{\sigma}$ (with n the arity of $\hat{\sigma}$) in the set and for every k, by induction on k.

Take any $\hat{\sigma}$, with arity n, and given by the BDE (o, d). The base case ($k = 0$) holds since $L_1 \approx_0 K_1$, …, $L_n \approx_0 K_n$ implies

$$\begin{aligned}
\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\
&= o(K_1(\varepsilon), \ldots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)\,.
\end{aligned}$$

Now suppose (2.5) holds for some $k \in \mathbb{N}$, and suppose that we have languages $L_1, \ldots, L_n, K_1, \ldots, K_n$ such that $L_i \approx_{k+1} K_i$ for each $i \le n$. We need to prove:

$$\hat{\sigma}(L_1, \ldots, L_n) \approx_{k+1} \hat{\sigma}(K_1, \ldots, K_n)\,. \tag{2.6}$$

Since $L_i \approx_{k+1} K_i$ by assumption, we have for each $i \leq n$: $L_i \approx_k K_i$, $L_i(\varepsilon) \approx_k K_i(\varepsilon)$ and for each $a \in A$: $(L_i)_a \approx_k (K_i)_a$. By the induction hypothesis (2.5) and Lemma 2.3.12 it follows that, for each $a \in A$:

$$\begin{aligned} &I(d(a)[x_i := L_i]_{i\leq n}[x_i^a := (L_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\leq n}) \\ &\approx_k I(d(a)[x_i := K_i]_{i\leq n}[x_i^a := (K_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i\leq n}) \end{aligned}$$

where d is the function that specifies the derivative of $\hat{\sigma}$, and thus

$$\begin{aligned} \hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n)) \\ &= I(d(a)[x_i := L_i]_{i\leq n}[x_i^a := (L_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\leq n}) \\ &\approx_k I(d(a)[x_i := K_i]_{i\leq n}[x_i^a := (K_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i\leq n}) \\ &= I(\rho_n(d(a))(K_1, \ldots, K_n)) \\ &= \hat{\sigma}(K_1, \ldots, K_n)_a\,. \end{aligned}$$

Since, moreover, $\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)$ we get (2.6) as desired. $\square$

By Theorem 2.3.6 and the above result we directly obtain causality as a sufficient condition for the soundness of bisimulation up-to.

Corollary 2.3.14. *Suppose every function of an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is causal. Then bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is sound, i.e., if R is a bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ then $(L, K) \in R$ implies $L = K$.*

2.4 Simulation (up-to)

So far we have focused on techniques for showing equality of languages. Of course, these methods can also be used to prove language inclusion, since $L \subseteq K$ iff $L + K = K$. However, there is a more direct way: instead of bisimulations, one can construct *simulations*, which in practice turns out to be easier for proving language inclusion.

Definition 2.4.1. Let (X, o, t) be a deterministic automaton. A *simulation* is a relation $R \subseteq X \times X$ such that for any $(x, y) \in R$:

1. $o(x) \leq o(y)$, and
2. for all $a \in A$: $(t(x)(a), t(y)(a)) \in R$.

The difference with bisimulation is that condition (1) is relaxed: if x is a final state then y should be final as well, but if x is not final then the output of y does not matter.

Theorem 2.4.2. *If $R \subseteq 2^{A^*} \times 2^{A^*}$ is a simulation (on the automaton of languages defined in Section 2.1) then for any $(L, K) \in R$: $L \subseteq K$.*

Thus simulation is a concrete proof principle for language inclusion, just like bisimulation is a proof principle for language equality.

Simulation up-to is based on a *precongruence* rather than a congruence closure; the difference is that the precongruence is not symmetric, and it relates L to K whenever L is included in K.

Definition 2.4.3. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *precongruence closure* $\leqq_R^\Sigma$ of R w.r.t. an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ as the least relation $\leqq$ satisfying the following rules:

$$\frac{L\,R\,K}{L \leqq K} \qquad \frac{L \subseteq K}{L \leqq K} \qquad \frac{L \leqq K \qquad K \leqq M}{L \leqq M}$$

$$\frac{L_1 \leqq K_1 \qquad \dots \qquad L_n \leqq K_n}{\hat{\sigma}(L_1, \dots, L_n) \leqq \hat{\sigma}(K_1, \dots, K_n)} \qquad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

R is a *simulation up-to (w.r.t. an interpretation* $\{\hat{\sigma}\}_{\sigma\in\Sigma}$*)*, if for any $(L, K) \in R$:

1. $L(\varepsilon) \leq K(\varepsilon)$, and
2. for all $a \in A$: $L_a \leqq_R^\Sigma K_a$.

Our soundness criterion for bisimulation up-to, which is that the operations can be given by BDEs, turns out not to be strong enough for simulation up-to, as witnessed by the following example.

Example 2.4.4. The complement operation can be given by BDEs (Lemma 2.3.7). Consider the relation $R = \{(aA^*, 0)\}$. We have $(aA^*)(\varepsilon) = 0 = 0(\varepsilon)$. Moreover $(aA^*)_a = A^* = \overline{0}$ and $0_a = 0 = \overline{A^*}$. Since $0 \subseteq A^*$, we have $\overline{0} \leqq_R \overline{A^*}$ and thus $(aA^*)_a \leqq_R 0_a$, showing that R is a simulation up-to. But clearly $aA^* \nsubseteq 0$, so simulation up-to with respect to language complement is not a sound proof principle.

Our solution is to require in addition that the operations under consideration satisfy a monotonicity condition.

Definition 2.4.5. A set $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ of operations is given by *monotone BDEs* if

1. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is given by BDEs, and
2. for each $\sigma \in \Sigma$: the associated (output) function $o\colon 2^n \to 2$ is monotone, i.e., if $o_j \leq u_j$ for all j with $1 \leq j \leq n$ then $o(o_1, \dots, o_n) \leq o(u_1, \dots, u_n)$.

Theorem 2.4.6. *If $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ can be given by monotone BDEs then for any relation R which is a simulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$: if $(L, K) \in R$ then $L \subseteq K$.*

Proof. The proof is mostly that of Theorem 2.3.6. One proves by induction that $\leqq$, the precongruence closure of R, is a simulation. The only difference is the first part of the inductive step, which concerns the output. Suppose $\hat{\sigma}$ is an operation of arity n, from a set $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ of operations given by monotone BDEs, and let o be its output function. Let $L_1, \dots, L_n$ and $K_1, \dots K_n$ be languages such that for all j: $L_j(\varepsilon) \leq K_j(\varepsilon)$. Then

$$\begin{aligned}\hat{\sigma}(L_1, \dots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \dots, L_n(\varepsilon)) \\ &\leq o(K_1(\varepsilon), \dots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \dots, K_n)(\varepsilon)\end{aligned}$$

where we use the assumption that o is monotone. □

Example 2.4.7. The general version of Arden's rule states that, given languages K and M, the *least* solution of $L = KL + M$ is K^*M. Furthermore, if $K(\varepsilon) = 0$ then it is the unique one, as we have seen in Example 2.2.6. For the proof of the general statement, first notice that K^*M is indeed a solution since $K^*M = (KK^*+1)M = KK^*M + M$. To show that it is the least one, let L be any language such that $L = KL + M$ and consider the relation $R = \{(K^*M, L)\}$. Then R is a simulation up-to, since $(K^*M)(\varepsilon) = M(\varepsilon) \leq (KL + M)(\varepsilon) = L(\varepsilon)$ and for any a:

$$\begin{aligned}(K^*M)_a &= K_aK^*M + M_a \leqq_R K_aL + M_a \\ &\subseteq K_aL + K(\varepsilon)L_a + M_a = (KL + M)_a = L_a\,.\end{aligned}$$

Thus K^*M is the least solution.

The reader is invited to formulate and prove a version of Arden's rule, where shuffle and shuffle closure (Section 2.3.2) replace concatenation and Kleene star. Further, in Example 2.2.5 we proved $M + KL \subseteq L \Rightarrow K^*M \subseteq L$ using a bisimulation up-to. The proof using a simulation up-to is the same as (part of) the above proof of Arden's rule. One might expect that $M + LK \subseteq L \Rightarrow MK^* \subseteq L$ has a similar treatment, but due to the asymmetry of the derivative of concatenation the proof is different.

Example 2.4.8. In order to prove $M+LK \subseteq L \Rightarrow MK^* \subseteq L$, consider the relation $R = \{(MK^*, L) \mid M + LK \subseteq L;\ L, K, M \in 2^{A^*}\}$. Let L, K, M be such languages; then $M(\varepsilon) \leq L(\varepsilon)$, so $(MK^*)(\varepsilon) \leq L(\varepsilon)$. For any $a \in A$, we have

$$(MK^*)_a = M_aK^* + M(\varepsilon)K_aK^* = (M_a + M(\varepsilon)K_a)K^*$$

In order to see that this is related by $\leqq_R$ to L_a, we start with our assumption $M+LK \subseteq L$ and compute derivatives: $(M+LK)_a \subseteq L_a$, so $M_a+L_aK+L(\varepsilon)K_a \subseteq L_a$. Reformulating this as $(M_a + L(\varepsilon)K_a) + L_aK \subseteq L_a$, we have

$$((M_a + L(\varepsilon)K_a)K^*, L_a) \in R\,.$$

Since $M(\varepsilon) \leq L(\varepsilon)$ we thus obtain

$$(MK^*)_a = (M_a + M(\varepsilon)K_a)K^* \subseteq (M_a + L(\varepsilon)K_a)K^* \leqq_R L_a$$

as desired, showing that R is a simulation up-to.

We conclude with the soundness of an axiom that concerns the interplay between shuffle and concatenation and that is used, for example, in concurrency theory [HMSW11].

Example 2.4.9. The *exchange law* states that

$$(M \otimes L)(K \otimes N) \subseteq (MK) \otimes (LN)$$

for any languages M, L, K, N. Consider the relation

$$R = \{((M \otimes L)(K \otimes N), (MK) \otimes (LN)) \mid M, K, L, N \in 2^{A^*}\}\,.$$

Then

$$((M \otimes L)(K \otimes N))(\varepsilon) = M(\varepsilon) \wedge L(\varepsilon) \wedge K(\varepsilon) \wedge N(\varepsilon) = ((MK) \otimes (LN))(\varepsilon)$$

and for any alphabet letter a:

$$\begin{aligned}
&((M \otimes L)(K \otimes N))_a \\
&= (M_a \otimes L + M \otimes L_a)(K \otimes N) + (M \otimes L)(\varepsilon)(K_a \otimes N + K \otimes N_a) \\
&= (M_a \otimes L)(K \otimes N) + (M \otimes L_a)(K \otimes N) + (M(\varepsilon) \wedge L(\varepsilon))(K_a \otimes N) \\
&\quad +(M(\varepsilon) \wedge L(\varepsilon))(K \otimes N_a) \\
&\leqq_R (M_a K) \otimes (LN) + (MK) \otimes (L_a N) \\
&\quad +(M(\varepsilon)K_a) \otimes (L(\varepsilon)N) + (M(\varepsilon)K) \otimes (L(\varepsilon)N_a) \\
&\subseteq (M_a K) \otimes (LN) + (MK) \otimes (L_a N) \\
&\quad +(M(\varepsilon)K_a) \otimes (LN) + (MK) \otimes (L(\varepsilon)N_a) \\
&= (M_a K + M(\varepsilon)K_a) \otimes (LN) + (MK) \otimes (L_a N + L(\varepsilon)N_a) \\
&= ((MK) \otimes (LN))_a
\end{aligned}$$

This shows that R is a simulation up-to and proves the exchange law.

The proof in the above example is clearly easier than one where the inclusion would be reduced to checking equality by means of bisimilarity.

2.5 Discussion and related work

We presented *bisimulation up-to* as a proof method for language equivalence, and *simulation up-to* as a proof method for language inclusion. These techniques are sound enhancements of the coinductive proof method based on (bi)simulation, if the operations under consideration adhere to the format of behavioural differential equations presented in this chapter. For the soundness of simulation up-to, the operations additionally need to satisfy a monotonicity condition.

Deterministic automata are coalgebras, and the notions of bisimulation and coinduction introduced in Section 2.1 are instances of general definitions [Rut98a]. The up-to techniques introduced in this chapter are also instances of much more general results developed in subsequent chapters of this thesis. In fact, the format of BDEs can be represented by a distributive law, which immediately proves the soundness (actually, a stronger notion) of bisimulation up-to. Moreover, the monotonicity condition for simulation up-to arises from a result in Chapter 5 that requires that the distributive law lifts to a certain category.

A discussion of related work with respect to more general up-to techniques is postponed to Chapter 5. Relevant in the present context is the work of Bonchi and Pous [BP13], which consists of a new algorithm for checking equivalence of non-deterministic automata based on bisimulation up to congruence, improving the state of the art significantly (see also [HR15]). That algorithm is based on the algebraic structure of the powerset of states, obtained by determinization. Our approach is different in that we consider algebraic structures for *arbitrary* calculi on languages (given by behavioural differential equations). Moreover, we do not focus on the algorithmic aspect, but consider up-to techniques for infinite state systems, in order to prove, e.g., inequalities over arbitrary languages.

Bisimulation up-to techniques have been applied to facilitate coinductive definitions and proofs in Coq [EHB13]. In fact, the latter paper uses causal contexts on streams as a condition for soundness; as shown in [HKR14] (and extended to languages in this chapter), this condition is equivalent to requiring that the operations under consideration can be defined by behavioural differential equations.

Our techniques are more widely applicable than only to regular languages, as we have shown in a number of examples involving equations over arbitrary languages. Nevertheless, we recall some of the related work on checking equivalence of regular expressions, for which a wide range of different tools and techniques has been developed. We only recall the ones most relevant to our work. CIRC [LGCR09] is a general coinductive theorem prover, which can deal with regular expressions. Recently, various algorithms based on Brzozowski derivatives and bisimulations have been implemented in Isabelle [KN12] and formalized in type theory, yielding an implementation in Coq [CS11] (while [CS11] does not mention bisimulations explicitly, their method is based on constructing a bisimulation). There is another Coq implementation of regular expression equivalence, which is based on partial derivatives [MPdS12]. An efficient algorithm for deciding equivalence in Kleene algebra, based on automata but not on derivatives and bisimulations, was recently implemented in Coq as well [BP12]. We refer to [NT14] for an overview and comparison between these approaches. Of course, one can reason about regular expressions in Kleene algebra. This is however a fundamentally different approach than the coinductive techniques of the present chapter. In [Gra05], a proof system for equivalence of regular expressions is presented, based on bisimulations but not on bisimulation up-to. In [HN11], a general coinductive axiomatization of regular expression containment is given, based on an interpretation of regular expressions as types. The authors of [HN11] instantiate their axiomatization with the main coinductive rule from [Gra05]. The focus of [HN11] is on constructive proofs based on parse trees of regular expressions. In contrast, our approach is based on bisimulations between languages.

The presented proof techniques apply to undecidable problems such as language equivalence of context-free grammars. Indeed, automation is not aimed at in this chapter. Nevertheless, the present techniques can be seen as a foundation for novel interactive theorem provers, and extensions of fully automated tools such as [KN12, LGCR09, CS11].

If one works with syntactic *terms*, such as regular expressions, rather than with languages, the notion of *bisimulation up to bisimilarity* becomes relevant. In the corresponding proof method, one relates terms modulo bisimilarity. Since we work directly with languages, in our case this is not necessary, but for dealing with terms our techniques can easily be combined with up-to-bisimilarity—see the subsequent chapters of this thesis for details. Bisimulation up to bisimilarity (alone, without context and equivalence closure) was originally introduced in [Mil83], and in the context of automata and languages *simulation up to similarity* was introduced in [Rut98a].

Chapter 3

Preliminaries

In the previous chapter, we studied coinduction for languages and deterministic automata. Deterministic automata are a special case of the theory of coalgebras, which encompasses coinduction principles for a wide variety of systems. In the remaining chapters we develop theory at this more abstract coalgebraic level, so that the results in Chapter 2 are just one instance, among others. In the current chapter we recall standard notions and results on coalgebras, coinduction and algebras. We assume familiarity with basic concepts from category theory such as functors and natural transformations (see, e.g., [Awo10, Lan98]).

Below, we first fix some basic notation regarding sets, relations, functions and categories. Then we introduce coalgebras, homomorphisms and bisimulations, and discuss examples of coinductive techniques (Section 3.1). We proceed to discuss a more classical interpretation of coinduction, and relate this to the coalgebraic perspective in Section 3.2. This discussion of coinduction is continued in Section 3.3, where we recall an approach to coinduction based on the categorical notion of fibrations. We recall algebras for functors and monads in Section 3.4, and conclude this chapter with a discussion of distributive laws and bialgebras (Section 3.5).

Section 3.3, on coinduction in a fibration, can be challenging to understand without prior knowledge of fibrations. However, in this thesis it is only required for Chapter 5. Moreover, most of Chapter 4 requires only basic concepts on coalgebras (Section 3.1) and algebras (the beginning of Section 3.4).

Most of the material in this chapter is taken from the literature; for more information, see, e.g., [Rut00, JR12, Jac12, Len98] (coalgebras and coinduction), [HJ98, HCKJ13] (coinduction in a fibration), [BW05, Awo10, Tur96] (algebras and monads) and [TP97, Kli11, Bar04] (distributive laws and bialgebras).

Sets. By Set we denote the category of sets and functions. We write 1 for the singleton $\{*\}$, 2 for the two elements set $\{0, 1\}$, $\mathbb{N}$ for the set of natural numbers and $\mathbb{R}$ for the set of real numbers. Given sets X and Y, $X \times Y$ is the Cartesian product of X and Y (with the usual projection maps π_1 and π_2) and $X + Y$ is the coproduct, i.e., disjoint union (with coproduct injections κ_1, κ_2).

Relations. Given a relation $R \subseteq X \times Y$, we write $\pi_1 \colon R \to X$ and $\pi_2 \colon R \to Y$ for its left and right projection, respectively. Given another relation $S \subseteq Y \times Z$ we denote the composition of R and S by $R \circ S$. We let $R^{op} = \{(y,x) \mid (x,y) \in R\}$. The diagonal relation on a set X is $\Delta_X = \{(x,x) \mid x \in X\}$.

Functions. Let $f \colon X \to Y$ be a function. The direct image of a set $S \subseteq X$ under f is denoted simply by $f(S) = \{f(x) \mid x \in S\}$, and the inverse image of $V \subseteq Y$ by $f^{-1}(V) = \{x \mid f(x) \in V\}$. The kernel of f is given by $\mathsf{ker}(f) = \{(x,y) \mid f(x) = f(y)\}$. The pairing of two functions f, g with a common domain is denoted $\langle f, g \rangle$ and the copairing (for functions f, g with a common codomain) is denoted by $[f, g]$. The set of functions from X to Y is denoted by Y^X; if we fix X, this yields a (covariant) functor on Set. The i-fold application of a function $f \colon X \to X$ is denoted by f^i, i.e., $f^0 = \mathsf{id}$ and $f^{i+1} = f \circ f^i$.

Categories. On any category, we write Id for the identity functor, and id_X or simply id for the identity morphism of an object X. The product of categories $\mathcal{C}$ and $\mathcal{D}$ is denoted by $\mathcal{C} \times \mathcal{D}$; an object of $\mathcal{C} \times \mathcal{D}$ is a pair consisting of an object from $\mathcal{C}$ and one from $\mathcal{D}$, and an arrow is a pair of arrows from $\mathcal{C}$ and $\mathcal{D}$ of the matching types. Any two functors $F \colon \mathcal{C} \to \mathcal{D}$ and $G \colon \mathcal{C}' \to \mathcal{D}'$ yield a functor $F \times G \colon \mathcal{C} \times \mathcal{C}' \to \mathcal{D} \times \mathcal{D}'$. We use the same notation for the product of functors (in a category of functors and natural transformations), i.e., given F, G as above so that $\mathcal{C} = \mathcal{C}'$, $\mathcal{D} = \mathcal{D}'$ and $\mathcal{D}$ has products, we let $(F \times G)(X) = FX \times GX$. It should always be clear from the context which meaning of $\times$ is referred to.

Given a set X, $\mathcal{P}(X)$ is the set of subsets of X, and $\mathcal{P}_\omega(X)$ is the set of finite subsets of X. Both $\mathcal{P}$ and $\mathcal{P}_\omega$ extend to functors on Set, defined on functions by direct image: $\mathcal{P}(f)(V) = f(V)$ and $\mathcal{P}_\omega(f)(V) = f(V)$. Given a semiring $\mathbb{S}$, we denote by $\mathcal{M}X$ the set of linear combinations of X with coefficients in $\mathbb{S}$. Formally, it is defined by $\mathcal{M}X = \{\varphi \in \mathbb{S}^X \mid \mathsf{supp}(\varphi) \text{ is finite}\}$, where $\mathsf{supp}(\varphi) = \{x \mid \varphi(x) \neq 0\}$. This extends to a functor $\mathcal{M} \colon \mathsf{Set} \to \mathsf{Set}$, sending $f \colon X \to Y$ to $\mathcal{M}(f)(\varphi) = \lambda y. \sum_{x \in f^{-1}(y)} \varphi(x)$. We often denote a linear combination $\varphi \in \mathcal{M}X$ by a formal sum of the form $\sum s_i x_i$, where $s_i \in \mathbb{S}$ and $x_i \in X$ for all i.

3.1 Coalgebras

A *coalgebra* for a functor $B \colon \mathcal{C} \to \mathcal{C}$, or *$B$-coalgebra*, is a pair (X, δ) where X is an object in $\mathcal{C}$ and $\delta \colon X \to BX$ a morphism. We often refer to X as the carrier or state space, δ as the transition map or transition structure, and B as the behaviour functor. A *(coalgebra) homomorphism* from (X, δ) to (Y, ϑ) is a map $h \colon X \to Y$ such that $\vartheta \circ h = Bh \circ \delta$:

$$\begin{tikzcd} X \arrow[r, "h"] \arrow[d, "\delta"'] & Y \arrow[d, "\vartheta"] \\ BX \arrow[r, "Bh"'] & BY \end{tikzcd}$$

The category of B-coalgebras is denoted by B-coalg.

A B-coalgebra (Z, ζ) is *final* if there exists, for any B-coalgebra (X, δ), a unique homomorphism from (X, δ) to (Z, ζ). Final coalgebras are unique up to isomorphism, therefore we often speak about *the* final coalgebra. In general, a final B-coalgebra does not necessarily exist, but there are mild conditions on B under which it does: for instance, when B is a bounded functor on Set (see, e.g., [Rut00]). The *coinductive extension* of a coalgebra (X, δ) is the unique homomorphism into the final coalgebra. Following [JR12], we make a conceptual identification of (coalgebraic) *coinduction* with the use of finality in categories of coalgebras. As we will see below, the unique existence of morphisms gives rise both to *definition* principles and to *proof* principles.

In the remainder of this section we assume that B is a functor on Set. Given a B-coalgebra (X, δ) and states $x, y \in X$, we say x and y are *behaviourally equivalent* or *observationally equivalent* if there exists a coalgebra homomorphism h from (X, δ) to some B-coalgebra so that $h(x) = h(y)$. In particular, $x, y \in X$ are behaviourally equivalent precisely if they are identified by the coinductive extension of δ. The largest relation on X containing only behaviourally equivalent pairs is called *behavioural equivalence*. We denote this relation by $\approx_\delta$, or simply $\approx$.

Example 3.1.1. We list several examples of coalgebras; see, e.g., [Rut00] for more.

1. Let $BX = A \times X$, for a fixed set A. A B-coalgebra $\langle o, t \rangle \colon X \to A \times X$ is a *stream system* (over A). For each state $x \in X$, we observe an output $o(x) \in A$, and a next state $t(x) \in X$.

   The *final* B-coalgebra is $\langle (-)_0, (-)' \rangle \colon A^\omega \to A \times A^\omega$, where $A^\omega = \{\sigma \mid \sigma \colon \mathbb{N} \to A^\omega\}$ is the set of *streams* over A, and for any stream $\sigma \in A^\omega$: $\sigma_0 = \sigma(0)$ and $\sigma'(n) = \sigma(n+1)$ for all $n \in \mathbb{N}$. The coinductive extension of a stream system $\langle o, t \rangle \colon X \to A \times X$ maps a state x to the stream $(o(x), o(t(x)), o(t(t(x))), \ldots)$.

   Stream systems do not involve termination, and therefore they generate only infinite streams. The final coalgebra of $(A \times \mathsf{Id}) + 1$ consists of all finite and infinite streams over A.

2. A *labelled transition system* over a set of labels A is a coalgebra for the functor $BX = \mathcal{P}(A \times X)$. Indeed, a B-coalgebra consists of a set X of states and a map $\delta \colon X \to \mathcal{P}(A \times X)$ that sends each state to a set of transitions. We write $x \xrightarrow{a} y$ if $(a, y) \in \delta(x)$. Labelled transition systems can equivalently be presented as coalgebras for the functor $(\mathcal{P}-)^A$. A *finitely branching* transition system is a coalgebra for the functor $\mathcal{P}_\omega(A \times \mathsf{Id})$. An *image finite* transition system is a coalgebra for $(\mathcal{P}_\omega -)^A$.

   The functor $\mathcal{P}(A \times \mathsf{Id})$ does not have a final coalgebra, for cardinality reasons (the transition map of any final coalgebra is an isomorphism). Nevertheless, $\mathcal{P}_\omega(A \times \mathsf{Id})$ has a final coalgebra: it consists of (finitely branching) trees edge-labelled in A, and quotiented by strong bisimilarity in the usual sense (see below). Similarly, $(\mathcal{P}_\omega -)^A$ has a final coalgebra given by equivalence classes of trees in which every node has only a finite number of a-successors, for each $a \in A$.

3. Let $BX = 2 \times X^A$. A coalgebra $\langle o, t \rangle \colon X \to BX$ is a deterministic automaton; a state x is accepting if $o(x) = 1$, and x makes an a-transition to y (denoted $x \xrightarrow{a} y$) if $t(x)(a) = y$.

   The final coalgebra for $2 \times \mathsf{Id}^A$ is the deterministic automaton introduced in Section 2.1: its carrier is given by the set 2^{A^*} of all languages over A, a state accepts if the corresponding language contains the empty word, and the transition map is given by language derivative. Given any deterministic automaton $\langle o, t \rangle \colon X \to 2 \times X^A$, the coinductive extension $l \colon X \to 2^{A^*}$ is the usual language semantics, i.e., for any $x \in X$: $l(x)(\varepsilon) = o(x)$ and $l(x)(aw) = l(t(x)(a))(w)$.

   More generally, we can consider *Moore automata*, which are coalgebras for the functor $BX = S \times X^A$, where S is a set of outputs. The carrier of the final coalgebra is S^{A^*}.

4. A *non-deterministic automaton* is a coalgebra for $BX = 2 \times (\mathcal{P}_\omega X)^A$. Given a coalgebra $\langle o, t \rangle \colon X \to 2 \times (\mathcal{P}_\omega X)^A$, for each state $x \in X$, a state is accepting if $o(x) = 1$, and for each $a \in A$ there is a set of next states $t(x)(a)$. We write $x \xrightarrow{a} y$ for $y \in t(x)(a)$.

   The final coalgebra of B does *not* consist of languages. Rather, it consists of trees edge-labelled in A and node-labelled in 2, quotiented by strong bisimilarity. Thus, the branching behaviour of automata is taken into account, and therefore we obtain a finer notion of behavioural equivalence than that arising from the usual language semantics.

5. Let $\mathbb{S}$ be a semiring, and $\mathcal{M}$ the associated functor mapping sets to linear combinations with coefficients in $\mathbb{S}$. A *weighted transition system* is a coalgebra for the functor $BX = (\mathcal{M}X)^A$. A *weighted automaton* is a weighted transition system where states additionally feature output, i.e., a coalgebra for the functor $\mathbb{S} \times (\mathcal{M}-)^A$. Weighted automata accept weighted languages, but the final coalgebra of B distinguishes more, similar to the case of non-deterministic automata; see [BBB$^+$12] for details.

3.1.1 Coinductive definitions

The final B-coalgebra provides a canonical semantics for B-coalgebras. In particular, we can use finality to define operations on the final coalgebra. As an elementary example, consider the functor $\mathbb{R} \times \mathsf{Id}$ of stream systems over the reals, and recall that its final coalgebra is the set of streams $\mathbb{R}^\omega$. To define a pointwise sum on streams, we construct a coalgebra $\langle o, t \rangle \colon \mathbb{R}^\omega \times \mathbb{R}^\omega \to \mathbb{R} \times (\mathbb{R}^\omega \times \mathbb{R}^\omega)$ as follows: $o(\sigma, \tau) = \sigma_0 + \tau_0$ and $t(\sigma, \tau) = (\sigma', \tau')$ (where we use the operations $(-)_0$ and $(-)'$, which form the transition map of the final coalgebra, see Example 3.1.1 (1)). By finality this gives rise to a unique homomorphism h:

$$\begin{tikzcd}
\mathbb{R}^\omega \times \mathbb{R}^\omega \arrow[r, dashed, "h"] \arrow[d, "\langle o,t \rangle"'] & \mathbb{R}^\omega \arrow[d, "\langle (-)_0, (-)' \rangle"] \\
\mathbb{R} \times (\mathbb{R}^\omega \times \mathbb{R}^\omega) \arrow[r, "\mathsf{id}\times h"'] & \mathbb{R} \times \mathbb{R}^\omega
\end{tikzcd}$$

which maps a pair of streams to their pointwise sum.

The above way of coinductively specifying and defining operations on streams is a special case of *behavioural differential equations* [Rut03] (see also Chapter 2), in which an operation is defined by specifying its initial value $(-)_0$ and its derivative $(-)'$. We illustrate this by defining several operators:

Initial value	Differential equation	Name
$(\sigma + \tau)_0 = \sigma_0 + \tau_0$	$(\sigma + \tau)' = \sigma' + \tau'$	sum
$(\sigma \times \tau)_0 = \sigma_0 \cdot \tau_0$	$(\sigma \times \tau)' = \sigma' \times \tau + [\sigma_0] \times \tau'$	convolution product
$[r]_0 = r$	$[r]' = [0]$	constant (for any $r \in \mathbb{R}$)

In the first column, the operations $+$ and $\cdot$ on the right of the equations are the standard operations on $\mathbb{R}$. We associate a set T of *terms* to the above operators, defined by the grammar

$$t ::= \sigma \mid t_1 + t_2 \mid t_1 \times t_2 \mid [r] \tag{3.1}$$

where σ ranges over $\mathbb{R}^\omega$ and $[r]$ ranges over $\{[r] \mid r \in \mathbb{R}\}$. Now the above differential equations specify how to define a stream system $T \to \mathbb{R} \times T$. The unique coalgebra morphism $T \to \mathbb{R}^\omega$ then provides the semantics of the operators [Rut03, HKR14]. In Section 3.5 we will see how to study such coinductive definition methods in a structured, categorical way.

In Chapter 2 we have seen behavioural differential equations for languages; notice that the characterization of union and concatenation of Lemma 2.1.3 resembles the above definition of the sum and convolution product on streams. One difference to the previous chapter is that there, we *characterize* pre-defined operations using differential equations, whereas here we use the differential equations to *define* the operations.

Two more operations on streams, which we study in the next chapter, are *shuffle* and *shuffle inverse*:

Initial value	Differential equation	Name
$(\sigma \otimes \tau)_0 = \sigma_0 \cdot \tau_0$	$(\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau'$	shuffle product
$(\sigma^{-1})_0 = (\sigma_0)^{-1}$	$(\sigma^{-1})' = -\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1})$	shuffle inverse

The inverse is only defined on streams σ for which $\sigma_0 \neq 0$. We abbreviate $[-1] \otimes \sigma$ by $-\sigma$. The set of terms involving sum, shuffle product and inverse can be defined as before by a grammar. However, since the inverse is only defined when $\sigma_0 \neq 0$, it is not directly clear how to turn the set of terms into a stream system. We call a term *well-formed* if the inverse is never applied to a subterm with initial value 0; this notion can be straightforwardly defined by induction, and we let T_{wf} be the set of well-formed terms. This set can now be turned into a stream system by induction, using the above specification.

A different use of coalgebras is to study *determinization* constructions at an abstract level, so that language semantics arises by finality [JSS12, SBBR13, Rut00]. Consider a non-deterministic automaton $\langle o, t\rangle \colon X \to 2 \times (\mathcal{P}_\omega X)^A$. As discussed in Example 3.1.1, the coinductive extension of such an automaton does not map a state to the language it accepts. However, we can turn this coalgebra into a deterministic automaton

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

according to the standard powerset construction. This is a deterministic automaton, and the language accepted by a singleton $\{x\}$ is precisely the language accepted by the state x of the original non-deterministic automaton.

For another example of a determinization construction, consider a weighted automaton $\langle o, t\rangle \colon X \to \mathbb{S} \times (\mathcal{M}X)^A$. This induces a Moore automaton $\langle o^\sharp, t^\sharp\rangle \colon \mathcal{M}X \to \mathbb{S} \times (\mathcal{M}X)^A$ where $o^\sharp \colon \mathcal{M}X \to \mathbb{S}$ and $t^\sharp \colon \mathcal{M}X \to (\mathcal{M}X)^A$ are the linear extensions of o and t. By finality, a unique coalgebra homomorphism $l \colon \mathcal{M}X \to \mathbb{S}^{A^*}$ arises, which corresponds to the language semantics of weighted automata. For a detailed explanation see [BBB$^+$12, Section 3] and Example 3.5.2.

3.1.2 Bisimulations and coinductive proofs

The definition of coalgebra homomorphisms provides us with a canonical notion of behavioural equivalence. However, this does not directly give us associated proof techniques, other than the rather abstract property that coinductive extensions are unique. A more concrete proof method is provided by the notion of bisimilarity, which is another fundamental part of the theory of coalgebras. Next, we introduce bisimulations and show a number of concrete examples, and subsequently relate bisimilarity to behavioural equivalence.

A relation $R \subseteq X \times Y$ is a *bisimulation* between coalgebras (X, δ) and (Y, ϑ) if there exists a transition map $\gamma \colon R \to BR$ such that the projections π_1 and π_2 of R are coalgebra homomorphisms, which means that the following diagram commutes [AM89]:

$$\begin{tikzcd} X \arrow[d, "\delta"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & Y \arrow[d, "\vartheta"] \\ BX & BR \arrow[l, "B\pi_1"] \arrow[r, "B\pi_2"'] & BY \end{tikzcd}$$

If $(X, \delta) = (Y, \vartheta)$ then we call R a bisimulation *on* (X, δ). The greatest bisimulation on a given coalgebra (X, δ) is called *bisimilarity* and is denoted by $\sim_\delta$, or simply $\sim$ if δ is clear from the context.

Example 3.1.2.

1. Let $\langle o,t\rangle\colon X \to A \times X$ be a stream system. A relation $R \subseteq X \times X$ is a bisimulation if for all $(x,y) \in R$: $o(x) = o(y)$ and $(t(x), t(y)) \in R$.

   As an example, let T be the set of terms as defined in Equation (3.1), and let $\langle (-)_0, (-)' \rangle\colon T \to \mathbb{R} \times T$ be the stream system defined by the corresponding behavioural differential equations. Let us prove that $s + u \sim u + s$ for any streams s, u. To this end, consider the relation $R = \{(s+u, u+s) \mid s, u \in \mathbb{R}^\omega\}$. For any s, u we have $(s + u)_0 = s_0 + u_0 = u_0 + s_0 = (u + s)_0$. Moreover, $(s + u)' = (s' + u')\ R\ (u' + s') = (u + s)'$. Thus, R is a bisimulation. As we will see below in a more general fashion, this implies that s and u are mapped to the same element in the final coalgebra, meaning that they are assigned the same behaviour. Commutativity of the sum is admittedly a rather trivial property, but it serves here to illustrate the basic methodology of constructing a bisimulation. For many examples of such proofs for streams, see [Rut03, HKR14]; we will also see more advanced proofs in Section 4.2.

2. On labelled transition systems, bisimilarity coincides with the classical notion of strong bisimilarity introduced by Milner and Park [Mil80, Par81]. Given $\delta\colon X \to \mathcal{P}(A \times X)$, a relation $R \subseteq X \times X$ is a bisimulation if for all $(x,y) \in R$: if $x \xrightarrow{a} x'$ then there is y' such that $y \xrightarrow{a} y'$ and $(x', y') \in R$; and if $y \xrightarrow{a} y'$ then there is x' such that $x \xrightarrow{a} x'$ and $(x', y') \in R$.

3. Let $\langle o,t\rangle\colon X \to S \times X^A$ be a Moore automaton. A relation $R \subseteq X \times X$ is a bisimulation if for all $(x,y) \in R$: $o(x) = o(y)$ and for all $a \in A$: $(t(x)(a), t(y)(a)) \in R$. The notion of bisimulation on deterministic automata (Definition 2.1.1) is a special case, and a concrete example of such a bisimulation is in Example 2.1.5.

4. Let $\delta\colon X \to X + 1$ be a coalgebra (for the functor $BX = X + 1$). A relation $R \subseteq X \times X$ is a bisimulation if for any pair $(x,y) \in R$: either $\delta(x) = * = \delta(y)$ or $(\delta(x), \delta(y)) \in R$.

Coalgebra homomorphisms preserve bisimilarity.

Lemma 3.1.3 ([Rut00], Lemma 5.3). *Suppose $f\colon X \to Y$ and $g\colon X \to Z$ are coalgebra homomorphisms. If $R \subseteq X \times X$ is a bisimulation then $(f \times g)(R)$ is a bisimulation.*

If the functor B preserves weak pullbacks, then the inverse image of a bisimulation along a coalgebra homomorphism is again a bisimulation [Rut00, Lemma 5.9]. Thus, in that case, homomorphisms also reflect bisimilarity.

The uniqueness of morphisms into the final coalgebra is, by Lemma 3.1.3 and the fact that the diagonal relation on any coalgebra is a bisimulation [Rut00, Proposition 5.1], equivalent to the following property.

Theorem 3.1.4. *Suppose B has a final coalgebra (Z, ζ). For any $x, y \in Z$:*

$$x \sim y \text{ iff } x = y\,.$$

This is sometimes called strong extensionality, the coinductive proof principle or simply coinduction. Together with Lemma 3.1.3, it entails that, given the bisimilarity relation $\sim$ on any coalgebra:

$$x \sim y \text{ implies } h(x) = h(y) \tag{3.2}$$

where h is the coinductive extension of that coalgebra. Thus, in order to prove that two states have the same behaviour, it suffices to construct a bisimulation.

Example 3.1.5. The foundation of the previous chapter is its coinduction principle Theorem 2.1.2, which states that bisimilarity of languages implies their equality. Indeed, languages form the final coalgebra for the functor $BX = 2 \times X^A$ of deterministic automata, and thus that coinduction principle is an instance of Theorem 3.1.4. Further, Equation (3.2) asserts that bisimilarity on any deterministic automaton implies behavioural equivalence. This means that, to prove that two states of an arbitrary deterministic automaton accept the same language, it suffices to prove that they are bisimilar.

If the functor B preserves weak pullbacks, then homomorphisms reflect bisimilarity, and thus together with Theorem 3.1.4 it implies the converse of (3.2).

Lemma 3.1.6. *If B preserves weak pullbacks then bisimilarity and behavioural equivalence coincide, on any B-coalgebra.*

As an example, the functor $BX = 2 \times X^A$ preserves weak pullbacks. Consequently, two states of a deterministic automaton accept the same language if and only if they related by a bisimulation.

Weak pullback preservation is a mild condition: for instance, it is satisfied by all functors mentioned in Example 3.1.1, except weighted automata and weighted transition systems. For weighted systems, weak pullback preservation only holds under certain conditions on the semiring [GS01, Kli09, BBB$^+$12]. In the cases where it does not hold, behavioural equivalence seems to be of more interest.

3.2 Classical and coalgebraic coinduction

A standard formalization of coinduction is in terms of complete lattices. This is, for instance, the basis of Sangiorgi's introductory text on coinduction [San12a]. This perspective on coinduction, which we call *classical* coinduction (as opposed to coalgebraic coinduction) also plays an important role in this thesis, therefore we recall the basics. In this section we also see how to define coalgebraic bisimulations in the lattice-theoretic setting, and how classical coinduction is generalized by coalgebraic coinduction, i.e., the finality principle in categories of coalgebras.

The starting point is a *complete lattice*: a partial order $(L, \leq)$ in which each subset of L has both a least upper bound and a greatest lower bound. Given a function $f \colon L \to L$, an element $x \in L$ is a fixed point of f if $f(x) = x$, and a post-fixed point if $x \leq f(x)$. If f is monotone (that is, $x \leq y$ implies $f(x) \leq f(y)$) then by the Knaster-Tarski theorem it has a greatest fixed point $\mathsf{gfp}(f)$, which is also the greatest post-fixed point (see, e.g., [San12a]).

The existence of a greatest fixed point constitutes a coinductive *definition* principle: we call $\mathsf{gfp}(f)$ the *coinductive predicate* defined by f. The fact that it is the greatest post-fixed point constitutes a coinductive *proof* principle: to prove that $x \leq \mathsf{gfp}(f)$, it suffice to show that $x \leq f(x)$. In the sequel we shall sometimes refer to post-fixed points of f as f-*invariants*.

Example 3.2.1. Consider the lattice $L = \mathcal{P}(A^\omega \times A^\omega)$ consisting of relations on streams, ordered by inclusion. Define the monotone function $f\colon L \to L$ by

$$f(R) = \{(\sigma, \tau) \mid \sigma_0 = \tau_0 \text{ and } (\sigma', \tau') \in R\}$$

where $(-)_0$ and $(-)'$ form the transition structure of the final stream system, as in Example 3.1.1 (1). A relation R is an f-invariant (post-fixed point of f) precisely if it is a bisimulation on the final stream system. Since f is monotone, the coinductive predicate (the greatest fixpoint) exists: it is given by bisimilarity on the final coalgebra of stream systems, that is, the diagonal relation on streams. The coinductive proof principle asserts that any bisimulation is contained in bisimilarity.

Notice that the above example can be adapted to define bisimilarity on any stream system with carrier X, by replacing $A^\omega \times A^\omega$ by $X \times X$, and replacing $(-)_0$ and $(-)'$ in the definition of f by the transition map of the stream system under consideration. Classical coinduction easily accommodates other predicates than bisimilarity, as shown by a few basic examples below.

Example 3.2.2.

1. Let $\langle o, t\rangle\colon X \to A \times X$ be a stream system where A is equipped with a partial order $\leq$, and consider the lattice $\mathcal{P}(X \times X)$ of relations on X ordered by inclusion. We define a monotone function $f\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ on this lattice:
$$f(R) = \{(x, y) \mid o(x) \leq o(y) \text{ and } (t(x), t(y)) \in R\}\,.$$
A relation R is an f-invariant if for all $(x, y) \in R$, we have $o(x) \leq o(y)$ and $(t(x), t(y)) \in R$. The coinductive predicate defined by f is the greatest such relation. Two states $x, y \in X$ are related by this coinductive predicate if the stream generated by x is pointwise less than the stream generated by y.

2. Let $\langle o, t\rangle\colon X \to A \times X$ be a stream system. Consider the lattice $\mathcal{P}(X)$ of subsets of X, ordered by inclusion, and define the monotone function $f\colon \mathcal{P}(X) \to \mathcal{P}(X)$ on this lattice as follows:
$$f(P) = \{x \mid o(x) \leq o(t(x)) \text{ and } t(x) \in P\}\,.$$
Then an f-invariant is a set $P \subseteq X$ so that for all $x \in P$: $o(x) \leq o(t(x))$, and $t(x) \in P$. The coinductive predicate defined by f, which is the largest f-invariant, thus captures increasing streams.

3. Let $\langle o, t \rangle \colon X \to 2 \times X^A$ be a deterministic automaton, and consider the monotone function f on the lattice of relations on X, defined as follows:

$$f(R) = \{(x, y) \mid o(x) \leq o(y) \text{ and } \forall a \in A.\, (t(x)(a), t(y)(a)) \in R\}$$

A relation R is an f-invariant precisely if it is a *simulation* (Definition 2.4.1). The coinductive predicate defined by f is *similarity*, the greatest simulation.

3.2.1 Coalgebraic bisimulations via relation lifting

In Example 3.2.1 we have seen how to capture bisimulations on stream systems as invariants for a monotone function. Next, we recall a general method of defining a monotone operator on the lattice of relations on the state space of a given coalgebra, so that the coinductive predicate defined by this monotone operator is bisimilarity. This approach was introduced in [HJ98, Rut98b] (see also [Jac12]; and see [Sta11] for a comparison of different notions of bisimulations).

For a functor $B \colon \mathsf{Set} \to \mathsf{Set}$, the (canonical) *relation lifting* $\mathsf{Rel}(B)$ of B maps a relation on X to a relation on BX (for any X). It is defined as follows:

$$\mathsf{Rel}(R \subseteq X \times X) = \{(x, y) \in BX \times BX \mid \exists z.\, B\pi_1(z) = x \text{ and } B\pi_2(z) = y\}$$

where π_1, π_2 are the projections of R. Thus, $\mathsf{Rel}(B)$ is the image of BR under $\langle B\pi_1, B\pi_2 \rangle$. For certain classes of functors there are concrete, inductively defined characterizations of relation lifting [HJ98, Jac12].

Now, given a coalgebra $\delta \colon X \to BX$ we define a function

$$\mathsf{b}_\delta = (\delta \times \delta)^{-1} \circ \mathsf{Rel}(B) : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X) \tag{3.3}$$

on the lattice of relations on X ordered by inclusion. Invariants of the function b_δ are bisimulations on δ (defined as in Section 3.1.2), as stated below.

Lemma 3.2.3. *A relation $R \subseteq X \times X$ on the carrier of a coalgebra $\delta \colon X \to BX$ is a bisimulation if and only if $R \subseteq \mathsf{b}_\delta(R)$. Bisimilarity on (X, δ) is the greatest fixed point of b_δ.*

A bisimulation is a relation with a transition *structure*, whereas a b_δ-invariant is a relation with a special *property*. This formulation is taken from [Jac12], to which we refer for a more elaborate comparison. Lemma 3.2.3 asserts that both characterizations are equivalent.

Relation lifting satisfies a number of properties that are be used in subsequent chapters; see [Jac12, Section 4.4] for proofs.

Lemma 3.2.4. *For any functor $B \colon \mathsf{Set} \to \mathsf{Set}$:*

1. $\mathsf{Rel}(B)(\Delta_X) = \Delta_{BX}$.
2. *If $R \subseteq S$ then* $\mathsf{Rel}(B)(R) \subseteq \mathsf{Rel}(B)(S)$.
3. $(\mathsf{Rel}(B)(R))^{op} = \mathsf{Rel}(B)(R^{op})$.

4. $\mathsf{Rel}(B)(R \circ S) \subseteq \mathsf{Rel}(B)(R) \circ \mathsf{Rel}(B)(S)$.

5. $\mathsf{Rel}(B)((f \times f)^{-1}(S)) \subseteq (Bf \times Bf)^{-1}(\mathsf{Rel}(B)(S))$.

If B preserves weak pullbacks, then the inclusions in items 4 and 5 are equalities.

As a consequence of item 2 above, b_δ is monotone.

Theorem 3.2.5. *Let $B\colon \mathsf{Set} \to \mathsf{Set}$ be a functor. The following are equivalent:*

1. *B preserves weak pullbacks.*

2. *$\mathsf{Rel}(B)$ preserves composition, i.e., $\mathsf{Rel}(B)(R \circ S) = \mathsf{Rel}(B)(R) \circ \mathsf{Rel}(B)(S)$.*

This is originally due to Trnková [Trn80]; for an accessible proof, see [Jac12, Theorem 4.4.6] or [KKV12, Fact 3.6].

3.2.2 Classical coinduction in a category

Classical coinduction can be phrased in terms of categories, via the basic observation that any preorder $(X, \leq)$ (and thus in particular any complete lattice) forms a category, whose set of objects is X, and which has an arrow from x to y if and only if $x \leq y$. A *functor* F on such a category is a monotone function on the preorder, and F-coalgebras are post-fixed points of F (seen as a monotone function). The *final* F-coalgebra then corresponds to the coinductive predicate defined by (the monotone map) F (see, e.g., [NR09, HCKJ13]).

The *definition* principle of classical coinduction here is reformulated to the definition of a final F-coalgebra, whereas the *proof* principle is the existence of a morphism from any F-coalgebra into the final coalgebra. In this setting, we will often refer to F-coalgebras as *F-invariants*. Instantiated to a lattice, the finality principle entails that any F-invariant is below the coinductive predicate defined by F. In this sense, the identification of coinduction with finality in categories of coalgebras still applies in this setting. However, the definition and proof principles carry a significantly different intuition than those discussed in Section 3.1; there, we were defining maps into the final coalgebra and reasoning about them, whereas here we take the final coalgebra itself as the defined object of interest, and the existence of arrows as a proof principle.

Example 3.2.6. Let Pred_X be the category of predicates on a fixed set X, as given by the complete lattice of subsets of X. Let $\delta\colon X \to \mathcal{P}_\omega(A \times X)$ be a labelled transition system, where the set of labels A contains a distinguished element $\tau \in A$. We define a functor $F\colon \mathsf{Pred}_X \to \mathsf{Pred}_X$ by

$$F(P \subseteq X) = \{x \in X \mid \exists y.(\tau, y) \in \delta(x)\}\,. \tag{3.4}$$

An F-invariant (F-coalgebra) is a predicate $P \subseteq X$ so that for any $x \in P$, there exists a τ-transition into a state that is again in P, that is, there is $y \in P$ such that $(\tau, y) \in \delta(x)$. The coinductive predicate defined by F is simply the greatest fixed point of F seen as a monotone function; thus, it is the largest subset of states $x \in X$ that *may diverge*, that is, states that have an infinite path of τ steps. In terms of modal logic, these are the states that satisfy $\nu u.\langle\tau\rangle u$.

In the above example, the *system* of interest is modelled by a coalgebra for the functor $\mathcal{P}_\omega(A \times \mathsf{Id})\colon \mathsf{Set} \to \mathsf{Set}$. The *invariants* of interest are coalgebras in a category of predicates.

3.3 Liftings and coinduction in a fibration

We have established coinduction as the principle of finality in a category of coalgebras. In Section 3.1 we have seen how to instantiate this to a setting where coalgebras model the systems of interest, yielding a canonical way of assigning behaviour and equivalence to a coalgebra. In this setting, coinduction provides a systematic account of bisimilarity and behavioural equivalence for all systems of the given type. On the other hand, in Section 3.2 we have seen how a different instantiation of coinduction yields the classical lattice-theoretic account, which is very flexible and allows to define many other predicates than bisimilarity, but is mainly suitable to define predicates on a single system. Here, bisimilarity and other predicates can be seen as objects that live in a category of predicates.

A very general and systematic approach for studying coinductive predicates on coalgebras can be achieved if the coalgebras of interest live in the base category of a *fibration*. This provides a means to speak about *properties* or *predicates* on coalgebras of interest. In this setting, invariants and coinductive predicates on a given coalgebra, are themselves coalgebras in a category of predicates, similar to the situation in the previous section. The functor on these predicates is defined in a uniform manner, based on a *lifting* of the behaviour functor.

This fibrational approach to coinductive predicates for coalgebras was proposed in [HJ98], and further developed in [HCKJ13] (as well as [AGJJ12, GJF13]). Below, we first list the necessary definitions related to fibrations (Section 3.3.1), and then describe the fibrational approach to coinductive predicates (Section 3.3.2). All of the examples in this thesis are based on two fibrations, described in Example 3.3.1 and Example 3.3.2. Of the remaining chapters in this thesis, the material in the current section is only necessary to understand Chapter 5.

3.3.1 Fibrations

We refer to [Jac99] for more information on fibrations, and recall only a few basic definitions and results.

A functor $p\colon \mathcal{E} \to \mathcal{A}$ is called a *fibration* when for every morphism $f\colon X \to Y$ in $\mathcal{A}$ and every R in $\mathcal{E}$ with $p(R) = Y$ there exists an object $f^*(R)$ with $p(f^*(R)) = X$ and a morphism $\widetilde{f}_R\colon f^*(R) \to R$ such that $p(\widetilde{f}_R) = f$ and $\widetilde{f}_R$ is *Cartesian*, which means that the following universal property holds: for all morphisms $g\colon Z \to X$ in $\mathcal{A}$ and $u\colon Q \to R$ in $\mathcal{E}$ sitting above $f \circ g$ (i.e., $p(u) = f \circ g$) there is a unique morphism $v\colon Q \to f^*(R)$ such that $u = \widetilde{f}_R \circ v$ and $p(v) = g$.

$$\begin{tikzcd}
Q \arrow[dr, dashed, "v"'] \arrow[drr, "u"] & & \\
& f^*(R) \arrow[r, "\widetilde{f}_R"'] & R
\end{tikzcd}$$

$$\begin{tikzcd}
Z \arrow[dr, "g"'] \arrow[drr, "f \circ g"] & & \\
& X \arrow[r, "f"'] & Y
\end{tikzcd}$$

We shall often use a special case of the universal property of $\widetilde{f}_R$ where $p(Q) = X$. Then for any $u\colon Q \to R$ sitting above f there exists a unique $v\colon Q \to f^*(R)$ above id_X such that $\widetilde{f}_R \circ v = u$:

$$\begin{tikzcd}
Q \arrow[d, dashed, "v"'] \arrow[dr, "u"] & \\
f^*(R) \arrow[r, "\widetilde{f}_R"'] & R \\
X \arrow[r, "f"] & Y
\end{tikzcd}$$

Given a fibration $p\colon \mathcal{E} \to \mathcal{A}$, we call $\mathcal{E}$ the *total category*, and $\mathcal{A}$ the *base category*. The *fibre* above an object X in $\mathcal{A}$, denoted by $\mathcal{E}_X$, is the subcategory of $\mathcal{E}$ with objects mapped by p to X and arrows mapped to the identity on X. We give a few examples of fibrations below; see [Jac99] for many more.

A morphism $\widetilde{f}$ as above is called a *(p)-Cartesian lifting* of f, and is unique up to isomorphism. If we make a choice of Cartesian liftings, the association $R \mapsto f^*(R)$ gives rise to the *reindexing functor* $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$. On a morphism $h\colon R \to S$ in $\mathcal{E}_Y$, it is defined using the universal property of the Cartesian lifting $\widetilde{f}_S$:

$$\begin{tikzcd}
f^*(R) \arrow[r, "\widetilde{f}_R"] \arrow[d, dashed, "f^*(h)"'] & R \arrow[d, "h"] \\
f^*(S) \arrow[r, "\widetilde{f}_S"'] & S
\end{tikzcd}$$

Given morphisms $f\colon X \to Y$ and $g\colon Y \to Z$ in $\mathcal{A}$, there is a natural isomorphism $(g \circ f)^* \cong f^* \circ g^*$ between reindexing functors.

A functor $p\colon \mathcal{E} \to \mathcal{A}$ is called a *bifibration* if both p and $p^{op}\colon \mathcal{E}^{op} \to \mathcal{A}^{op}$ are fibrations. Equivalently [Jac99, Lemma 9.1.2], p is a bifibration if each reindexing functor $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$ has a left adjoint $\coprod_f$:

$$\begin{tikzcd}
\mathcal{E}_X \arrow[r, bend left, "\coprod_f"] \arrow[r, phantom, "\perp"] & \mathcal{E}_Y \arrow[l, bend left, "f^*"]
\end{tikzcd}$$

We call $\coprod_f$ the *direct image* along f. This choice becomes more clear in the examples below.

For a fibration $p\colon \mathcal{E} \to \mathcal{A}$ we say that p has *fibred finite (co)products* if each fibre has finite (co)products, preserved by reindexing functors. If p is a bifibration with fibred finite products and coproducts, and $\mathcal{A}$ has finite products and coproducts, then the total category $\mathcal{E}$ also has finite products and coproducts, strictly preserved by p [Jac99, Example 9.2.5]. All bifibrations considered in this thesis are assumed to have this structure.

Example 3.3.1 (The predicate bifibration)**.** Let Pred be the category whose objects are pairs of sets (P, X) with $P \subseteq X$ and morphisms $f\colon (P, X) \to (Q, Y)$ are maps $f\colon X \to Y$ so that $f(P) \subseteq Q$. The functor $p\colon \mathsf{Pred} \to \mathsf{Set}$ mapping (P, X) to X is a fibration. The fibre Pred_X above X is the complete lattice of subsets of X ordered by inclusion. For any map $f\colon X \to Y$ in Set the reindexing functor $f^*\colon \mathsf{Pred}_Y \to \mathsf{Pred}_X$ maps (Q, Y) to $(f^{-1}(Q), X)$. Products and coproducts in a fibre Pred_X correspond to intersection and union, respectively. Products and coproducts in the total category $\mathcal{E}$ are simply computed as in Set. The functor f^* has a left adjoint $\coprod_f$ mapping (P, X) to the direct image $(f(P), Y)$.

We note that predicates can alternatively be seen as functions $X \to 2$. Reindexing along a function f then simply becomes precomposition with f.

Example 3.3.2 (The relation bifibration)**.** Similarly, we can consider the category Rel whose objects are pairs of sets (R, X) with $R \subseteq X \times X$ and morphisms $f\colon (R, X) \to (S, Y)$ are maps $f\colon X \to Y$ such that $(f \times f)(R) \subseteq S$. The functor $p\colon \mathsf{Rel} \to \mathsf{Set}$ mapping (R, X) to X is a fibration. The fibre Rel_X above X is the complete lattice of relations on X ordered by inclusion. For $f\colon X \to Y$ in Set the reindexing functor $f^*\colon \mathsf{Rel}_Y \to \mathsf{Rel}_X$ maps (R, Y) to $((f \times f)^{-1}(R), X)$. Its left adjoint $\coprod_f$ is given by direct image, that is, $\coprod_f(R, X) = ((f \times f)(R), Y)$.

Given fibrations $p\colon \mathcal{E} \to \mathcal{A}$ and $p'\colon \mathcal{E}' \to \mathcal{A}'$ and a functor $B\colon \mathcal{A} \to \mathcal{A}'$, we call $\overline{B}\colon \mathcal{E} \to \mathcal{E}'$ a *lifting* of B if the following diagram commutes:

$$\begin{tikzcd} \mathcal{E} \arrow[r, "\overline{B}"] \arrow[d, "p"'] & \mathcal{E}' \arrow[d, "p'"] \\ \mathcal{A} \arrow[r, "B"'] & \mathcal{A}' \end{tikzcd} \tag{3.5}$$

Such a lifting $\overline{B}$ restricts to a functor $\overline{B}_X\colon \mathcal{E}_X \to \mathcal{E}'_{BX}$ between fibres, for any X in $\mathcal{A}$. We sometimes omit the subscript X when it is clear from the context. A lifting $(\overline{B}, B)$ is a *fibration map* if it maps Cartesian morphisms to Cartesian morphisms. This means that there is an isomorphism

$$(Bf)^* \circ \overline{B} \cong \overline{B} \circ f^* \tag{3.6}$$

for any $\mathcal{A}$-morphism f. An important example for this thesis is the canonical relation lifting $\mathsf{Rel}(B)$, which is a fibration map whenever B preserves weak pullbacks.

Lemma 3.3.3. *For any $B\colon \mathsf{Set} \to \mathsf{Set}$, the lifting $(\mathsf{Rel}(B), B)$ is a fibration map (from the relation fibration to itself) if B preserves weak pullbacks.*

The isomorphism (3.6) means that $\mathsf{Rel}(B)$ preserves inverse images if B preserves weak pullbacks, that is, in that case the inclusion $\mathsf{Rel}(B)((f \times f)^{-1}(S)) \subseteq (Bf \times Bf)^{-1}(\mathsf{Rel}(B)(S))$ is an equality (see Lemma 3.2.4). The inclusion holds for any functor B; it is a special case of the following lemma.

Lemma 3.3.4. *Let $p\colon \mathcal{E} \to \mathcal{A}$ and $p'\colon \mathcal{E}' \to \mathcal{A}'$ be fibrations, and assume $\overline{B}\colon \mathcal{E} \to \mathcal{E}'$ is a lifting of some functor $B\colon \mathcal{A} \to \mathcal{A}'$. For any morphism $f\colon X \to Y$ in $\mathcal{A}$ there is a natural transformation*

$$\theta\colon \overline{B}_X \circ f^* \Rightarrow (Bf)^* \circ \overline{B}_Y : \mathcal{E}_Y \to \mathcal{E}'_{BX}\,.$$

If p and p' are bifibrations then there is another natural transformation

$$\theta'\colon \textstyle\coprod_{Bf} \circ \overline{B}_X \Rightarrow \overline{B}_Y \circ \coprod_f : \mathcal{E}_X \to \mathcal{E}'_{BY}\,.$$

Proof. To define θ_R on an object R in $\mathcal{E}_Y$, we apply $\overline{B}$ to the p-Cartesian lifting $\widetilde{f}_R\colon f^*(R) \to R$ and use the universal property of the p'-Cartesian lifting $\widetilde{(Bf)}_{\overline{B}R}$:

$$\begin{tikzcd}
\overline{B}(f^*(R)) \arrow[d, dashed, "\theta_R"'] \arrow[dr, "\overline{B}(\widetilde{f}_R)"] & \\
(Bf)^*(\overline{B}(R)) \arrow[r, "\widetilde{(Bf)}_{\overline{B}(R)}"'] & \overline{B}(R)
\end{tikzcd}$$

Naturality follows from the universal property of $\widetilde{(Bf)}_{\overline{B}R}$ and the definition of reindexing functors.

The natural transformation θ' can be defined as follows:

$$\begin{tikzcd}
\coprod_{Bf} \circ \overline{B}_X \arrow[r, Rightarrow] & \coprod_{Bf} \circ \overline{B}_X \circ f^* \circ \coprod_f \arrow[d, Rightarrow, "\coprod_{Bf}\theta\coprod_f"] & \\
& \coprod_{Bf} \circ (Bf)^* \circ \overline{B}_Y \circ \coprod_f \arrow[r, Rightarrow] & \overline{B}_Y \circ \coprod_f
\end{tikzcd}$$

using the unit of the adjunction $\coprod_f \dashv f^*$ and the counit of the adjunction $\coprod_{Bf} \dashv (Bf)^*$. (The way we obtain θ' from θ is an instance of a more general construction: θ' is called the *(adjoint) mate* of θ.) $\square$

3.3.2 Coinductive predicates in a fibration

Let $p\colon \mathcal{E} \to \mathcal{A}$ be a fibration, and let $B\colon \mathcal{A} \to \mathcal{A}$ be a functor whose coalgebras model the systems of interest. We show how to define functors on the fibre above the carrier of a B-coalgebra, such that the coalgebras for those functors are the invariants that model coinductive properties of the B-coalgebra in the base category. Following [HJ98], we then say a *coinductive predicate* is a *final coalgebra in a fibre*.

The crucial observation of this approach to coinductive predicates, is that we can uniformly define a functor $\mathcal{E}_X \to \mathcal{E}_X$ for any B-coalgebra $\delta\colon X \to BX$, from a given lifting $\overline{B}\colon \mathcal{E} \to \mathcal{E}$ of B. For a coalgebra $\delta\colon X \to BX$ it is defined as follows:

$$\mathcal{E}_X \xrightarrow{\overline{B}_X} \mathcal{E}_{BX} \xrightarrow{\delta^*} \mathcal{E}_X$$

A coalgebra $R \to \delta^* \circ \overline{B}_X(R)$ is called a $\delta^* \circ \overline{B}_X$*-invariant*; sometimes we shall refer only to the carrier R as an invariant and leave the transition structure implicit. The *final* $\delta^* \circ \overline{B}_X$*-coalgebra* (if it exists) can be seen as the coinductive predicate determined by $\overline{B}$ on the coalgebra δ. Finality of this coinductive predicate amounts to a proof principle: any invariant has a morphism to the coinductive predicate. For instance, if the state space X is a set and $\mathcal{E}_X$ is the lattice of predicates, this principle means that the carrier of any invariant is contained in the coinductive predicate. We refer to [HCKJ13] for more details on the existence of final coalgebras in a fibre.

Example 3.3.5. Recall from Example 3.2.6 the functor F whose final coalgebra is the divergence predicate on some coalgebra for the functor $BX = \mathcal{P}_\omega(A \times X)$. We define a lifting $\overline{B}\colon \mathsf{Pred} \to \mathsf{Pred}$ of B:

$$\overline{B}_X(P \subseteq X) = \{S \subseteq \mathcal{P}_\omega(A \times X) \mid \exists y \in P.(\tau, y) \in S\}$$

Then, given any $\delta\colon X \to BX$, we consider the composition

$$\mathsf{Pred}_X \xrightarrow{\overline{B}_X} \mathsf{Pred}_{BX} \xrightarrow{\delta^*} \mathsf{Pred}_X$$

where δ^* is the reindexing functor, i.e., inverse image along δ. The functor $\delta^* \circ \overline{B}_X$ now coincides with F from Example 3.2.6. Here it is defined uniformly on any B-coalgebra, based on a lifting of B that does not mention any concrete transition system.

Example 3.3.6. The functor $\mathsf{b}_\delta\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$, defined for a given coalgebra $\delta\colon X \to BX$ using relation lifting (see Section 3.2.1), decomposes as

$$\mathsf{Rel}_X \xrightarrow{\mathsf{Rel}(B)_X} \mathsf{Rel}_{BX} \xrightarrow{\delta^*} \mathsf{Rel}_X$$

A $\delta^* \circ \mathsf{Rel}(B)_X$-invariant is simply a b_δ-invariant. Equivalently, it is a bisimulation (Lemma 3.2.3).

Given a lifting $\overline{B}$ of B, we thus have a way of defining a functor on the fibre $\mathcal{E}_X$ above the carrier of any B-coalgebra. We now emphasize that this *uniformly* defines a predicate on B-coalgebras, by showing that coalgebra homomorphisms preserve invariants (and also reflect them, under a certain condition). The second item appears as Proposition 3.11 in [HCKJ13], with a proof in the appendix.

Proposition 3.3.7. *Let $p\colon \mathcal{E} \to \mathcal{A}$ be a bifibration, $\overline{B}\colon \mathcal{E} \to \mathcal{E}$ a lifting of a functor $B\colon \mathcal{A} \to \mathcal{A}$ and let $h\colon X \to Y$ be a coalgebra morphism from $\delta\colon X \to BX$ to $\vartheta\colon Y \to BY$.*

- *If R is a $\delta^* \circ \overline{B}_X$-invariant, then $\coprod_h(R)$ is a $\vartheta^* \circ \overline{B}_Y$-invariant.*
- *If S is a $\vartheta^* \circ \overline{B}_Y$-invariant and $(\overline{B}, B)$ is a fibration map, then $h^*(S)$ is a $\delta^* \circ \overline{B}_X$-invariant.*

Proof. Since h is a coalgebra homomorphism, we have $Bh \circ \delta = \vartheta \circ h$. Thus

$$\delta^* \circ (Bh)^* \cong (Bh \circ \delta)^* = (\vartheta \circ h)^* \cong h^* \circ \vartheta^* \,. \tag{3.7}$$

Using the unit of the adjunction $\coprod_{Bh} \dashv (Bh)^*$ and the counit of $\coprod_h \dashv h^*$ we construct the mate of the above natural transformation (read from left to right):

$$\textstyle\coprod_h \circ\, \delta^* \Longrightarrow \coprod_h \circ\, \delta^* \circ (Bh)^* \circ \coprod_{Bh} \Longrightarrow \coprod_h \circ\, h^* \circ \vartheta^* \circ \coprod_{Bh} \Longrightarrow \vartheta^* \circ \coprod_{Bh} .$$

We can use this to construct a natural transformation

$$\textstyle\gamma\colon \coprod_h \circ\, \delta^* \circ \overline{B}_X \Longrightarrow \vartheta^* \circ \coprod_{Bh} \circ\, \overline{B}_X \Longrightarrow \vartheta^* \circ \overline{B}_Y \circ \coprod_h$$

where the second part is given by Lemma 3.3.4. Then any $\delta^* \circ \overline{B}$-invariant, that is, a coalgebra $R \to \delta^* \circ \overline{B}_X(R)$ in $\mathcal{E}_X$, yields a coalgebra (invariant) $\coprod_h(R) \to \vartheta^* \circ \overline{B}_Y \circ \coprod_h(R)$ in $\mathcal{E}_Y$, simply by applying $\coprod_h$ and the natural transformation γ.

For the second item, we construct a natural isomorphism

$$h^* \circ \vartheta^* \circ \overline{B}_Y \cong \delta^* \circ (Bh)^* \circ \overline{B}_Y \cong \delta^* \circ \overline{B}_X \circ h^*$$

using (3.7) and the fact that $(\overline{B}, B)$ is a fibration map. Then, given an invariant $S \to \vartheta^* \circ \overline{B}_Y(S)$, we apply the isomorphism to get the invariant $h^*(S) \to h^* \circ \vartheta^* \circ \overline{B}_Y(S) \cong \delta^* \circ \overline{B}_X \circ h^*(S)$. ☐

As stated in Lemma 3.2.3, a relation R is a bisimulation on a coalgebra δ precisely if it is a $\delta^* \circ \mathsf{Rel}(B)$-invariant. Thus, the first item of the above Proposition 3.3.7 is a generalization of the fact that coalgebra homomorphisms preserve bisimilarity (Lemma 3.1.3), for $\overline{B}$ instantiated to the canonical relation lifting $\mathsf{Rel}(B)$ (Lemma 3.1.3 mentions two homomorphisms rather than one; this can also be accommodated in the current setting by choosing a slightly different fibration). Moreover, if B preserves weak pullbacks, then $(\mathsf{Rel}(B), B)$ is a fibration map (Lemma 3.3.3). Hence, a special case of the second item is that bisimulations are preserved by inverse image along coalgebra homomorphisms, whenever the functor B preserves weak pullbacks.

3.4 Algebras

In this thesis, algebras play an important role to model coalgebras whose carrier has algebraic structure; for example, the set of closed terms over some signature.

Other examples include automata over sets or linear combinations of states, which arise in determinization constructions.

An *algebra* for a functor $T\colon \mathcal{C} \to \mathcal{C}$, or *$T$-algebra*, is a pair (X, α) where X is an object in $\mathcal{C}$ and $\alpha\colon TX \to X$ is a morphism. We call X the carrier and α the algebra structure. An *(algebra) homomorphism* from $\alpha\colon TX \to X$ to $\beta\colon TY \to Y$ is a function $h\colon X \to Y$ such that $h \circ \alpha = \beta \circ Th$. The category of algebras and their homomorphisms is denoted by T-alg.

An *initial T-algebra* is an initial object in the category T-alg. Thus, given an initial T-algebra (A, κ) there exists, for each T-algebra (X, α) a unique algebra homomorphism from (A, κ) to (X, α). We call such a morphism the *inductive extension* of (X, α). Similar to the case of final coalgebras, initial algebras exist under mild conditions on the functor.

A *signature* Σ is a (possibly infinite) set of operator names $\sigma \in \Sigma$ with (finite) arities $|\sigma| \in \mathbb{N}$. Equivalently, it is a polynomial functor on Set:

$$\begin{aligned} \Sigma X &= \coprod_{\sigma \in \Sigma} \{\sigma\} \times X^{|\sigma|} \cong \{\sigma(x_1, \ldots, x_{|\sigma|}) \mid \sigma \in \Sigma \text{ and } \forall i.\, x_i \in X\} \\ (\Sigma(f\colon X \to Y))(\sigma(x_1, \ldots, x_n)) &= \sigma(f(x_1), \ldots, f(x_n)) \end{aligned} \tag{3.8}$$

A Σ-algebra coincides with the standard notion of an interpretation of the signature Σ: a set X together with a function of type $X^{|\sigma|} \to X$ for every operator σ (see also Section 2.3). The carrier of the initial Σ-algebra is given by the set of all *closed terms* over the signature.

3.4.1 Monads

A *monad* is a triple $\mathcal{T} = (T, \eta, \mu)$ where $T\colon \mathcal{C} \to \mathcal{C}$ is a functor, and $\eta\colon \mathsf{Id} \Rightarrow T$ and $\mu\colon TT \Rightarrow T$ are natural transformations called *unit* and *multiplication* respectively, such that the following diagrams commute:

$$\begin{tikzcd} T \arrow[r, Rightarrow, "\eta T"] \arrow[dr, equal] & TT \arrow[d, Rightarrow, "\mu"] & T \arrow[l, Rightarrow, "T\eta"'] \arrow[dl, equal] \\ & T & \end{tikzcd} \qquad \begin{tikzcd} TTT \arrow[r, Rightarrow, "T\mu"] \arrow[d, Rightarrow, "\mu T"'] & TT \arrow[d, Rightarrow, "\mu"] \\ TT \arrow[r, Rightarrow, "\mu"'] & T \end{tikzcd} \tag{3.9}$$

An Eilenberg-Moore algebra for $\mathcal{T}$ (or *$\mathcal{T}$-algebra*, or *algebra for the monad $\mathcal{T}$*) is a T-algebra $\alpha\colon TX \to X$ such that the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\alpha"] & TTX \arrow[l, "T\alpha"'] \arrow[d, "\mu_X"] \\ & X & TX \arrow[l, "\alpha"] \end{tikzcd}$$

A *$\mathcal{T}$-algebra homomorphism* is simply a T-algebra homomorphism. We denote the category of $\mathcal{T}$-algebras and their homomorphisms by $\mathcal{T}$-Alg, and the associated forgetful functor by $U\colon \mathcal{T}\text{-}\mathsf{Alg} \to \mathcal{C}$.

Throughout this thesis we often use $\mathcal{T}$ to denote a monad and T to denote a functor. Accordingly, a $\mathcal{T}$-algebra is an (Eilenberg-Moore) algebra for a *monad*, whereas a T-algebra is an algebra for a *functor*.

Given any $\mathcal{C}$-object X, the algebra (TX, μ_X) satisfies a universal property: for any $\mathcal{T}$-algebra (A, α) and any arrow $f\colon X \to A$, there is a unique algebra homomorphism $f^\sharp\colon TX \to A$ such that $f^\sharp \circ \eta_X = f$, given by $f^\sharp = \alpha \circ Tf$.

Let (T, η, μ) and (K, θ, ν) be monads. A *monad morphism* is a natural transformation $\sigma\colon T \Rightarrow K$ such that the following diagram commutes:

$$\begin{tikzcd} \mathsf{Id} \arrow[r, Rightarrow, "\eta"] \arrow[dr, Rightarrow, "\theta"'] & T \arrow[d, Rightarrow, "\sigma"] & TT \arrow[l, Rightarrow, "\mu"'] \arrow[d, Rightarrow, "\sigma\sigma"] \\ & K & KK \arrow[l, Rightarrow, "\nu"] \end{tikzcd} \tag{3.10}$$

where $\sigma\sigma = K\sigma \circ \sigma T = \sigma K \circ T\sigma$.

Example 3.4.1. We list a few examples of monads.

1. The powerset functor $\mathcal{P}$ is a monad, with unit $\eta\colon \mathsf{Id} \Rightarrow \mathcal{P}$ and multiplication $\mu\colon \mathcal{P}\mathcal{P} \Rightarrow \mathcal{P}$ given by:

$$\eta_X(x) = \{x\} \quad \text{and} \quad \mu_X(S) = \bigcup_{U \in S} U\,.$$

   The finite powerset functor $\mathcal{P}_\omega$ extends to a monad in a similar way.

2. Given a semiring $\mathbb{S}$, the functor $\mathcal{M}$ extends to a monad, by taking

$$\eta_X(x)(y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \qquad \mu_X(\varphi)(x) = \sum_{\psi \in \mathbb{S}^X} \varphi(\psi) \cdot \psi(x)$$

   The case of $\mathcal{P}_\omega$ is obtained by taking the Boolean semiring. Notice that μ is well-defined since its argument φ has finite support, by definition of $\mathcal{M}$.

3. Suppose Σ is a polynomial functor representing a signature (3.8). Consider the functor Σ^*, which maps a set X to the set of terms over Σ with variables in X, as given by the grammar $t ::= x \mid \sigma(t_1, \ldots, t_{|\sigma|})$, where x ranges over X and σ ranges over the operator names. Given $f\colon X \to Y$, the function $\Sigma^* f\colon \Sigma^* X \to \Sigma^* Y$ is defined by substitution. The functor Σ^* extends to a monad, where the multiplication μ glues terms over terms, and the unit η interprets a variable as a term. This monad is defined properly below; it is called the *free monad* for Σ.

Let $\Sigma\colon \mathcal{C} \to \mathcal{C}$ be an arbitrary functor. A *free Σ-algebra* for a $\mathcal{C}$-object X is an initial $\Sigma + X$-algebra. The existence of free algebras for every object X amounts to the existence of a left adjoint F to the forgetful functor $U\colon \Sigma\text{-}\mathsf{alg} \to \mathcal{C}$. It is a standard fact in category theory that an adjunction yields a monad; we spell out some of the details. Suppose a left adjoint F to U exists, let $\Sigma^* = UF\colon \mathcal{C} \to \mathcal{C}$ and let $\eta\colon \mathsf{Id} \Rightarrow \Sigma^*$ be the unit of the adjunction. The functor F induces a natural transformation $\kappa\colon \Sigma\Sigma^* \Rightarrow \Sigma^*$ such that (the copairing of)

$$\Sigma\Sigma^* X \xrightarrow{\kappa_X} \Sigma^* X \xleftarrow{\eta_X} X \tag{3.11}$$

is the free Σ-algebra for X. This means that for any Σ-algebra (Y, α) and any $f\colon X \to Y$ there exists a unique homomorphism $f^\sharp$ as in the following diagram:

$$\begin{tikzcd}
\Sigma\Sigma^* X \arrow[r, "\Sigma f^\sharp"] \arrow[d, "\kappa_X"'] & \Sigma Y \arrow[d, "\alpha"] \\
\Sigma^* X \arrow[r, dashed, "f^\sharp"] & Y \\
X \arrow[u, "\eta_X"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Then the *free monad* for Σ is defined as (Σ^*, η, μ) where η is the unit of the adjunction, and μ is defined on a component X as the unique morphism $\mu_X\colon \Sigma^*\Sigma^* X \to \Sigma^* X$ such that $\mu_X \circ \eta_{\Sigma^* X} = \mathsf{id}$.

Example 3.4.2. Suppose Σ^* is the (underlying functor of the) free monad for Σ arising from a signature, as described in Example 3.4.1 (3). The fact that $\Sigma^* X$ is a free Σ-algebra amounts to the following: given any Σ-algebra A, there is a one-to-one correspondence between maps $f\colon X \to A$ and algebra homomorphisms $f^\sharp\colon \Sigma^* X \to A$. Here f can be viewed as a variable assignment, and $f^\sharp$ as its inductive extension to terms.

Suppose (Σ^*, η, μ) is the free monad for a functor Σ. Any Σ-algebra $\alpha\colon \Sigma X \to X$ then yields an Eilenberg-Moore algebra $\widehat{\alpha}\colon \Sigma^* X \to X$, defined by the unique extension of id_X to an algebra morphism from $\Sigma^* X$ to X. In fact, this construction yields an isomorphism between the category Σ-alg of algebras for the functor Σ and the category Σ^*-Alg of algebras for the free monad Σ^*.

3.5 Bialgebras and distributive laws

Bialgebras consist of an algebra and a coalgebra structure over a common carrier. The interaction between algebra and coalgebra can be captured by *distributive laws*. These provide enough structure to study operational semantics, determinization and recursive equations in a systematic manner; see [TP97, Bar04, Kli11, JSS12] for more information.

Let $T, B\colon \mathcal{C} \to \mathcal{C}$ be functors. A *distributive law* of T over B is a natural transformation $\lambda\colon TB \Rightarrow BT$. This is the simplest type of distributive law, and we sometimes refer to it as a distributive law between functors. Given such a λ, a

λ-*bialgebra* is a triple (X, α, δ) so that $\alpha \colon TX \to X$ is a T-algebra, $\delta \colon X \to BX$ is a B-coalgebra and the following diagram commutes:

$$\begin{tikzcd} TX \arrow[r, "\alpha"] \arrow[d, "T\delta"'] & X \arrow[r, "\delta"] & BX \\ TBX \arrow[rr, "\lambda_X"'] & & BTX \arrow[u, "B\alpha"'] \end{tikzcd} \tag{3.12}$$

A λ-bialgebra homomorphism is a map that is both an algebra and a coalgebra homomorphism. Any distributive law defines *liftings* of T and B:

$$\begin{tikzcd} B\text{-coalg} \arrow[r, "\overline{T}"] \arrow[d] & B\text{-coalg} \arrow[d] \\ \mathcal{C} \arrow[r, "T"'] & \mathcal{C} \end{tikzcd} \qquad \begin{tikzcd} T\text{-alg} \arrow[r, "\overline{B}"] \arrow[d] & T\text{-alg} \arrow[d] \\ \mathcal{C} \arrow[r, "B"'] & \mathcal{C} \end{tikzcd}$$

defined on objects by

$$\overline{T}(X, \delta) = (TX, \lambda_X \circ T\delta) \qquad \overline{B}(X, \alpha) = (BX, B\alpha \circ \lambda_X)$$

Notice that (3.12) commutes iff δ is a $\overline{B}$-coalgebra with carrier (X, α) iff α is a $\overline{T}$-algebra with carrier (X, δ). Indeed, the category of λ-bialgebras is isomorphic to the category of $\overline{B}$-coalgebras and the category of $\overline{T}$-algebras.

If B has a final coalgebra (Z, ζ), we can use $\overline{T}$ and coinduction to construct a bialgebra on Z:

$$\begin{tikzcd} TZ \arrow[r, dashed, "\alpha"] \arrow[d, "T\zeta"'] & Z \arrow[dd, "\zeta"] \\ TBZ \arrow[d, "\lambda_Z"'] & \\ BTZ \arrow[r, "B\alpha"'] & BZ \end{tikzcd}$$

This bialgebra is *final* in the category of λ-bialgebras.

Lemma 3.5.1. *Let $\lambda \colon TB \Rightarrow BT$ be a distributive law (between functors). The final coalgebra (Z, ζ) lifts to a final λ-bialgebra.*

Similarly, if T has an initial algebra (A, κ) then we can lift it to an initial λ-bialgebra, see, e.g., [Kli11] for details. Instead of spelling this out, we will consider initial bialgebras and their properties for a more general type of distributive law in the next subsection.

3.5.1 Distributive laws of monads over (copointed) functors

Let $\mathcal{T} = (T, \eta, \mu)$ be a monad. A distributive law of $\mathcal{T}$ over B is a natural transformation $\lambda \colon TB \Rightarrow BT$ such that the following diagrams commute:

$$\begin{tikzcd} B \arrow[r, Rightarrow, "\eta B"] \arrow[dr, Rightarrow, "B\eta"'] & TB \arrow[d, Rightarrow, "\lambda"] \\ & BT \end{tikzcd} \qquad \begin{tikzcd} TTB \arrow[r, Rightarrow, "T\lambda"] \arrow[d, Rightarrow, "\mu B"'] & TBT \arrow[r, Rightarrow, "\lambda T"] & BTT \arrow[d, Rightarrow, "B\mu"] \\ TB \arrow[rr, Rightarrow, "\lambda"'] & & BT \end{tikzcd}$$

A distributive law λ as above induces a lifting $\overline{B} \colon \mathcal{T}\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg}$ of B to the category of Eilenberg-Moore algebras for the monad $\mathcal{T}$. In fact, there is a one-to-one correspondence between distributive laws λ as above and liftings of B to $\mathcal{T}$-Alg [Joh75, TP97].

Suppose λ is a distributive law of $\mathcal{T}$ over B. Any coalgebra $\delta \colon X \to BTX$ can then be extended to a homomorphism $\delta^\sharp \colon (TX, \mu_X) \to \overline{B}(TX, \mu_X)$ so that $\delta^\sharp \circ \eta_X = \delta$. Notice that $\delta^\sharp$ is defined by

$$TX \xrightarrow{T\delta} TBTX \xrightarrow{\lambda_{TX}} BTTX \xrightarrow{B\mu_X} BTX \tag{3.13}$$

This yields another lifting $\widehat{T} \colon TB\text{-}\mathsf{coalg} \to B\text{-}\mathsf{coalg}$ of T. Now, consider the coinductive extension below:

$$\begin{tikzcd} X \arrow[r, "\eta_x"] \arrow[d, "\delta"'] & TX \arrow[r, dashed, "h"] \arrow[dl, "\delta^\sharp"] & Z \arrow[d, "\zeta"] \\ BTX \arrow[rr, "Bh"'] & & BZ \end{tikzcd} \tag{3.14}$$

Since the final B-coalgebra lifts to a final $\overline{B}$-coalgebra (similar to Lemma 3.5.1), the coinductive extension h is an algebra homomorphism. This can be interpreted as stating that the semantics is compositional, in the sense that behavioural equivalence on $\delta^\sharp$ is a congruence.

The above use of distributive laws to turn TB-coalgebras into $\overline{B}$-coalgebras (and obtaining a semantics from the coinductive extension) is sometimes interpreted as a general way of solving corecursive equations (e.g., [Bar04, Jac06b]); it is also called the *generalized powerset construction* [SBBR13, JSS12].

Example 3.5.2 ([SBBR13, JSS12])**.** In Section 3.1.1 we have seen informally how to determinize non-deterministic automata. This construction arises from a distributive law $\lambda \colon \mathcal{P}_\omega(2 \times \mathsf{Id}^A) \Rightarrow 2 \times (\mathcal{P}-)^A$ of the powerset monad over the functor $2 \times \mathsf{Id}^A$, given by

$$\lambda_X(S) = \left(\bigvee_{(o,t)\in S} o, \lambda a. \bigcup_{(o,t)\in S} t(a) \right) .$$

Spelling out the details of the construction in Equation 3.13 yields the classical powerset construction, as in Section 3.1.1. The composition $h \circ \eta_X$ in (3.14) is the usual language semantics of non-deterministic automata (obtained via determinization).

Similarly, the determinization of weighted automata arises from a distributive law $\lambda\colon \mathcal{M}B \Rightarrow B\mathcal{M}$ where $BX = \mathbb{S} \times X^A$ and λ is defined by

$$\lambda_X \left(\sum r_i(o_i, t_i) \right) = \left(\sum r_i \cdot o_i, \lambda a. \sum r_i \cdot t_i(a) \right) .$$

The composition $h \circ \eta_X$ in (3.14) maps a state to the weighted language that it accepts, see [JSS12, BBB$^+$12].

There is yet another type of distributive law, which is particularly suitable for operational semantics, as we will see below. To define it we need the notion of a *copointed* functor, which is a pair (B, ϵ) where $B\colon \mathcal{C} \to \mathcal{C}$ is an endofunctor and $\epsilon\colon B \Rightarrow \mathsf{Id}$ a natural transformation. A coalgebra for a copointed functor (B, ϵ) is a B-coalgebra (X, δ) such that $\epsilon_X \circ \delta = \mathsf{id}$. We will frequently consider copointed functors $(B \times \mathsf{Id}, \pi_2)$; such a functor is called the *cofree* copointed functor for B. It is easy to see that B-coalgebras are in one-to-one correspondence to coalgebras for $(B \times \mathsf{Id}, \pi_2)$. Now, a distributive law of a monad (T, η, μ) over a copointed functor (B, ϵ) is a distributive law λ of (T, η, μ) over B such that, additionally, the axiom

$$\begin{tikzcd} TB \arrow[d, Rightarrow, "\lambda"'] \arrow[dr, Rightarrow, "T\epsilon"] & \\ BT \arrow[r, Rightarrow, "\epsilon T"'] & T \end{tikzcd}$$

is satisfied. (We note that this can be further generalized by considering distributive laws of monads over comonads; for a formal definition see, e.g., [Kli11].)

3.5.2 Abstract GSOS

In this section we consider *abstract GSOS*, which provides specification formats for defining operations on coalgebras, and allows to study operational semantics in a general fashion. It is a generalization of *GSOS*, which is a syntactic format for transition system specifications (see Example 3.5.4 below). An abstract GSOS specification of Σ over B is a natural transformation $\rho\colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$. The following result [TP97] states that distributive laws of monad over copointed functor can be presented by abstract GSOS specifications.

Lemma 3.5.3. *There is a one-to-one correspondence between abstract GSOS specifications ρ of Σ over B and distributive laws $\rho^\dagger$ of the free monad Σ^* over the cofree copointed functor $B \times \mathsf{Id}$.*

For a full proof, see [LPW04, Bar04]. Given ρ, $\rho^\dagger$ is defined on a component X using initiality:

$$\begin{tikzcd}
\Sigma\Sigma^*(BX \times X) \arrow[r, "\Sigma\rho^\dagger_X"] \arrow[dd, "\kappa_{BX\times X}"'] & \Sigma(B\Sigma^* X \times \Sigma^* X) \arrow[d, "\langle \rho_{\Sigma^* X}, \Sigma\pi_2 \rangle"] \\
& B\Sigma^*\Sigma^* X \times \Sigma\Sigma^* X \arrow[d, "B\mu_X \times \kappa_X"] \\
\Sigma^*(BX \times X) \arrow[r, "\rho^\dagger_X"] & B\Sigma^* X \times \Sigma^* X \\
BX \times X \arrow[u, "\eta_{BX\times X}"] \arrow[ur, "B\eta_X \times \eta_X"'] &
\end{tikzcd} \tag{3.15}$$

A *model* of ρ is a triple (X, α, δ) where $\alpha \colon \Sigma X \to X$ is a Σ-algebra and $\delta \colon X \to BX$ a B-coalgebra, such that the diagram

$$\begin{tikzcd}
\Sigma X \arrow[r, "\alpha"] \arrow[d, "\Sigma\langle \delta, \mathsf{id} \rangle"'] & X \arrow[r, "\delta"] & BX \\
\Sigma(BX \times X) \arrow[rr, "\rho_X"'] & & B\Sigma^* X \arrow[u, "B\widehat{\alpha}"']
\end{tikzcd}$$

commutes. There is a one-to-one correspondence between models for ρ and $\rho^\dagger$-bialgebras; more precisely, a triple (X, α, δ) is a model of ρ iff $(X, \widehat{\alpha}, \langle \delta, \mathsf{id} \rangle)$ is a $\rho^\dagger$-bialgebra. Based on this correspondence, it is easy to establish that behavioural equivalence on (the coalgebra part of) any ρ-model is a congruence. The ρ-model corresponding to the initial $\rho^\dagger$-bialgebra is sometimes referred to as the *operational model* of ρ. We consider a few examples of abstract GSOS for particular choices of the behaviour functor B; for many other instances of abstract GSOS, see (the references in) [Kli11].

Example 3.5.4. Abstract GSOS is a generalization of *GSOS*, a format for transition system specifications introduced in [BIM95]. Given a signature, a *GSOS* rule for an operator σ of arity n is of the form

$$\frac{\{x_{i_j} \xrightarrow{a_j} y_j\}_{j=1..m} \qquad \{x_{i_k} \stackrel{b_k}{\not\to}\}_{k=1..l}}{\sigma(x_1, \ldots, x_n) \xrightarrow{c} t} \tag{3.16}$$

where m is the number of positive premises, l is the number of negative premises, and $a_1, \ldots, a_m, b_1, \ldots, b_l, c \in A$ are labels. The variables $x_1, \ldots, x_n$, $y_1, \ldots, y_m$ are pairwise distinct, and t is a term over these variables.

GSOS rules for a signature Σ induce abstract GSOS specifications of Σ over the functor $BX = (\mathcal{P}_\omega X)^A$ of labelled transition systems. Conversely, every GSOS specification arises from an abstract GSOS specification. This correspondence was first observed in [TP97], and proved in detail in [Bar04]; see also [Kli11] for a detailed explanation. The unique ρ-model on the initial algebra corresponds to the supported model of a GSOS specification. The well-known fact that bisimilarity on the supported model of a GSOS specification is a congruence, thus follows from the abstract underlying theory of distributive laws.

A simple example of a GSOS specification is given by the parallel composition operation. Let $A = N \cup \overline{N} \cup \{\tau\}$ where N is a set of labels and $\overline{N} = \{\overline{a} \mid a \in N\}$; we let $\overline{\overline{a}} = a$. The parallel composition is then defined by the following rules:

$$\frac{x \xrightarrow{a} x'}{x|y \xrightarrow{a} x'|y} \qquad \frac{y \xrightarrow{a} y'}{x|y \xrightarrow{a} x|y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{\overline{a}} y'}{x|y \xrightarrow{\tau} x'|y'}$$

We define this as an abstract GSOS specification $\rho\colon \Sigma((\mathcal{P}_\omega -)^A \times \mathsf{Id}) \Rightarrow \mathcal{P}_\omega(\Sigma -)^A$, where $\Sigma X = X \times X$ (we model a binary operator). Then, on a component X, $\rho_X : ((\mathcal{P}_\omega X)^A \times X) \times ((\mathcal{P}_\omega X)^A \times X) \to (\mathcal{P}_\omega(\Sigma^* X))^A$ is given by

$$\begin{aligned}\rho(f,x,g,y)(\tau) = &\{(x'|y) \mid x' \in f(\tau)\} \cup \{(x|y') \mid y' \in g(\tau)\} \\ &\cup \{(x'|y') \mid x' \in f(a) \text{ and } y' \in g(\overline{a})\}\end{aligned}$$

and for any $a \in A$ with $a \neq \tau$:

$$\rho(f,x,g,y)(a) = \{(x'|y) \mid x' \in f(a)\} \cup \{(x|y') \mid y' \in g(a)\}\,.$$

Notice that the carrier of the operational model of ρ is empty; this is because we did not add any constants to the signature and the specification. If we do, then the operational model will be a transition system whose states are the terms built from these constants and the parallel operator, and where the behaviour of a term $p|q$ is dictated by the GSOS rules above.

Example 3.5.5. Behavioural differential equations for streams, such as those defined in Section 3.1.1, can be presented by abstract GSOS specifications of Σ over $BX = A \times X$, where Σ is the signature functor representing the syntax. The precise format and definitions are well explained in [HKR14]. For example, to define the operations of sum, convolution product and the constants $[r]$ we take $\Sigma X = X \times X + X \times X + \mathbb{R}$ and define $\rho\colon \Sigma((\mathbb{R} \times \mathsf{Id}) \times \mathsf{Id}) \Rightarrow \mathbb{R} \times \Sigma^*$ by cases:

$$\begin{aligned}\rho_X^{[r]} &= (r, [0]) \\ \rho_X^{+}((a,x',x),(b,y',y)) &= (a+b, x'+y') \\ \rho_X^{\times}((a,x',x),(b,y',y)) &= (a \cdot b, (x' \times y) + (a \times y'))\end{aligned}$$

The shuffle product is also easily defined this way. The shuffle inverse is a bit more problematic, since it is not always defined. One ad-hoc way of solving this is by just assigning it some fixed constant value in those cases.

The operational model of ρ then consists of the closed terms over Σ, and its coalgebra structure is defined by induction according to ρ. The coinductive extension yields the semantics, and this is compositional with respect to the algebraic structure induced on the final coalgebra.

Similarly, the format of behavioural differential equations for deterministic automata presented in Definition 2.3.3 induces GSOS specifications for the functor $BX = 2 \times X^A$. We did not formally prove the converse, i.e., that every such specification arises from an abstract GSOS specification, although this seems rather likely. The format of behavioural differential equations in Definition 2.3.3 thus consititutes a concrete, syntactic presentation of GSOS specifications for deterministic automata.

The GSOS format for streams and the one for deterministic automata give rise to specific types of behavioural differential equations. BDEs can be defined more generally, for instance involving second derivative, which can not be expressed in these formats. The advantage of (abstract) GSOS is that it is quite expressive and covers most examples encountered in the literature, while these specifications are still well-behaved: they give rise to a compositional semantics, and have distributive laws and bialgebras as a solid underlying mathematical theory.

Chapter 4

Bisimulation up-to

The theory of coalgebras provides bisimilarity as a fundamental notion of equivalence between systems. In this chapter, we introduce enhancements of the proof technique for bisimilarity at this abstract level, by providing a general account of bisimulation up-to techniques for arbitrary coalgebras. We show the use of these up-to techniques by instantiating them to (non)deterministic automata, weighted automata and stream systems.

The main challenge is to provide generic up-to techniques that are *sound*, meaning that they can safely be used for proving bisimilarity. One difficulty is that sound functions do not compose, thus obstructing a modular approach to proving the soundness of up-to techniques in terms of their basic constituents. This issue was addressed by Sangiorgi [San98] and Pous [Pou07, PS12], who introduced up-to techniques in the setting of coinduction in a lattice. The central feature in the framework of [Pou07] is the notion of *compatible* functions, defining a class of sound enhancements that is closed under composition. By instantiating this framework to coalgebraic bisimilarity, we obtain compatibility as a *modular* way of proving soundness.

The first up-to technique that appeared in the literature is Milner's *bisimulation up to bisimilarity* [Mil83]. We show that this is compatible whenever the behaviour functor under consideration preserves weak pullbacks. The *equivalence closure* is also useful as an up-to technique, and its compatibility depends on weak pullback preservation as well. In the presence of algebraic structure on the state space, the notion of *bisimulation up to context* becomes relevant; we show that this is compatible whenever the coalgebraic and algebraic structure together form a λ-bialgebra. This implies, for instance, that bisimulation up to context is sound on the supported model of any GSOS specification, which is more general than the De Simone format considered in [San98]. Moreover, our compatibility results can be combined; for instance, the compatibility of the *congruence closure* follows from that of the equivalence and contextual closure. The soundness of bisimulation up-to techniques for languages, as considered in Chapter 2, is an immediate consequence.

If the behaviour functor under consideration does not preserve weak pullbacks, then one may be interested in behavioural equivalence rather than bisimilarity (if the functor preserves weak pullbacks then these two coincide, see Section 3.1). This is the case, for example, for certain weighted transition systems [GS01, Kli09, BBB+12] and for neighbourhood structures used in modal logic [HKP09]. We conclude this chapter with a treatment of up-to techniques for behavioural equivalence, and show in particular the compatibility of the contextual closure and the equivalence closure.

Throughout this chapter we only consider coalgebras in the category Set of sets and functions. Most of the technical results are a special case of more general results on coinductive up-to techniques, presented in Chapter 5 of this thesis. The current chapter explains the essentials of up-to techniques for the fundamental coinductive predicate of coalgebraic bisimilarity, requiring only basic knowledge of category theory.

Outline. The next section contains the definition of bisimulation up-to. The main instances of up-to techniques as well as a number of example proofs are in Section 4.2. Section 4.3 is a short overview of Pous's framework. This is instantiated in Section 4.4 to prove the main soundness results. Section 4.5 treats behavioural equivalence up-to. In Section 4.6 a short summary of the soundness results is provided.

4.1 Progression and bisimulation up-to

The definition of bisimulation up-to on labelled transition systems can be stated conveniently in terms of *progression* [PS12], which we generalize to a coalgebraic setting as follows.

Definition 4.1.1. For a coalgebra $\delta\colon X \to BX$ and relations $R, S \subseteq X \times X$, we say R *progresses to* S if there exists a function $\gamma\colon R \to BS$ making the following diagram commute:

$$\begin{tikzcd} X \arrow[d, "\delta"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & X \arrow[d, "\delta"] \\ BX & BS \arrow[l, "B\pi_1"] \arrow[r, "B\pi_2"'] & BX \end{tikzcd}$$

We recover the standard definition of a bisimulation on a single coalgebra (Section 3.1) by taking $R = S$, i.e., a relation R that progresses to itself. Progression allows to define bisimulation up-to, and the crucial associated notion of soundness.

Definition 4.1.2. Let $\delta\colon X \to BX$ be a coalgebra and $g\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ be a function. A relation R is a *bisimulation up to* g if R progresses to $g(R)$, i.e., if there is a function $\gamma\colon R \to B(g(R))$ making the following diagram commute:

$$\begin{tikzcd}
X \arrow[d, "\delta"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"] & X \arrow[d, "\delta"] \\
BX & B(g(R)) \arrow[l, "B\pi_1"] \arrow[r, "B\pi_2"'] & BX
\end{tikzcd}$$

We say that g is (δ)-*sound* if the following implication holds, for any $R \subseteq X \times X$:

$$\text{if } R \text{ is a bisimulation up to } g \text{ then } R \subseteq {\sim_\delta}\,,$$

that is, g is sound if every bisimulation up to g is contained in bisimilarity.

Informally, to check that R is a bisimulation up to g, the derivatives or next states need not be related by R again, but by $g(R)$. Depending on $g(R)$, which in most examples is a bigger relation than R, this is a weaker requirement than the usual conditions for showing R to be a bisimulation. However, we only obtain a valid proof principle for bisimilarity if g is sound: then, to prove that two states are bisimilar, it suffices to relate them by a bisimulation up to g. Therefore, our main aim is to find useful functions g that are sound.

Not every function is sound; for a simple example, take the function g that maps every relation R on X to the Cartesian product $X \times X$. Then, a relation R on the states of a transition system is a bisimulation up to g if for each $(x, y) \in R$ and each label a: there is x' such that $x \xrightarrow{a} x'$ if and only if there is y' such that $x \xrightarrow{a} y'$. Clearly, this g is not sound.

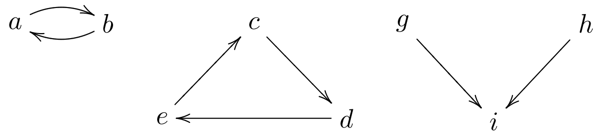
4.2 Examples

We introduce the most important instances of bisimulation up-to for a variety of systems. In each case, the up-to technique under consideration is sound (under certain assumptions), which follows from results in subsequent sections of this chapter. Thus, all of these examples can be seen as actual proofs of bisimilarity. More details on the types of coalgebras under consideration and their associated notions of bisimulation can be found in Example 3.1.1 and Example 3.1.2.

Bisimulation up to equivalence

Consider the function eq mapping a relation R to its equivalence closure $\mathsf{eq}(R)$. A bisimulation up to eq is also called a *bisimulation up to equivalence*.

Example 4.2.1. Given a coalgebra $\delta\colon X \to X + 1$, a relation R on X is a bisimulation up to equivalence if for all $(x, y) \in R$: either $\delta(x) = * = \delta(y)$, or $(\delta(x), \delta(y)) \in \mathsf{eq}(R)$. This is different than a bisimulation, which requires $(\delta(x), \delta(y)) \in R$ rather than $(\delta(x), \delta(y)) \in \mathsf{eq}(R)$ (Example 3.1.2). Consider the following coalgebras and relations:



$\{(a,b)\}$ $\{(c,d),(d,e)\}$ $\{(g,h)\}$

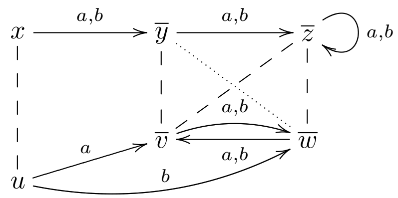
All three relations are bisimulations up to equivalence, whereas none of them are actual bisimulations. Consider, for example, the relation $\{(a,b)\}$: we have $\delta(a) = b$ and $\delta(b) = a$, but $(b,a) \notin \{(a,b)\}$. However, the pair (b,a) *is* in the least equivalence relation containing $\{(a,b)\}$.

The equivalence closure decomposes as

$$\mathsf{eq} = \mathsf{tra} \circ \mathsf{sym} \circ \mathsf{rfl}$$

where tra is transitive closure, sym is symmetric closure and rfl is reflexive closure. The relation $\{(a,b)\}$ from the above example is a bisimulation up to sym, $\{(g,h)\}$ is a bisimulation up to rfl and $\{(c,d),(d,e)\}$ is a bisimulation up to $\mathsf{tra} \circ \mathsf{sym}$.

Example 4.2.2. Consider the deterministic automaton below, with final states $\overline{y}, \overline{z}, \overline{v}, \overline{w}$ and transitions given by the solid arrows. The relation given by the four dashed lines together with the dotted line $(\overline{y}, \overline{w})$ is a bisimulation.



The relation R denoted by the four dashed lines is not a bisimulation, since $x \xrightarrow{b} y$ and $u \xrightarrow{b} w$ but $(y,w) \notin R$. However, R *is* a bisimulation up to equivalence, since the pair (y,w) is in $\mathsf{eq}(R)$. Hopcroft and Karp's algorithm [HK71] exploits this technique for checking equivalence of deterministic automata: rather than exploring n^2 pairs of states (where n is the number of states), the algorithm visits at most n pairs (that is the number of equivalence classes) (cf. [BP13]).

Bisimulation up to bisimilarity

Let $\sim$ be the bisimilarity relation of a given coalgebra $\delta\colon X \to BX$, and consider the *bisimilarity closure* function $\mathsf{bis}\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ defined by

$$\mathsf{bis}_\delta(R) = \sim \circ\, R \circ \sim .$$

The function bis_δ composes a relation with bisimilarity on both sides. In the sequel we sometimes drop the subscript δ and write bis, if the coalgebra under consideration is clear from the context.

A bisimulation up to bis is called a *bisimulation up to bisimilarity*. This is the very first up-to technique that appeared in the literature, in the context of labelled transition systems [Mil83].

Example 4.2.3. In this example, we prove that the stream $[1] = (1, 0, 0, \ldots)$ is the unit for the shuffle product $\otimes$, that is, $\sigma \otimes [1] \sim \sigma$. Let T be the set of terms given by the grammar $t ::= t \otimes t \mid t + t \mid [r]$, where $[r]$ ranges over $\{[r] \mid r \in \mathbb{R}\}$. As explained in Section 3.1.1, together with the appropriate behavioural differential equations, this induces a coalgebra $\langle (-)_0, (-)' \rangle \colon T \to \mathbb{R} \times T$.

We make use of the relation $R = \{(\sigma \otimes [1], \sigma) \mid \sigma \in T\}$. For any $\sigma \in T$, we have $(\sigma \otimes [1])_0 = \sigma_0 \cdot [1]_0 = \sigma_0$. Further $(\sigma \otimes [1])' = \sigma' \otimes [1] + \sigma \otimes [1]' = \sigma' \otimes [1] + \sigma \otimes [0]$; this element is not related to σ', so R is not a bisimulation. However given some basic laws of stream calculus, in particular $\sigma \otimes [0] \sim [0]$, $\sigma + [0] \sim \sigma$ and the fact that $\sim$ is a congruence, we obtain

$$((\sigma' \otimes [1]) + (\sigma \otimes [0])) \sim ((\sigma' \otimes [1]) + [0]) \sim (\sigma' \otimes [1]) \mathrel{R} \sigma'$$

so R is a bisimulation up to bisimilarity (we use that $\sim$ is reflexive and transitive on stream systems), proving that $\sigma \otimes [1] \sim \sigma$.

On a final coalgebra, bisimilarity implies equality, so bisimulation up to bisimilarity is not interesting there.

Bisimulation up to union

Given a fixed relation S, we define $\mathsf{un}_S \colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ by

$$\mathsf{un}_S(R) = R \cup S\,.$$

A bisimulation up to un_S, is called a *bisimulation up to union with S* or *bisimulation up to S-union*. If R is a bisimulation up to union with S, then next states are related either by R or by S. This technique is useful in combination with other ones, such as the equivalence closure eq. For instance, any bisimulation up to bisimilarity is also a bisimulation up to $\mathsf{eq} \circ \mathsf{un}_\sim$.

Bisimulation up to context

If the state space of the coalgebra under consideration has algebraic structure, then the notion of bisimulation up to context becomes relevant. Let $T \colon \mathsf{Set} \to \mathsf{Set}$ be a functor. For a T-algebra (X, α), the *contextual closure* function $\mathsf{ctx}_\alpha \colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ is defined using relation lifting (Section 3.2.1):

$$\mathsf{ctx}_\alpha(R) = (\alpha \times \alpha)(\mathsf{Rel}(T)(R)) = \{(\alpha \circ T\pi_1(t), \alpha \circ T\pi_2(t)) \mid t \in TR\}\,. \tag{4.1}$$

We call $\mathsf{ctx}_\alpha(R)$ the *contextual closure* of R. Whenever α is clear from the context we simply write $\mathsf{ctx}(R)$. If R is a bisimulation up to ctx then we call R a *bisimulation up to context*. In many of the examples, T is the underlying functor of a monad, and α is an algebra for the monad. However, the above definition does not require this: α is simply an algebra for the functor T.

Example 4.2.4. Let Σ^* be the free monad for a polynomial functor representing a signature, with multiplication $\mu\colon \Sigma^*\Sigma^* \Rightarrow \Sigma^*$ (Section 3.4). Given a relation $R \subseteq \Sigma^* X \times \Sigma^* X$, the contextual closure $\mathsf{ctx}_{\mu_X}(R) \subseteq \Sigma^* X \times \Sigma^* X$ can be inductively characterized by the following rules:

$$\frac{s \; R \; t}{s \; \mathsf{ctx}(R) \; t} \qquad \frac{s_i \; \mathsf{ctx}(R) \; t_i \quad i = 1 \dots n}{\sigma(s_1, \dots, s_n) \; \mathsf{ctx}(R) \; \sigma(t_1, \dots, t_n)} \quad \text{for each } \sigma \in \Sigma, |\sigma| = n$$

This slightly differs from the definition in [PS12] where the contextual closure is defined as

$$\mathsf{ctx}'(R) = \{(C[s_1, \dots s_n], C[t_1, \dots t_n]) \mid C \text{ a context and for all } i\text{: } (s_i, t_i) \in R\}$$

(a context C is a term with $n \geq 0$ holes $[\cdot]_i$ in it). In our case, ctx' can be obtained as $\mathsf{ctx} \circ \mathsf{rfl}$, i.e., by precomposing ctx with the reflexive closure function rfl. To see the difference, consider, for instance, the signature which has only a binary operator $+$, and let $R = \{(x, y)\}$. Then the pair $\{(x + x, x + y)\}$ is in $\mathsf{ctx}'(R)$ but not in $\mathsf{ctx}(R)$.

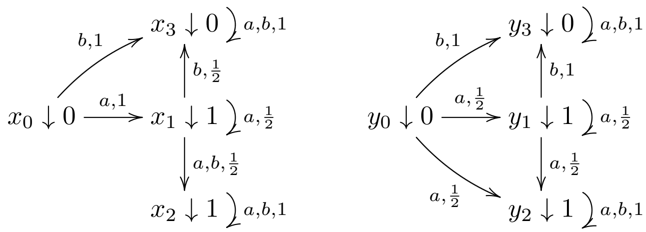
Example 4.2.5. Every weighted automaton $(X, \langle o, t\rangle)$ induces a coalgebra of the form $\langle o^\sharp, t^\sharp\rangle\colon \mathcal{M}X \to \mathbb{R} \times (\mathcal{M}X)^A$, where $\mathcal{M}X$ is the set of linear combinations with coefficients in $\mathbb{R}$. The coinductive extension of $\langle o^\sharp, t^\sharp\rangle$ maps a state x to the weighted language it accepts (Example 3.5.2). Therefore, we can prove weighted language equivalence between states x, y by proving that they are bisimilar on $\langle o^\sharp, t^\sharp\rangle$. In this example, we prove bisimilarity by constructing a bisimulation up to context, thus making use of the algebraic structure on $\mathcal{M}X$.

Given a relation $R \subseteq \mathcal{M}X \times \mathcal{M}X$, its contextual closure $\mathsf{ctx}(R) \subseteq \mathcal{M}X \times \mathcal{M}X$ (where the algebra is given by the multiplication of the monad $\mathcal{M}$, see Example 3.4.1) can be inductively characterized by the following rules:

$$\frac{v \; R \; w}{v \; \mathsf{ctx}(R) \; w} \qquad \frac{-}{0 \; \mathsf{ctx}(R) \; 0} \qquad \frac{v_1 \; \mathsf{ctx}(R) \; w_1 \quad v_2 \; \mathsf{ctx}(R) \; w_2}{v_1 + v_2 \; \mathsf{ctx}(R) \; w_1 + w_2} \qquad \frac{v \; \mathsf{ctx}(R) \; w \quad r \in \mathbb{R}}{r \cdot v \; \mathsf{ctx}(R) \; r \cdot w}$$

Now given a weighted automaton $\langle o, t\rangle\colon X \to \mathbb{R} \times (\mathcal{M}X)^A$, a bisimulation up to context is a relation $R \subseteq \mathcal{M}X \times \mathcal{M}X$ such that for all $(v, w) \in R$ we have $o_1^\sharp(v) = o_2^\sharp(w)$ and for all $a \in A$: $(t_1^\sharp(v)(a), t_2^\sharp(w)(a)) \in \mathsf{ctx}(R)$.

As an example, consider the following weighted automaton:



To prove that x_0 and y_0 are language equivalent, we need to prove that they are bisimilar on the induced $\mathbb{R} \times \mathsf{Id}^A$-coalgebra. But a bisimulation containing (x_0, y_0)

has to be infinite, since it needs to contain the pairs shown below by the dashed lines:

$$\begin{tikzcd}
x_0 \downarrow 0 \arrow[r, "a"] \arrow[d, dash, dashed] & x_1 \downarrow 1 \arrow[r, "a"] \arrow[d, dash, dashed] & \frac{1}{2}x_1 + \frac{1}{2}x_2 \downarrow 1 \arrow[r, "a"] \arrow[d, dash, dashed] & \frac{1}{4}x_1 + \frac{3}{4}x_2 \downarrow 1 \arrow[r, "a"] \arrow[d, dash, dashed] & \cdots \arrow[d, dash, dashed] \\
y_0 \downarrow 0 \arrow[r, "a"'] & \frac{1}{2}y_1 + \frac{1}{2}y_2 \downarrow 1 \arrow[r, "a"'] & \frac{1}{4}y_1 + \frac{3}{4}y_2 \downarrow 1 \arrow[r, "a"'] & \frac{1}{8}y_1 + \frac{7}{8}y_2 \downarrow 1 \arrow[r, "a"'] & \cdots
\end{tikzcd}$$

However, the *finite* relation $R = \{(x_0, y_0), (x_2, y_2), (x_3, y_3), (x_1, \frac{1}{2}y_1 + \frac{1}{2}y_2)\}$ is a bisimulation up to context: consider $(x_1, \frac{1}{2}y_1 + \frac{1}{2}y_2)$ (the other pairs are trivial) and observe that we have the following related pairs:

$$\begin{tikzcd}
x_1 \arrow[r, "a"] \arrow[d, dash, dashed, "R"'] & \frac{1}{2}x_1 + \frac{1}{2}x_2 \arrow[d, dash, dashed, "\mathsf{ctx}(R)"] \\
\frac{1}{2}y_1 + \frac{1}{2}y_2 \arrow[r, "a"'] & \frac{1}{4}y_1 + \frac{3}{4}y_2
\end{tikzcd}
\qquad
\begin{tikzcd}
x_1 \arrow[r, "b"] \arrow[d, dash, dashed, "R"'] & \frac{1}{2}x_3 + \frac{1}{2}x_2 \arrow[d, dash, dashed, "\mathsf{ctx}(R)"] \\
\frac{1}{2}y_1 + \frac{1}{2}y_2 \arrow[r, "b"'] & \frac{1}{2}y_3 + \frac{1}{2}y_2
\end{tikzcd}$$

Thus, the finite relation R is a bisimulation up to context. Since this technique is sound (as we will see in Section 4.4), this suffices to prove that x_0 and y_0 are bisimilar, and hence accept the same weighted language.

In the above example we used a finite bisimulation up to context to show weighted language equivalence. Finite bisimulations up to context for weighted automata are used in [Win15] to obtain a decidability result for weighted language equivalence for a certain class of semirings.

Bisimulation up to congruence

Given a T-algebra $\alpha \colon TX \to X$, the *congruence closure* function $\mathsf{cgr}_\alpha \colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ is defined by

$$\mathsf{cgr}_\alpha = \bigcup_{i \geq 0} (\mathsf{tra} \cup \mathsf{sym} \cup \mathsf{ctx}_\alpha \cup \mathsf{rfl})^i$$

(cf. [BP13]) where $\cup$ is pointwise union. If R is a bisimulation up to cgr_α, then we call R a *bisimulation up to congruence*. The congruence closure and associated notion of bisimulation up to congruence given in Definition 2.3.1 are a special case of the above. (In fact, many of the examples in Chapter 2 do not use the equivalence closure, and therefore are also examples of bisimulations up to context.)

Example 4.2.6. We consider weighted automata for the tropical semiring $\mathbb{T} = (\mathbb{R} \cup \{\infty\}, min, \infty, +, 0)$. In this semiring, the addition operation is given by the function min having ∞ as neutral element. The multiplication is given by the function $+$ having 0 as neutral element.

The weighted automaton $(X, \langle o, t\rangle)$ given as follows:

$$\begin{tikzcd}
x \downarrow 0 \arrow[r, bend left, "a{,}2"] \arrow[rr, bend right, "a{,}3"'] & y \downarrow 0 \arrow[l, bend left, "a{,}2"] & z \downarrow 0 \arrow[l, "a{,}2"'] & & u \downarrow 0 \arrow[loop right, "a{,}2"]
\end{tikzcd}$$

induces the coalgebra $(\mathcal{M}X, \langle o^\sharp, t^\sharp\rangle)$ which is partially depicted below (the transitions are given by the solid arrows, the dashed lines represent a relation).

$$\begin{tikzcd}
x \downarrow 0 \arrow[r, "a"] \arrow[d, dash, dashed] & min(2+y, 3+z) \downarrow 2 \arrow[r, "a"] \arrow[d, dash, dashed] & min(4+x, 5+y) \downarrow 4 \arrow[r, "a"] \arrow[d, dash, dashed] & \cdots \arrow[d, dash, dashed] \\
u \downarrow 0 \arrow[r, "a"'] & (2+u) \downarrow 2 \arrow[r, "a"'] & (4+u) \downarrow 4 \arrow[r, "a"'] & \cdots
\end{tikzcd}$$

The states x and u are weighted language equivalent. To prove it we would need an infinite bisimulation, since it should relate all the pairs of states linked by the dashed lines in the above figure.

Given a relation $R \subseteq \mathcal{M}X \times \mathcal{M}X$, its congruence closure cgr (where the algebra is given by the multiplication of the monad $\mathcal{M}$, see Example 3.4.1) can be characterized inductively by the following rules:

$$\frac{v\ R\ w}{v\ \mathsf{cgr}(R)\ w} \qquad \frac{}{v\ \mathsf{cgr}(R)\ v} \qquad \frac{v\ \mathsf{cgr}(R)\ w}{w\ \mathsf{cgr}(R)\ v} \qquad \frac{u\ \mathsf{cgr}(R)\ v\ \mathsf{cgr}(R)\ w}{u\ \mathsf{cgr}(R)\ w}$$

$$\frac{v_1\ \mathsf{cgr}(R)\ w_1 \quad v_2\ \mathsf{cgr}(R)\ w_2}{min(v_1, v_2)\ \mathsf{cgr}(R)\ min(w_1, w_2)} \qquad \frac{v\ \mathsf{cgr}(R)\ w \quad r \in \mathbb{R} \cup \{\infty\}}{r + v\ \mathsf{cgr}(R)\ r + w}$$

Now consider the relation $R = \{(x, u), (min(2+y, 3+z), 2+u)\}$. To prove that R is a bisimulation up to congruence we only have to show that $(min(4+x, 5+y), 4+u) \in \mathsf{cgr}(R)$:

$$\begin{array}{lll}
 & min(4+x, 5+y) & \\
\mathsf{cgr}(R) & min(4+u, 5+y) & ((x,u) \in R) \\
\mathsf{cgr}(R) & min(2+min(2+y, 3+z), 5+y) & ((min(2+y,3+z), 2+u) \in R) \\
= & 2 + min(2+y, 3+z) & \\
\mathsf{cgr}(R) & 4+u & ((min(2+y,3+z), 2+u) \in R)
\end{array}$$

Note that R is not a bisimulation up to context, since $(min(4+x, 5+y), 4+u) \notin \mathsf{ctx}(R)$. Here transitivity is really necessary.

Bisimulation up to union, context and equivalence

A bisimulation up to $\mathsf{eq} \circ \mathsf{ctx} \circ \mathsf{un}_S$ is called a *bisimulation up to S-union, context and equivalence*. This extension of bisimulation up to context allows to relate derivatives of R using $\mathsf{ctx}(R \cup S)$ in “multiple steps”, similar to the case of up-to-congruence.

Example 4.2.7. Recall the operations of shuffle product and inverse from Section 3.1.1, and let T_{wf} be the set of well-formed terms over shuffle product and inverse introduced there. We prove that the inverse operation is really the inverse of shuffle product, that is, $\sigma \otimes \sigma^{-1} \sim [1]$ for all $\sigma \in T_{wf}(\mathbb{R}^\omega)$ such that $\sigma_0 \neq 0$. We use that $\otimes$ is associative and commutative (so $\sigma \otimes \tau \sim \tau \otimes \sigma$, etc.) and that $\sigma + (-\sigma) \sim [0]$ (see, e.g., [Rut03]). Let

$$R = \{(\sigma \otimes \sigma^{-1}, [1]) \mid \sigma \in T_{wf}(\mathbb{R}^\omega), \sigma_0 \neq 0\} .$$

We can now establish that R is a bisimulation up to $\sim$-union, context and equivalence. First we consider the initial values:

$$(\sigma \otimes \sigma^{-1})_0 = \sigma_0 \cdot (\sigma^{-1})_0 = \sigma_0 \cdot (\sigma_0)^{-1} = 1 = [1]_0 .$$

Next, we relate the derivatives by $\mathsf{eq}(\mathsf{ctx}(R \cup \sim))$:

$$\begin{aligned}
(\sigma \otimes \sigma^{-1})' &= \sigma' \otimes \sigma^{-1} + \sigma \otimes (\sigma^{-1})' \\
&= \sigma' \otimes \sigma^{-1} + \sigma \otimes (-\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1})) \\
&\mathrel{\mathsf{tra}(\mathsf{ctx}(\sim))}\ (\sigma' \otimes \sigma^{-1}) + (-(\sigma' \otimes \sigma^{-1}) \otimes (\sigma \otimes \sigma^{-1})) \\
&\mathrel{\mathsf{ctx}(R \cup \sim)}\ (\sigma' \otimes \sigma^{-1}) + (-(\sigma' \otimes \sigma^{-1}) \otimes 1) \\
&\mathrel{\mathsf{tra}(\mathsf{ctx}(\sim))}\ [0] = [1]'
\end{aligned}$$

Since $\mathsf{tra}(\mathsf{ctx}(\sim)) \subseteq \mathsf{eq}(\mathsf{ctx}(R \cup \sim))$ and $\mathsf{ctx}(R \cup \sim) \subseteq \mathsf{eq}(\mathsf{ctx}(R \cup \sim))$ we may conclude that R is a bisimulation up to $\sim$-union, context, and equivalence. Notice that R is not a bisimulation; establishing that it is a bisimulation up-to is much easier than finding a bisimulation which contains R.

In the step above where we use $\mathsf{ctx}(R \cup \sim)$, we could have used $\mathsf{ctx}(\mathsf{rfl}(R))$ instead. Further, since in this example $\sim = \mathsf{tra}(\mathsf{ctx}(\sim))$, the above is also an example of *bisimulation up to context, reflexivity and bisimilarity*, that is, a bisimulation up to $\mathsf{bis} \circ \mathsf{ctx} \circ \mathsf{rfl}$. (Any bisimulation up to context, reflexivity and bisimilarity is also a bisimulation up to $\sim$-union, context and equivalence.)

4.3 Compatible functions

The above examples illustrate various up-to techniques available for bisimilarity. Many of these techniques are combinations of simpler ones; for instance, the equivalence closure is a composition of the transitive, symmetric and reflexive closure, and the congruence closure is a pointwise union of compositions of the transitive, symmetric, contextual and reflexive closure. Unfortunately, the soundness of a composed function does not follow from its basic constituents: the class of sound functions is not closed under composition. It is rather undesirable and sometimes difficult to reprove soundness of every suitable combination from scratch.

This calls for a theory of enhancements which allows one to freely compose them. Such a theory was developed in the setting of classical coinduction (Section 3.2), at the level of complete lattices [Pou07, PS12]. In the current section, we recall the basic definitions and results of this theory. In the next section, we instantiate it to prove soundness of coalgebraic bisimulation up-to in a modular way. In Section 5.1, the framework is generalized to an abstract categorical setting.

Let f be a monotone function on a complete lattice L. Recall from Section 3.2 that the coinductive proof principle then asserts that, to prove that $x \leq \mathsf{gfp}(f)$, it suffices to prove that $x \leq f(x)$. Enhancements of the coinductive proof method allow one to weaken the requirement that x is an f-invariant: rather than checking $x \leq f(x)$, we would like to check $x \leq f(y)$ for some y which is possibly above x. The key idea consists in using a function g to obtain this larger y out of x: $y = g(x)$. For instance, in the lattice of relations on a fixed set, we often consider functions that add more pairs to the relation.

Definition 4.3.1. Let $f, g \colon L \to L$ be monotone functions.

- An *f-invariant up to g* is an $f \circ g$-invariant, i.e., a post-fixed point of $f \circ g$.
- g is *f-sound* if all f-invariants up to g are below $\mathsf{gfp}(f)$, that is, if $x \leq f(g(x))$ then $x \leq \mathsf{gfp}(f)$.
- g is *f-compatible* if $g \circ f \leq f \circ g$.

The notion of f-compatible function, which is the heart of the matter, is introduced to get around the fact that f-sound functions cannot easily be composed. Compatible functions satisfy two crucial properties: f-compatible functions are f-sound (Theorem 4.3.2) and the composition of two f-compatible functions is again an f-compatible function (Proposition 4.3.3).

Theorem 4.3.2. *All f-compatible functions are f-sound.*

Proof. Let $f, g \colon L \to L$ be monotone and suppose g is f-compatible, i.e., $g \circ f \leq f \circ g$. Let $x \leq f(g(x))$ be an f-invariant up to g; we need to prove that $x \leq \mathsf{gfp}(f)$.

We first show that $g^i(x) \leq f(g^{i+1}(x))$ for every $i \in \mathbb{N}$, by induction on i. The base case $x \leq f(g(x))$ holds by the assumption that x is an f-invariant up to g. Now suppose $g^i(x) \leq f(g^{i+1}(x))$. Since g is monotone, this means $g^{i+1}(x) \leq g(f(g^{i+1}(x)))$, and since g is f-compatible we get

$$g^{i+1}(x) \leq g(f(g^{i+1}(x))) \leq f(g(g^{i+1}(x))) = f(g^{i+2}(x))$$

as desired.

Monotonicity of f gives $g^i(x) \leq f(\bigvee_{i \in \mathbb{N}} g^i(x))$, so $\bigvee_{i \in \mathbb{N}} g^i(x) \leq f(\bigvee_{i \in \mathbb{N}} g^i(x))$, which means $\bigvee_{i \in \mathbb{N}} g^i(x) \leq \mathsf{gfp}(f)$, so $x \leq \bigvee_{i \in \mathbb{N}} g^i(x) \leq \mathsf{gfp}(f)$. □

The main reason for the introduction of compatible functions is that they can be constructed by combining other compatible functions, as stated by the next result.

Proposition 4.3.3. *The following functions on L are f-compatible:*

1. id—*the identity function;*
2. cst_x—*the constant-to-x function, for any f-invariant x;*

3. *$g \circ h$ for any f-compatible functions g and h;*

4. *$\bigvee F$ for any set F of f-compatible functions.*

In a lattice of relations, the last item states that compatible functions can also be combined using pointwise union. There is another way of combining two functions g and h on relations, using relational composition:

$$(g \bullet h)(R) = g(R) \circ h(R) \tag{4.2}$$

This composition operator does *not* always preserve f-compatibility, but the following lemma gives a sufficient condition.

Proposition 4.3.4. *If $f\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ satisfies the following condition:*

$$\textit{for all relations } R, S \subseteq X \times X: \quad f(R) \circ f(S) \subseteq f(R \circ S) \tag{4.3}$$

then $g \bullet h$ is f-compatible for all f-compatible functions g and h.

This section is concluded with two lemmas that will be useful in the sequel. The first one gives an alternative characterization of f-compatible functions. The second lemma states that the coinductive predicate defined by f is closed under any f-compatible function.

Lemma 4.3.5. *A monotone function g is f-compatible iff for all x, y: $x \leq f(y)$ implies $g(x) \leq f(g(y))$.*

Lemma 4.3.6. *If g is f-compatible then $g(\mathsf{gfp}(f)) \leq \mathsf{gfp}(f)$.*

4.4 Compatibility results

We instantiate the framework of the previous section to prove soundness of bisimulation up-to techniques in a modular way, using the notion of compatible functions. To this end, recall from Section 3.2.1 that, given a coalgebra $\delta\colon X \to BX$, one can define the monotone function $\mathsf{b}_\delta(R) = (\delta \times \delta)^{-1}(\mathsf{Rel}(B)(R))$ on the complete lattice of relations on X ordered by inclusion, so that b_δ-invariants are precisely the bisimulations on δ. Progression and bisimulation up-to can also be stated in terms of this function, as an easy extension of Lemma 3.2.3.

Lemma 4.4.1. *For any coalgebra $\delta\colon X \to BX$ and for any relations $R, S \subseteq X \times X$: $R \subseteq \mathsf{b}_\delta(S)$ if and only if R progresses to S. As a consequence, given any monotone function $g\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ on the lattice of relations,*

$$R \textit{ is a bisimulation up to } g \textit{ if and only if it is a } \mathsf{b}_\delta\textit{-invariant up to } g\,.$$

Bisimilarity on δ coincides with the coinductive predicate defined by b_δ (i.e., $\mathsf{gfp}(\mathsf{b}_\delta)$).

Spelling out Definition 4.3.1, a monotone function g is b_δ-compatible if

$$g \circ \mathsf{b}_\delta \subseteq \mathsf{b}_\delta \circ g\,.$$

As a consequence of Lemma 4.4.1 and the fact that compatible functions are sound (Theorem 4.3.2), if g is b_δ-compatible then it is sound in the sense of Definition 4.1.2, i.e., bisimulation up to g is a sound proof technique for bisimilarity. Since compatible functions can be combined in various ways (Proposition 4.3.3), in particular by function composition, the advantage of proving compatibility rather than soundness is that it allows us to compositionally reason about the soundness of bisimulation up-to.

The instances of bisimulation up-to introduced in Section 4.2 can be roughly divided into three groups: (1) simple enhancements like up-to-union, (2) those that involve relational composition, such as up-to-transitivity and up-to-bisimilarity, and finally (3) up-to-context. Derived techniques such as up-to-congruence are just combinations of these basic enhancements, so their compatibility follows from proving the compatibility of their constituents.

In the remainder of this section, we show that functions (1) are compatible for any coalgebra, functions (2) are compatible under a mild condition on the behaviour functor, and functions (3) are compatible in the presence of a λ-bialgebra.

Theorem 4.4.2. *For any B-coalgebra (X, δ), the following are b_δ-compatible:*

1. un_S*—union with S, where S is a bisimulation on δ;*
2. rfl*—the reflexive closure;*
3. sym*—the symmetric closure.*

Proof. By definition, un_S is b_δ-compatible if $\mathsf{un}_S \circ \mathsf{b}_\delta \subseteq \mathsf{b}_\delta \circ \mathsf{un}_S$. Instead of proving this directly, we first decompose un_S as

$$\mathsf{un}_S(R) = R \cup S = \mathsf{id}(R) \cup \mathsf{cst}_S(R)\,.$$

By Proposition 4.3.3, id is b_δ-compatible, and the union of compatible functions is again compatible; so we only need to prove that the constant-to-S function cst_S is b_δ-compatible. Since S is a bisimulation, it is a b_δ-invariant, and thus by Proposition 4.3.3, the constant function cst_S is indeed b_δ-compatible.

For the compatibility of the reflexive closure, we use that the diagonal relation on any coalgebra is a bisimulation [Rut00]. Since $\mathsf{rfl} = \mathsf{un}_{\Delta_X}$, where Δ_X is the diagonal relation on X, rfl is b_δ-compatible by the first item.

Let $\mathsf{inv}(R) = R^{op}$. The symmetric closure sym is given by $\mathsf{sym}(R) = R \cup R^{op} = \mathsf{id}(R) \cup \mathsf{inv}(R)$. Thus, by Proposition 4.3.3, we obtain b_δ-compatibility of sym if we prove that inv is b_δ-compatible, i.e., that $\mathsf{inv} \circ \mathsf{b}_\delta \subseteq \mathsf{b}_\delta \circ \mathsf{inv}$. But this follows easily from the fact that $\mathsf{Rel}(B)(R^{op}) = (\mathsf{Rel}(B)(R))^{op}$ (Lemma 3.2.4). $\square$

4.4.1 Relational composition

Bisimilarity on coalgebras is not a transitive relation, in general. However, the mild condition that the behaviour functor preserves weak pullbacks guarantees that it is [Rut00]. Similarly, up-to techniques that are based on composition, such as bisimulation up to transitivity, are not sound in general. In this section, we show that weak pullback preservation is equivalent to the property (4.3) of Section 4.3. This property implies that the composition operator $\bullet$ from Section 4.3 (Equation (4.2)) preserves compatibility. From this fact, compatibility of the transitive closure and the bisimilarity closure can be derived.

First, we adapt an example from [AM89] to show that bisimulation up to bisimilarity is not sound in general.

Example 4.4.3. Define the functor $B\colon \mathsf{Set} \to \mathsf{Set}$ as

$$\begin{aligned} BX &= \{(x_1, x_2, x_3) \in X^3 \mid |\{x_1, x_2, x_3\}| \leq 2\} \\ B(f)(x_1, x_2, x_3) &= (f(x_1), f(x_2), f(x_3)) \end{aligned}$$

Consider the B-coalgebra with states $X = \{0, 1, 2, \tilde{0}, \tilde{1}\}$ and transition structure

$$\begin{array}{lll} 0 \mapsto (0,1,0) & \tilde{0} \mapsto (0,0,0) & 2 \mapsto (2,2,2) \\ 1 \mapsto (0,0,1) & \tilde{1} \mapsto (1,1,1) & \end{array}$$

Then $0 \not\sim 1$. To see this, note that in order for the pair $(0, 1)$ to be contained in a bisimulation R, there must be a transition structure on this relation which maps $(0, 1)$ to $((0, 0), (1, 0), (0, 1))$. But this triple can not be in BR, because it consists of three different elements. However, it is easy to show that $0 \sim 2$ and $1 \sim 2$: the relation $\{(0, 2), (1, 2)\}$ is a bisimulation.

The relation $S = \{(\tilde{0}, \tilde{1}), (2, 2)\}$ is not a bisimulation, since for that there should be a function from S to BS mapping the elements as follows:

$$(\tilde{0}, \tilde{1}) \mapsto ((0, 1), (0, 1), (0, 1)) \qquad (2, 2) \mapsto ((2, 2), (2, 2), (2, 2))$$

and $((0, 1), (0, 1), (0, 1))$ is not contained in BS. However, since $0 \sim 2\ S\ 2 \sim 1$, the triple $((0, 1), (0, 1), (0, 1))$ *is* contained in $B(\sim \circ S \circ \sim)$; so S is a bisimulation up to bisimilarity. Thus, if up-to-bisimilarity is sound, then $S \subseteq \sim$ and consequently $\tilde{0} \sim \tilde{1}$. It follows that $0 \sim 1$, which is a contradiction.

The key to obtaining b_δ-compatibility of functions that involve relational composition, is to assume that the behaviour functor B preserves weak pullbacks. Recall that a functor $B\colon \mathsf{Set} \to \mathsf{Set}$ preserves weak pullbacks if and only if $\mathsf{Rel}(B)$ preserves composition of relations (Theorem 3.2.5). A further equivalent condition is that bisimulations are closed under composition.

Theorem 4.4.4. *A functor $B\colon \mathsf{Set} \to \mathsf{Set}$ preserves weak pullbacks if and only if the composition of two B-bisimulations is again a B-bisimulation.*

Rutten [Rut00] established the implication from left to right, and the reverse implication is due to Gumm and Schröder [GS00]. Using Theorem 3.2.5 and Theorem 4.4.4 we show that preservation of weak pullbacks coincides with the property (4.3) of Section 4.3.

Proposition 4.4.5. *B preserves weak pullbacks iff for any B-coalgebra (X, δ), b_δ satisfies (4.3), i.e., for all relations R, S: $\mathsf{b}_\delta(R) \circ \mathsf{b}_\delta(S) \subseteq \mathsf{b}_\delta(R \circ S)$.*

Proof. Suppose B preserves weak pullbacks. Let (X, δ) be an B-coalgebra, $R, S \subseteq X \times X$ relations, and $(x, z) \in \mathsf{b}_\delta(R) \circ \mathsf{b}_\delta(S)$, so there is some y such that $(x, y) \in \mathsf{b}_\delta(R)$ and $(y, z) \in \mathsf{b}_\delta(S)$. Then we have $(\delta(x), \delta(y)) \in \mathsf{Rel}(B)(R)$ and $(\delta(y), \delta(z)) \in \mathsf{Rel}(B)(S)$, so $(\delta(x), \delta(z)) \in \mathsf{Rel}(B)(R) \circ \mathsf{Rel}(B)(S)$. But by assumption and Theorem 3.2.5 $\mathsf{Rel}(B)$ preserves composition, so $\mathsf{Rel}(B)(R) \circ \mathsf{Rel}(B)(S) = \mathsf{Rel}(B)(R \circ S)$. Consequently $(x, z) \in \mathsf{b}_\delta(R \circ S)$ as desired.

Conversely, suppose that (4.3) holds; then by Proposition 4.3.4, b_δ-compatible functions are closed under $\bullet$. Let R, S be bisimulations, so the constant-to-R function cst_R and the constant-to-S function cst_S are both b_δ-compatible by Proposition 4.3.3. By assumption $\mathsf{cst}_R \bullet \mathsf{cst}_S$ is b_δ-compatible, so by Lemma 4.3.5 we have $R \circ S \subseteq \mathsf{b}_\delta(R \circ S)$, and thus $R \circ S$ is a bisimulation. From Theorem 4.4.4 we conclude that B preserves weak pullbacks. (In fact, we only considered bisimulations on a single coalgebra, whereas the condition 2 of the theorem mentions arbitrary bisimulations; however, it is easy to prove that, in Set, if bisimulations on a single coalgebra compose then bisimulations on different coalgebras compose as well [RBB$^+$15]). □

As a consequence of Proposition 4.3.4 and the above result, b-compatible functions are closed under $\bullet$ if the behaviour functor preserves weak pullbacks.

Theorem 4.4.6. *Let (X, δ) be a coalgebra for a functor B that preserves weak pullbacks. The following functions are b_δ-compatible:*

1. tra—*the transitive closure;*
2. eq—*the equivalence closure;*
3. bis_δ—*the bisimilarity closure.*

Proof. If B preserves weak pullbacks, then b_δ-compatible functions are closed under $\bullet$, by Proposition 4.4.5 and Proposition 4.3.4.

For tra, inductively define the functions $(-)^{\bullet n}$ as $(-)^{\bullet 1} = \mathsf{id}$ and $(-)^{\bullet n+1} = \mathsf{id} \bullet (-)^{\bullet n}$. We thus have $(R)^{\bullet 1} = R$ and $(R)^{\bullet n+1} = R \circ R^{\bullet n}$. We prove by induction on n that $(-)^{\bullet n}$ is b_δ-compatible for any $n \in \mathbb{N}$. The base case is b_δ-compatibility of id, which follows from Proposition 4.3.3. Further, if $(-)^{\bullet n}$ is compatible then $(-)^{\bullet n+1} = \mathsf{id} \bullet (-)^{\bullet n}$ is also compatible. Thus

$$\mathsf{tra} = \bigcup_{n \geq 1} (-)^{\bullet n}$$

is a union of b_δ-compatible functions, so by Proposition 4.3.3 it is b_δ-compatible.

The equivalence closure is $\mathsf{eq} = \mathsf{tra} \circ \mathsf{sym} \circ \mathsf{rfl}$, which is a composition of b_δ-compatible functions and therefore b_δ-compatible.

For the bisimilarity closure bis_δ we have

$$\mathsf{bis}_\delta(R) = \sim \circ R \circ \sim = \mathsf{cst}_\sim \bullet \mathsf{id} \bullet \mathsf{cst}_\sim .$$

Since $\sim$ is a bisimulation, $\mathsf{cst}_\sim$ is b_δ-compatible. The b_δ-compatibility of bis_δ follows since b_δ-compatible functions are closed under $\bullet$, using the assumption. $\square$

4.4.2 Contextual closure

The contextual closure ctx_α is defined with respect to a T-algebra $\alpha\colon TX \to X$ on the states of a coalgebra $\delta\colon X \to BX$, see (4.1) in Section 4.2. A first thought may be that for compatibility of the contextual closure, it suffices if bisimilarity is a congruence with respect to this algebra, i.e., that bisimilarity is closed under the algebra structure. However, this is not even enough for the soundness of bisimulation up to context [PS12]. As we show below, in order to prove that ctx is compatible, it is sufficient to assume that (X, α, δ) is a λ-*bialgebra* for a distributive law $\lambda\colon TB \Rightarrow BT$ of the functor T over the functor B (thus, λ is simply a natural transformation).

Theorem 4.4.7. *Let (X, α, δ) be a λ-bialgebra for a distributive law $\lambda\colon TB \Rightarrow BT$ of T over B. The contextual closure function ctx_α is b_δ-compatible.*

Proof. Suppose $R \subseteq \mathsf{b}_\delta(S)$ for some R and S. We prove that $\mathsf{ctx}_\alpha(R) \subseteq \mathsf{b}_\delta(\mathsf{ctx}_\alpha(S)$; by Lemma 4.3.5 this implies that ctx_α is b_δ-compatible. Consider the following diagram:

$$\begin{tikzcd}
X \arrow[dd, "\delta"'] & TX \arrow[l, "\alpha"'] \arrow[d, "T\delta"'] & TR \arrow[l, "T\pi_1^R"'] \arrow[r, "T\pi_2^R"] \arrow[d, "T\gamma"] & TX \arrow[r, "\alpha"] \arrow[d, "T\delta"] & X \arrow[dd, "\delta"] \\
& TBX \arrow[d, "\lambda_X"'] & TBS \arrow[l, "TB\pi_1^S"'] \arrow[r, "TB\pi_2^S"] \arrow[d, "\lambda_S"] & TBX \arrow[d, "\lambda_X"] & \\
BX & BTX \arrow[l, "B\alpha"] & BT \arrow[l, "BT\pi_1^S"] \arrow[r, "BT\pi_2^S"'] & BTX \arrow[r, "B\alpha"'] & BX
\end{tikzcd}$$

The existence of γ and commutativity of the upper squares follow since $R \subseteq \mathsf{b}_\delta(S)$, by Lemma 4.4.1. The lower squares commute by naturality. The (outer) rectangles commute since (X, α, δ) is a λ-bialgebra.

We show that the above argument implies that $\mathsf{ctx}_\alpha(R)$ progresses to $\mathsf{ctx}_\alpha(S)$. Let $f_R\colon TR \to \mathsf{ctx}_\alpha(R)$ be the corestriction of $\langle \alpha\circ T\pi_1^R, \alpha\circ T\pi_2^R\rangle\colon TR \to X\times X$ to its range, so that $f_R(TR) = \mathsf{ctx}_\alpha(R)$. Let $f_S\colon TS \to \mathsf{ctx}_\alpha(S)$ be defined analogously, and take f_R^{-1} to be any right inverse of f_R (so we use the axiom of choice). Then the following diagram commutes:

$$
\begin{tikzcd}
 & & \mathsf{ctx}_\alpha(R) \arrow[dll, bend right, "\pi_1^{\mathsf{ctx}_\alpha(R)}"'] \arrow[drr, bend left, "\pi_2^{\mathsf{ctx}_\alpha(R)}"] \arrow[d, shift left, "f_R^{-1}"] & & \\
X \arrow[d, "\delta"] & TX \arrow[l, "\alpha"'] & TR \arrow[l, "T\pi_1^R"'] \arrow[r, "T\pi_2^R"] \arrow[u, shift left, "f_R"] \arrow[d, "\lambda_S \circ T\gamma"] & TX \arrow[r, "\alpha"] & X \arrow[d, "\delta"] \\
BX & BTX \arrow[l, "B\alpha"'] & BTS \arrow[l, "BT\pi_1^S"'] \arrow[r, "BT\pi_2^S"] \arrow[d, "B(f_S)"] & BTX \arrow[r, "B\alpha"] & BX \\
 & & B(\mathsf{ctx}_\alpha(S)) \arrow[ull, bend left, "B\pi_1^{\mathsf{ctx}_\alpha(S)}"] \arrow[urr, bend right, "B\pi_2^{\mathsf{ctx}_\alpha(S)}"'] & &
\end{tikzcd}
$$

This means that $\mathsf{ctx}_\alpha(R)$ progresses to $\mathsf{ctx}_\alpha(S)$, and thus $\mathsf{ctx}_\alpha(R) \subseteq \mathsf{b}_\alpha(\mathsf{ctx}_\alpha(S))$ by Lemma 4.4.1. □

Remark 4.4.8. The greatest bisimulation on a λ-bialgebra is closed under the algebraic operations. This was first shown by Turi and Plotkin [TP97] under the assumption that B preserves weak pullbacks; Bartels [Bar04] showed that this assumption is not necessary. We obtain the same result (for Set functors) as a direct consequence of the above Theorem and Lemma 4.3.6.

Under the assumption of a behaviour functor that preserves weak pullbacks and a λ-bialgebra, the *congruence closure* cgr_α is compatible as well, since it is a union of (compositions of) rfl, tra, sym and ctx_α, and each of these is compatible by Theorems 4.4.2, 4.4.6 and 4.4.7.

Coalgebras for copointed functors. There are many interesting examples of λ-bialgebras of the form $(X, \alpha, \langle \delta, \mathsf{id} \rangle)$, for some $\lambda \colon T(B \times \mathsf{Id}) \Rightarrow BT \times T$; in particular, this is relevant when λ arises from an abstract GSOS specification (Section 3.5). However, while Theorem 4.4.7 gives us $\mathsf{b}_{\langle \delta, \mathsf{id} \rangle}$-compatibility of the contextual closure ctx_α, it does *not* provide b_δ-compatibility. We recall a counterexample from [PS12].

Example 4.4.9 ([PS12])**.** Consider the following specification of the prefix and the replication operation on labelled transition systems:

$$
\frac{}{a.x \xrightarrow{a} x} \qquad\qquad \frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} !x|x'}
$$

together with the standard definition of the parallel operator $x|y$ (Example 3.5.4), and the constant 0, which has no transitions. This specification is in the GSOS format. While this is arguably not the best way to specify replication in the context of CCS [PS12], it suffices for our purposes. This specification induces a coalgebra on closed terms. Now abbreviate $b.0$ and $c.0$ by b and c respectively, and consider the relations $R = \{(!a.b, !a.c)\}$ and $S = \{(!a.b|b, !a.c|c)\}$. Then R progresses to S, but $\mathsf{ctx}(R)$ does not progress to $\mathsf{ctx}(S)$. For example, $(a.!a.b, a.!a.c) \in \mathsf{ctx}(R)$ but $!a.b$ is not related to $!a.c$ by $\mathsf{ctx}(S)$. Thus, by Lemma 4.3.5 the contextual closure ctx is not b_δ-compatible.

The solution of [PS12] is to consider invariants for a different function b'_δ, defined as $\mathsf{b}'_\delta(R) = \mathsf{b}_\delta(R) \cap R$. But $\mathsf{b}'_\delta = \mathsf{b}_{\langle \delta, \mathsf{id} \rangle}$ (an exercise in relation lifting), so in our framework this function arises naturally from the fact that one needs to consider a coalgebra for the cofree copointed functor in order to obtain compatibility.

In terms of progressions, we have $R \subseteq \mathsf{b}'_\delta(S)$ if and only if R progresses to S and $R \subseteq S$. Thus if R progresses to $g(R)$ for a function satisfying $R \subseteq g(R)$, then $R \subseteq \mathsf{b}'_\delta(g(R))$. But notice that for most functions g considered in Theorem 4.4.2 and Theorem 4.4.6 we have $R \subseteq g(R)$; an exception is the constant-to function. For the contextual closure function it suffices to assume that the functor T is *pointed*, i.e., there is a natural transformation $\eta\colon \mathsf{Id} \Rightarrow T$, and α is an algebra for this pointed functor, meaning that $\alpha \circ \eta = \mathsf{id}$. This holds in particular when α is an algebra for a monad (T, η, μ).

4.4.3 Bisimulation up-to modulo bisimilarity

We investigate the situation that there are two coalgebras on a common carrier, that behave the same up to bisimilarity. It turns out that in this case, if the functor preserves weak pullbacks, compatibility of a function g on one coalgebra can be transferred to compatibility of $\mathsf{bis} \circ g \circ \mathsf{bis}$ on the other. This rather technical result is only applied in Chapter 6, and does not play a further role in the current chapter. It was presented in [RB15].

Definition 4.4.10. Let δ, ϑ be B-coalgebras on a common carrier. We say δ and ϑ are *equal up to bisimilarity* if the bisimilarity relation $\sim_{\delta,\vartheta}$ between δ and ϑ is reflexive.

If B preserves weak pullbacks, then an equivalent definition is that the identity relation Δ is a bisimulation up to bisimilarity.

Lemma 4.4.11. *Let $\delta, \vartheta\colon X \to BX$ be coalgebras that are equal up to bisimilarity and assume that B preserves weak pullbacks. Then $\sim_\delta = \sim_{\delta,\vartheta} = \sim_\vartheta$.*

Proof. By assumption $\sim_{\delta,\vartheta}$ is reflexive, and by Theorem 4.4.4 the composition of two bisimulations is again a bisimulation. The desired equalities are now easy to prove; for example, $\sim_\delta \subseteq \sim_\delta \circ \sim_{\delta,\vartheta}$ by reflexivity of $\sim_{\delta,\vartheta}$, and $\sim_\delta \circ \sim_{\delta,\vartheta} \subseteq \sim_{\delta,\vartheta}$ since $\sim_\delta \circ \sim_{\delta,\vartheta}$ is a bisimulation between δ and ϑ and therefore contained in $\sim_{\delta,\vartheta}$, the greatest such bisimulation. Conversely, $\sim_{\delta,\vartheta} \subseteq \sim_{\delta,\vartheta} \circ \sim_{\vartheta,\delta} \subseteq \sim_\delta$ by a similar argument. $\square$

Lemma 4.4.12. *Let B, δ and ϑ be as in Lemma 4.4.11.*

1. *If $R \subseteq \mathsf{b}_\delta(S)$ then $\mathsf{bis}(R) \subseteq \mathsf{b}_\vartheta(\mathsf{bis}(S))$.*
2. *If g is b_δ-compatible then $\mathsf{bis} \circ g \circ \mathsf{bis}$ is b_ϑ-compatible.*

where bis *is defined w.r.t. the bisimilarity relation* $\sim$ *(of both* δ *and* ϑ*).*

Proof. Suppose $R \subseteq \mathsf{b}_\delta(S)$, and let $(x, y) \in R$; then $\delta(x)\, \mathsf{Rel}(B)(S)\, \delta(y)$. Since δ and ϑ are equal up to bisimilarity, we have $\vartheta(x)\, \mathsf{Rel}(B)(\sim)\, \delta(x)$ and $\delta(y)\, \mathsf{Rel}(B)(\sim)\, \vartheta(y)$. Hence

$$\vartheta(x)\, \mathsf{Rel}(B)(\sim)\, \delta(x)\, \mathsf{Rel}(B)(S)\, \delta(y)\, \mathsf{Rel}(B)(\sim)\, \vartheta(y)$$

and since B preserves weak pullbacks, this implies $\vartheta(x)\, \mathsf{Rel}(B)(\sim \circ S \circ \sim)\, \vartheta(y)$ (Theorem 3.2.5). Thus $R \subseteq \mathsf{b}_\vartheta(\sim \circ S \circ \sim)$; by compatibility of bis and Lemma 4.3.5 this implies $\sim \circ R \circ \sim \subseteq \mathsf{b}_\vartheta(\sim \circ \sim \circ S \circ \sim \circ \sim)$, and by transitivity of $\sim$ (B preserves weak pullbacks) then $\mathsf{bis}(R) \subseteq \mathsf{b}_\vartheta(\mathsf{bis}(S))$.

For (2), suppose $R \subseteq \mathsf{b}_\vartheta(S)$. By (1) (replacing δ by ϑ and vice versa) then $\mathsf{bis}(R) \subseteq \mathsf{b}_\delta(\mathsf{bis}(S))$. We apply b_δ-compatibility of g to obtain $g \circ \mathsf{bis}(R) \subseteq \mathsf{b}_\delta(g \circ \mathsf{bis}(S))$. Finally, again apply (1) and get $\mathsf{bis} \circ g \circ \mathsf{bis}(R) \subseteq \mathsf{b}_\vartheta(\mathsf{bis} \circ g \circ \mathsf{bis}(S))$. □

4.5 Behavioural equivalence up-to

Whenever the functor B does not preserve weak pullbacks (as it is the case, for instance, with certain types of weighted transition systems [GS01, Kli09, BBB+12]) one can consider *behavioural equivalence*, rather than bisimilarity. In the current section, we instantiate the framework of Section 4.3 to develop up-to techniques for behavioural equivalence.

Recall from Section 3.1 that behavioural equivalence $\approx$ on a coalgebra $\delta\colon X \to BX$ is defined as follows: $x \approx y$ iff there is a homomorphism h from (X, δ) into some coalgebra such that $h(x) = h(y)$. As we see below (Lemma 4.5.1), behavioural equivalence $\approx$ can equivalently be characterized as the greatest fixed point of the monotone function $\mathsf{be}_\delta\colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ on the lattice of relations on X, defined as follows [AM89]:

$$\mathsf{be}_\delta(R) = \{(x, y) \mid Bq_R \circ \delta(x) = Bq_R \circ \delta(y)\}$$

where $q_R\colon X \to X/\mathsf{eq}(R)$ is the quotient map of $\mathsf{eq}(R)$ (we sometimes drop the subscript δ from be_δ if it is clear from the context).

Lemma 4.5.1. *Let* $\approx$ *be behavioural equivalence on a coalgebra* $\delta\colon X \to BX$*. Then* $x \approx y$ *if and only if there is a relation* R *such that* $R \subseteq \mathsf{be}_\delta(R)$ *and* $(x, y) \in R$*.*

Proof. The quotient map q_R from the definition of $\mathsf{be}_\delta(R)$ is a coequalizer, and therefore a coalgebra morphism [Rut00, Theorem 4.2], which gives the implication from right to left. For the converse, we let h be a coalgebra morphism from δ and we prove that the kernel $\mathsf{ker}(h) = \{(x, y) \mid h(x) = h(y)\}$ of h is a be_δ-invariant, i.e., we show that the following inclusion holds:

$$\mathsf{ker}(h) \subseteq \mathsf{be}_\delta(\mathsf{ker}(h)) = \{(x, y) \mid Bq \circ \delta(x) = Bq \circ \delta(y)\}$$

where $q\colon X \to X/\mathsf{ker}(h)$ is the quotient map of (the equivalence relation) $\mathsf{ker}(h)$. By [Rut00, Theorem 7.1], h equals the composition of coalgebra homomorphisms

$h = m \circ q$ where q is as above and m is a monomorphism. This means that $q(x) = q(y)$ for any $(x, y) \in \mathsf{ker}(h)$, and since q is a coalgebra morphism from δ, we get $Bq \circ \delta(x) = Bq \circ \delta(y)$. Thus $\mathsf{ker}(h) \subseteq \mathsf{be}_\delta(\mathsf{ker}(h))$. □

The relation R of Example 4.2.2 is a be_δ-invariant. Note that, intuitively, be_δ-invariants are implicitly "up to equivalence", since the next states can be related by the equivalence closure $\mathsf{eq}(R)$.

We proceed to consider be-compatibility of the equivalence closure and contextual closure. In the previous section, we used the property (4.3) from Section 4.3 to prove b-compatibility of transitive and equivalence closure. However, this property does *not* hold for be, that is, in general it does not hold that $\mathsf{be}(R) \circ \mathsf{be}(S) \subseteq \mathsf{be}(R \circ S)$. This is shown by the following example.

Example 4.5.2. Consider the identity functor $BX = X$ and the B-coalgebra with states $\{x, y\}$ and transitions $x \mapsto x$ and $y \mapsto y$. Let $R = \{(x, y)\}$. Then $\mathsf{be}(R) = \{(x, x), (y, y), (x, y), (y, x)\}$ and $\mathsf{be}(\emptyset) = \{(x, x), (y, y)\}$. Now

$$\mathsf{be}(R) \circ \mathsf{be}(\emptyset) = \{(x, x), (y, y), (x, y), (y, x)\}\,, \text{whereas}$$
$$\mathsf{be}(R \circ \emptyset) = \mathsf{be}(\emptyset) = \{(x, x), (y, y)\}\,.$$

Indeed, $\mathsf{be}(R) \circ \mathsf{be}(\emptyset)$ is not included in $\mathsf{be}(R \circ \emptyset)$.

This motivates to prove be-compatibility of eq directly.

Theorem 4.5.3. *Let (X, δ) be any coalgebra. The following are be_δ-compatible:*

1. rfl—*the reflexive closure;*
2. eq—*the equivalence closure;*
3. un_S—*union with S (for a behavioural equivalence S).*

Proof. Items 1 and 3 are analogous to Theorem 4.4.2. We proceed with the compatibility of the equivalence closure. First, notice that $\mathsf{eq} \circ \mathsf{be} = \mathsf{be}$ since $\mathsf{be}(R)$ is an equivalence relation for any relation R. Second, since $\mathsf{eq}(R) = \mathsf{eq}(\mathsf{eq}(R))$ for any R, the quotient maps in the definition of $\mathsf{be}(R)$ and $\mathsf{be}(\mathsf{eq}(R))$ are equal, so $\mathsf{be}(R) = \mathsf{be}(\mathsf{eq}(R))$. Thus $\mathsf{eq} \circ \mathsf{be} = \mathsf{be} = \mathsf{be} \circ \mathsf{eq}$. □

Notice that the be-compatibility of the equivalence closure does not require any assumptions on the functor.

For the compatibility of contextual closure a λ-bialgebra is required, similar to the case of bisimulations in Theorem 4.4.7. However, in the case of behavioural equivalence, we require an algebra for a *monad*, although λ is still only required to be a distributive law between functors, that is, a plain natural transformation. Further, in the proof we need an additional assumption. A pair of functions $f, g \colon X \to Y$ is *reflexive* if it has a common section: a map $s \colon Y \to X$ such that $f \circ s = \mathsf{id} = g \circ s$. A *reflexive coequalizer* is a coequalizer of a reflexive pair. Reflexive coequalizers are important in the theory of monads, see, e.g., [BW05]. Below we need the underlying functor T of the monad to preserve reflexive coequalizers, which is a non-trivial condition in Set; see [AKV00, Example 4.3] for an example of a functor that does not satisfy this property. We do not know if these additional assumptions can be dropped.

Theorem 4.5.4. *Let (T, η, μ) be a monad so that T preserves reflexive coequalizers, and let (X, α, δ) be a λ-bialgebra for a distributive law $\lambda \colon TB \Rightarrow BT$ (between functors), where α is an algebra for the monad (T, η, μ). Then $\mathsf{ctx}_\alpha \circ \mathsf{rfl}$ is be_δ-compatible.*

Proof. Suppose $R \subseteq \mathsf{be}_\delta(S)$ for some relations $R, S \subseteq X \times X$. By Theorem 4.5.3 rfl is be_δ-compatible, so $\mathsf{rfl}(R) \subseteq \mathsf{be}_\delta \circ \mathsf{rfl}(S)$. Further $\mathsf{rfl}(S) \subseteq \mathsf{ctx}_\alpha \circ \mathsf{rfl}(S)$, using the fact that α is an algebra for the monad (see the last part of Section 4.4.2). Therefore

$$\mathsf{rfl}(R) \subseteq \mathsf{be}_\delta \circ \mathsf{ctx}_\alpha \circ \mathsf{rfl}(S)\,. \tag{4.4}$$

Let $q \colon X \to X'$ be the quotient map of $\mathsf{eq} \circ \mathsf{ctx}_\alpha \circ \mathsf{rfl}(S)$ and its projections, or, equivalently, the coequalizer of the two composite arrows $\alpha \circ T\pi_1$, $\alpha \circ T\pi_2$ in the bottom of the diagram below:

$$\begin{tikzcd}
TT(\mathsf{rfl}(S)) \arrow[r, "TT\pi_1{,}TT\pi_2"] \arrow[d, "\mu_{\mathsf{rfl}(S)}"'] & TTX \arrow[r, "T\alpha"] \arrow[d, "\mu_X"] & TX \arrow[r, "Tq"] \arrow[d, "\alpha"] & TX' \arrow[d, dashed, "\alpha'"] \\
T(\mathsf{rfl}(S)) \arrow[r, "T\pi_1{,}T\pi_2"'] & TX \arrow[r, "\alpha"'] & X \arrow[r, "q"'] & X'
\end{tikzcd} \tag{4.5}$$

Define $d \colon X \to \mathsf{rfl}(S)$ by $d(x) = (x, x)$. Then the map $Td \circ \eta_X \colon X \to T(\mathsf{rfl}(S))$ is a section of the pair $\alpha \circ T\pi_1$, $\alpha \circ T\pi_2$, since $\alpha \circ T\pi_1 \circ Td \circ \eta_X = \alpha \circ \eta_X = \alpha \circ T\pi_2 \circ Td \circ \eta_X$ and $\alpha \circ \eta_X = \mathsf{id}$. Thus, $\alpha \circ T\pi_1$, $\alpha \circ T\pi_2$ is a reflexive pair, and q a reflexive coequalizer. The square on the left commutes (for $T\pi_1$ and $T\pi_2$ separately) by naturality, and the middle since α is an algebra for the monad. Since T preserves reflexive coequalizers, Tq is a coequalizer, and the map α' making the right-hand square commute arises by its universal property.

Now consider the following diagram:

$$\begin{tikzcd}
T(\mathsf{rfl}(R)) \arrow[r, shift left, "T\pi_1"] \arrow[r, shift right, "T\pi_2"'] & TX \arrow[r, "T\delta"] \arrow[dd, "\alpha"'] & TBX \arrow[r, "TBq"] \arrow[d, "\lambda_X"] & TBX' \arrow[d, "\lambda_{X'}"] \\
 & & BTX \arrow[r, "BTq"] \arrow[d, "B\alpha"] & BTX' \arrow[d, "B\alpha'"] \\
 & X \arrow[r, "\delta"'] & BX \arrow[r, "Bq"'] & BX'
\end{tikzcd}$$

The top horizontal paths commute by (4.4) and functoriality. The rectangle commutes by the assumption that (X, α, δ) is a λ-bialgebra. The upper square commutes by naturality of λ, and the lower square by (4.5) and functoriality. Thus we have $Bq \circ \delta \circ \alpha \circ T\pi_1 = Bq \circ \delta \circ \alpha \circ T\pi_2$, and consequently

$$\begin{tikzcd} \mathsf{ctx}_\alpha(\mathsf{rfl}(R)) \arrow[r, shift left, "\pi_1"] \arrow[r, shift right, "\pi_2"'] & X \arrow[r, "\delta"] & BX \arrow[r, "Bq"] & BX' \end{tikzcd}$$

commutes, which means $\mathsf{ctx}_\alpha \circ \mathsf{rfl}(R) \subseteq \mathsf{be}_\delta \circ \mathsf{ctx}_\alpha \circ \mathsf{rfl}(S)$. By Lemma 4.3.5, $\mathsf{ctx}_\alpha \circ \mathsf{rfl}$ is be_δ-compatible. $\square$

The above result also applies to coalgebras of the form $\langle \delta, \mathsf{id} \rangle$, similar to the situation described for b_δ-compatibility in Section 4.4.2.

Example 4.5.5. For an example of behavioural equivalence up-to, we consider the *general process algebra with transition costs* (GPA) from [BK01]. GPA processes are defined for a given set of labels A and a semiring $\mathbb{S}$ which, for this example, we fix to be the semiring of reals $\mathbb{R}$ with the usual addition and multiplication. The operational semantics of GPA is expressed in terms of weighted transition systems, that is, coalgebras for the functor $(\mathcal{M}-)^A$ (Example 3.1.1).

As shown in Section 2.3 of [BBB$^+$12], the functor $(\mathcal{M}-)^A$ does not preserve weak pullbacks and therefore bisimulation up-to cannot be used in this context. However, thanks to Theorem 4.5.3 we can use behavioural equivalence up-to.

First observe that, by instantiating the definition of be above to a coalgebra $\delta \colon X \to (\mathcal{M}X)^A$, one obtains the function $\mathsf{be}_\delta \colon \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ defined for a relation $R \subseteq X \times X$ as

$$\mathsf{be}_\delta(R) = \{(x_1, x_2) \mid \forall a \in A, y \in X : \sum_{y' \in [y]_R} \delta(x_1)(a)(y') = \sum_{y' \in [y]_R} \delta(x_2)(a)(y')\}$$

where $[y]_R$ denotes the equivalence class of y with respect to $\mathsf{eq}(R)$. Our notion of behavioural equivalence coincides with the notion of bisimilarity in [BK01] (which differs from coalgebraic bisimilarity).

To illustrate our example it suffices to consider a small fragment of GPA. The set P of *basic GPA processes* is defined by

$$p ::= 0 \mid p + p \mid (a, r).p$$

where $a \in A$, $r \in \mathbb{R}$. The operational semantics of basic GPA processes is given by the coalgebra $\delta \colon P \to (\mathcal{M}P)^A$ defined for all $a' \in A$ and $p' \in P$ as follows:

$$\begin{array}{rcl} \delta(0)(a')(p') & = & 0 \\ \delta((a,r).p)(a')(p') & = & \begin{cases} r & \text{if } a = a', p = p' \\ 0 & \text{otherwise} \end{cases} \\ \delta(p_1 + p_2)(a')(p') & = & \delta(p_1)(a')(p') + \delta(p_2)(a')(p') \end{array}$$

As an example, the operational semantics of $(a,1).0 + (a,-1).(a,0).0$ is as follows.

$$\begin{tikzcd} & 0 \\ (a,1).0 + (a,-1).(a,0).0 \arrow[ur, "{a,1}"] \arrow[dr, "{a,-1}"'] & \\ & (a,0).0 \end{tikzcd} \tag{4.6}$$

Since $0 \approx (a,0).0$, we have that $(a,1).0 + (a,-1).(a,0).0 \approx 0$. More generally, it holds that for all $a \in A$, $r \in R$, p_1 and $p_2 \in P$:

$$\text{if } p_1 \approx p_2 \text{ then } 0 \approx (a,r).p_1 + (a,-r).p_2. \tag{4.7}$$

We prove (4.7) using behavioural equivalence up to union with $\approx$ (Theorem 4.5.3). To this end, consider the relation

$$R = \{(0, (a,r).p_1 + (a,-r).p_2) \mid p_1 \approx p_2\}\,.$$

Note that R is *not* a be_δ-invariant. For instance, 0 does not make any transitions whereas $(a,1).0 + (a,-1).(a,0).0$ makes two transitions, to processes that are not in the same equivalence class with respect to $\mathsf{eq}(R)$ (see (4.6)); thus $R \not\subseteq \mathsf{be}_\delta(R)$.

Instead, we prove that R is a be_δ-invariant up to $\mathsf{un}_\approx$, that is, $R \subseteq \mathsf{be}_\delta(R \cup \approx)$. We must show that for any $p = (a,r).p_1 + (a,-r).p_2$ and any process $q \in P$:

$$\sum_{y' \in [q]_{R\cup\approx}} \delta(0)(a)(y') = 0 = \sum_{y' \in [q]_{R\cup\approx}} \delta(p)(a)(y').$$

The left-hand equality comes from the semantics of the process 0. For the right-hand equality, if $p_1 \in [q]_{R\cup\approx}$ then also $p_2 \in [q]_{R\cup\approx}$ (and vice versa), which means that $\sum_{y' \in [q]_{R\cup\approx}} \delta(p)(a)(y') = r - r = 0$. If $p_1 \notin [q]_{R\cup\approx}$, then $p_2 \notin [q]_{R\cup\approx}$, so $\sum_{y' \in [q]_{R\cup\approx}} \delta(p)(a)(y') = 0$. We conclude that R is a be_δ-invariant up to $\mathsf{un}_\approx$.

4.6 Discussion and related work

In this chapter we have proved the soundness of a range of bisimulation up-to techniques by proving their *compatibility*. Compatible functions are sound, and are closed under composition. We conclude with a technical summary of the main compatibility results that are introduced in this chapter. In the table below we assume an arbitrary coalgebra $\delta\colon X \to BX$, an algebra $\alpha\colon TX \to X$ (for a functor T) and a distributive law λ of the functor T over the functor B. All functions in the table are defined in Section 4.2. Recall that if a function is b_δ-compatible, then bisimulation up to g is sound (Section 4.4).

Name	Notation	Condition b_δ-compatibility
Union with S	un_S	S is a bisimulation
Equivalence closure	eq	B preserves weak pullbacks
Bisimilarity closure	bis_δ	B preserves weak pullbacks
Contextual closure	ctx_α	(X,α,δ) a λ-bialgebra
Congruence closure	cgr_α	(X,α,δ) a λ-bialgebra, B pres. weak pullbacks

Further, we proved soundness of several up-to techniques for behavioural equivalence, by proving that they are be_δ-compatible (Section 4.5). The equivalence closure eq is be_δ-compatible for *any* functor. The contextual closure ctx_α is be_δ-compatible if (X, α, δ) is a λ-bialgebra, α is an algebra for a *monad*, and T preserves reflexive coequalizers. It remains open whether the latter two assumptions are necessary.

A discussion of related work can be found in Chapter 5, which generalizes all of the above results on the soundness of bisimulation up-to.

Chapter 5

Coinduction up-to

In the previous chapter, we have seen how up-to techniques enhance the proof method for bisimilarity. In the current chapter, we extend these results to a coalgebraic framework for up-to techniques that is applicable not only to bisimilarity but to a wide variety of coinductive predicates. For instance, this approach allows us to obtain sound up-to techniques for unary predicates such as divergence of processes and for binary predicates such as similarity, or language inclusion of weighted automata over an ordered semiring.

We build on the observation that coinductive predicates can be viewed as final coalgebras in a suitable category, so that the classical coinductive proof principle amounts to finality (explained in Section 3.2). We show that Pous's modular framework of compatible up-to techniques (Section 4.3) has a natural counterpart at this categorical level in terms of *compatible functors*, which are functors equipped with a suitable natural transformation. The modular aspect of this framework amounts to elementary manipulations and constructions on natural transformations. Moreover, the fact that every compatible functor yields a sound up-to technique turns out to be a basic result on distributive laws between functors.

In Section 3.3, we recalled how coinductive predicates can be studied in a structural and systematic way using *fibrations*, which provide an abstract notion of predicates. There, the coinductive predicate of interest is defined uniformly based on a lifting of the behaviour functor to a category of predicates. We instantiate the above mentioned framework of compatible functors within this fibrational setting, and consequently obtain a modular approach for defining and reasoning about up-to techniques for general coinductive predicates. In this setting, we introduce enhancements such as up-to-context, up-to-equivalence and up-to-behavioural equivalence. We prove their compatibility under conditions on the functor liftings under consideration.

By instantiating these abstract results we obtain concrete sound enhancements, with the results of Chapter 4 on bisimulation up-to as a special case. We treat divergence of processes as an example of a unary predicate, and inclusion of weighted automata as an example based on a non-standard version of up-to-context. Further, we apply the framework to prove the soundness of up-to techniques for *simulation* as introduced in [HJ04]. As a special case, we obtain that simulation up to context is compatible (sound) for any *monotone* GSOS specification (instantiated to GSOS for labelled transition systems, this means that there are no negative premises). This includes simulation up-to for languages as introduced in Chapter 2.

Outline. In the next section, we propose the notion of compatible functor. The (technical) heart of this chapter is Section 5.2, where we introduce the main up-to techniques and associated compatibility theorems. In Section 5.3, we show how to instantiate these theorems, and in Section 5.4 we derive the compatibility of simulation up-to for a mild restriction of abstract GSOS. In Section 5.5, we discuss related and future work, and provide a short summary of the soundness results.

5.1 Compatible functors

In Chapter 4, we have used Pous's lattice-theoretic framework of up-to techniques as a modular approach for proving the soundness of bisimulation up-to techniques. In the current section, we show how Pous's framework generalizes to a categorical setting, where complete lattices and monotone functions are replaced by categories and functors (Section 3.2.2).

In that categorical setting, proving a coinductive predicate determined by a given functor $F\colon \mathcal{C} \to \mathcal{C}$ amounts to the construction of a suitable F-invariant (F-coalgebra). In the current chapter, we introduce up-to techniques to construct F-invariants in an easier way; hence, these techniques can be seen as enhanced proof techniques for the coinductive predicate (final coalgebra) of F. However, we focus on proof techniques for constructing invariants and ignore the coinductive predicate, and therefore we do not depend on the existence of a final F-coalgebra.

In the definition below, the intuition is that F-invariants are the coinductive properties of interest, and $G\colon \mathcal{C} \to \mathcal{C}$ is a potential up-to technique.

- An *F-invariant up to G* is an FG-invariant, i.e., a coalgebra $R \to FGR$.
- G is *F-sound* if, for every FG-invariant, there exists a $\mathcal{C}$-arrow from its carrier into the carrier of an F-invariant.

It is easy to see that these definitions generalize the notions of invariants up-to and soundness from Section 4.3.

Recall that *compatibility* is the central notion of Pous's framework: given two monotone functions f, g on a complete lattice, g is said to be f-compatible if $g \circ f \subseteq f \circ g$. If g is f-compatible then it is sound, i.e., every f-simulation up to g is contained in an f-simulation (Theorem 4.3.2). This result is an instance of a more general fact from the theory of distributive laws between functors.

Theorem 5.1.1. *Suppose $\mathcal{C}$ is a category with countable coproducts, $F, G\colon \mathcal{C} \to \mathcal{C}$ are functors and $\gamma\colon GF \Rightarrow FG$ is a natural transformation. Then for any FG-coalgebra*

δ there is an F-coalgebra ϑ making the next diagram commute:

$$
\begin{tikzcd}
X \arrow[r, "\kappa_0"] \arrow[d, "\delta"'] & G^\omega X \arrow[d, "\vartheta"] \\
FGX \arrow[r, "F\kappa_1"'] & FG^\omega X
\end{tikzcd}
$$

Here $G^\omega X$ denotes the coproduct $\coprod_{i\in\mathbb{N}} G^i X$ of all finite iterations of G applied to X, with coproduct injections $\kappa_i \colon G^i X \to G^\omega X$.

This appears in the proof of [Bar03, Theorem 3.8], but for a complete presentation we include a proof.

Proof. Define $\vartheta_i \colon G^i X \to FG^{i+1} X$ inductively as $\vartheta_0 = \delta$ and

$$\vartheta_{i+1} = \; GG^i X \xrightarrow{G\vartheta_i} GFG^{i+1} X \xrightarrow{\gamma_{G^{i+1}X}} FGG^{i+1} X$$

Postcomposing these morphisms with the coproduct injections yields a cocone $(F\kappa_{i+1} \circ \vartheta_i \colon G^i X \to FG^\omega X)_{i\in\mathbb{N}}$ and by the universal property of $G^\omega X$ we obtain a coalgebra $\vartheta \colon G^\omega X \to FG^\omega X$. Commutativity of the diagram amounts to the base case ϑ_0. $\square$

(Alternatively, we can replace the countable coproduct G^ω by the free monad for G, assuming it exists. In this case, the result is an instance of the construction (3.14) in Section 3.5.1.)

If $\mathcal{C}$ is a preorder, then F and G are monotone functions, and the existence of a natural transformation amounts to compatibility as in Pous's framework. The fact that compatible functions are sound, is thus an instance of Theorem 5.1.1. Similarly, that f-compatible functions preserve the coinductive predicate defined by f (Lemma 4.3.6) is an instance of the fact that, if $\gamma \colon GF \Rightarrow FG$ is a distributive law, then a final F-coalgebra lifts to a final γ-bialgebra (Lemma 3.5.1). When $\mathcal{C}$ is a lattice, the fact that there is a G-algebra structure on the final coalgebra $Z = \mathsf{gfp}(F)$ simply means that $G(Z) \leq Z$ (cf. Lemma 4.3.6).

The main reason for studying compatible functions is their compositionality properties. To achieve a flexible approach to the construction of compatible functors, we define them as follows.

Definition 5.1.2. Let $F_1 \colon \mathcal{C}_1 \to \mathcal{C}_1$ and $F_2 \colon \mathcal{C}_2 \to \mathcal{C}_2$ be functors. We say a functor $G \colon \mathcal{C}_1 \to \mathcal{C}_2$ is (F_1, F_2)-*compatible* when there exists a natural transformation $\gamma \colon GF_1 \Rightarrow F_2 G$.

The pair (G, γ) is a morphism between endofunctors F_1 and F_2 in the sense of [LPW00]. In the remainder of this chapter, we often leave γ implicit, as the examples involve only categories that are preorders.

An important instance of the above definition is (F^n, F^m)-compatibility of a functor $G \colon \mathcal{C}^n \to \mathcal{C}^m$; in this case, we simply say that $G \colon \mathcal{C}^n \to \mathcal{C}^m$ is F-compatible. For example, coproduct then becomes a compatible functor by itself, rather than a way to compose compatible functors.

Proposition 5.1.3. *Compatible functors are closed under the following constructions:*

1. *composition: if G is (F_1, F_2)-compatible and G' is (F_2, F_3)-compatible, then $G' \circ G$ is (F_1, F_3)-compatible;*
2. *pairing: if $(G_i)_{i \in I}$ are (F_1, F_2)-compatible, then $\langle G_i \rangle_{i \in I}$ is (F_1, F_2^I)-compatible.*

Moreover, for any functor $F \colon \mathcal{C} \to \mathcal{C}$:

3. *the identity functor $\mathsf{Id} \colon \mathcal{C} \to \mathcal{C}$ is F-compatible;*
4. *the constant functor to the carrier of an F-coalgebra is F-compatible, in particular to the coinductive predicate defined by F (carrier of the final F-coalgebra), if it exists;*
5. *the coproduct functor $\coprod_I \colon \mathcal{C}^I \to \mathcal{C}$ is (F^I, F)-compatible.*

Proof. 1. By assumption we have natural transformations $\gamma \colon GF_1 \Rightarrow F_2 G$ and $\gamma' \colon G'F_2 \Rightarrow F_3 G'$, and composing them yields

$$G'GF_1 \xRightarrow{G'\gamma} G'F_2G \xRightarrow{\gamma' G} F_3G'G$$

which is a natural transformation of the desired type.

2. Given natural transformations $\gamma_i \colon G_i F_1 \Rightarrow F_2 G_i$ for all $i \in I$, we have

$$\langle G_i \rangle_{i \in I} F_1 = \langle G_i F_1 \rangle_{i \in I} \xRightarrow{\gamma} \langle F_2 G_i \rangle_{i \in I} = F_2^I \langle G_i \rangle_{i \in I}$$

where $\gamma_X = ((\gamma_i)_X)_{i \in I}$ for any X.

Items 3 and 4 are trivial. For 5, we must find a natural transformation

$$\gamma \colon \coprod_I \circ F^I \Rightarrow F \circ \coprod_I .$$

On a component $(X_i)_{i \in I}$ it is defined using the universal property; applying F to the coproduct injections $\kappa_i \colon X_i \to \coprod_{i \in I} X_i$ yields a morphism $F\kappa_i \colon FX_i \to F \coprod_{i \in I} X_i$ for each $i \in I$. $\square$

In a lattice, the pointwise join of compatible functions is again compatible (Proposition 4.3.3). To retrieve this in the current setting, suppose $(G_i)_{i \in I}$ are (F_1, F_2)-compatible. Since the pairing of compatible functors is compatible, and the coproduct functor is compatible, composing them yields a compatible functor $\coprod_I \circ \langle G_i \rangle_{i \in I}$ (this is the coproduct of the functors G_i), which, in a lattice, is pointwise join of monotone functions. Further, in the next section we will see how to obtain the operator $\bullet$ defined in Equation (4.2) of Section 4.3, by combining a functor that composes relations with the pairing constructor.

Further compositionality could be obtained by defining a pair (G, G') of endofunctors to be F-compatible if there exists a natural transformation $\gamma \colon GF \Rightarrow FG'$. A suitable variant of Proposition 5.1.3 then allows to prove compatibility, modular in the shape of the functor F. A related approach is taken in [LLYL14]. In this chapter we do not consider such constructions, instead focusing on the combination of up-to techniques for a fixed functor F.

5.2 Compatibility results

In Section 3.3, we have seen how fibrations can be used to speak generally about coinductive predicates on coalgebras. In that approach, the invariants of interest are themselves coalgebras which live in the fibre above the carrier of a coalgebra in the base category.

In order to define both coinductive predicates and up-to techniques, we assume

- a bifibration $p\colon \mathcal{E} \to \mathcal{A}$ (see Section 3.3.1 for details);
- a coalgebra $\delta\colon X \to BX$ for a functor $B\colon \mathcal{A} \to \mathcal{A}$, and
- a lifting $\overline{B}\colon \mathcal{E} \to \mathcal{E}$ of B.

As explained in Section 3.3, the lifting $\overline{B}$ and the transition structure δ determine a functor on the fibre $\mathcal{E}_X$ above the carrier X of the coalgebra (X, δ), defined as follows:

$$\overline{B}_\delta = \delta^* \circ \overline{B}_X : \mathcal{E}_X \to \mathcal{E}_X\,.$$

We spell out the important definitions of invariants up-to, soundness and compatibility, for the functor $\overline{B}_\delta$. A *$\overline{B}_\delta$-invariant* is a coalgebra $R \to \overline{B}_\delta(R)$, where R is an object in $\mathcal{E}_X$. Given a functor $G\colon \mathcal{E}_X \to \mathcal{E}_X$, a *$\overline{B}_\delta$-invariant up to G* is a coalgebra $R \to \overline{B}_\delta(G(R))$.

Our interest is to find functors G that are *sound*, so that invariants up to G are a valid proof principle for the construction of $\overline{B}_\delta$-invariants. Instead of proving soundness, we focus on proving the stronger notion of *compatibility*. By definition, a functor $G\colon \mathcal{E}_X \to \mathcal{E}_X$ is $\overline{B}_\delta$-compatible if there exists a natural transformation

$$\gamma\colon G \circ \overline{B}_\delta \Rightarrow \overline{B}_\delta \circ G\,.$$

In the remainder of this section, we introduce three families of up-to techniques:

- behavioural equivalence (Section 5.2.1),
- equivalence closure (Section 5.2.2) and
- contextual closure (Section 5.2.3).

We prove their compatibility, based on conditions on the lifting $\overline{B}$ of B. As explained in the previous section, this suffices to show that they are sound, and that they can be combined in various ways to form new sound up-to techniques.

In Section 3.2.1 we associated to each coalgebra $\delta\colon X \to BX$ for a functor $B\colon \mathsf{Set} \to \mathsf{Set}$ a function b_δ, whose invariants are bisimulations. In the current setting, this can be obtained by choosing $\overline{B}$ to be the canonical relation lifting $\mathsf{Rel}(B)$ of B. Then:

$$\overline{B}_\delta(R) = \mathsf{Rel}(B)_\delta(R) = (\delta \times \delta)^{-1}(\mathsf{Rel}(B)(R)) = \mathsf{b}_\delta(R)$$

which means that $\overline{B}_\delta$-invariants are bisimulations on δ (Lemma 3.2.3). For all three types of up-to techniques, we study the canonical relation lifting as a special case, and retrieve all the b_δ-compatibility results from the previous chapter.

In Section 5.3 and Section 5.4, we consider examples and instances for liftings other than $\mathsf{Rel}(B)$, to obtain proof techniques for other coinductive predicates than bisimilarity.

5.2.1 Behavioural equivalence

The first technique that we introduce is up-to-behavioural equivalence. If $\delta\colon X \to BX$ is a coalgebra for a functor $B\colon \mathsf{Set} \to \mathsf{Set}$, then behavioural equivalence is the relation $\approx$ on its carrier given by $x \approx y$ iff $h(x) = h(y)$, where h is the coinductive extension of δ, i.e., the unique coalgebra morphism into the final coalgebra (assumed to exist), see Section 3.1. Now consider the function $\mathsf{bhv}_\delta\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ defined by

$$\mathsf{bhv}_\delta(R) = \approx \circ\, R \circ \approx .$$

To define bhv_δ more generally in the setting of a bifibration, observe that

$$\begin{aligned}\mathsf{bhv}_\delta(R) &= \{(x,y) \mid \exists u, v.\, h(x) = h(u),\, h(y) = h(v) \text{ and } (u,v) \in R\} \\ &= h^{-1}(\{(h(u), h(v)) \mid (u,v) \in R\}) \\ &= h^{-1}(h(R)) .\end{aligned}$$

But $h^{-1} \circ h$ is simply direct image followed by reindexing in the fibration $\mathsf{Rel} \to \mathsf{Set}$, i.e., $h^{-1}(h(R)) = h^* \circ \coprod_h(R)$ (see Section 3.3.1). Therefore, we can generalize the above function bhv_δ to an arbitrary bifibration $p\colon \mathcal{E} \to \mathcal{A}$, a functor $B\colon \mathcal{A} \to \mathcal{A}$ with a final coalgebra, and a coalgebra $\delta\colon X \to BX$ by defining the *behavioural equivalence closure* bhv_δ as

$$\mathsf{bhv}_\delta = h^* \circ \textstyle\coprod_h : \mathcal{E}_X \to \mathcal{E}_X$$

where h is the coinductive extension of δ. We sometimes write bhv instead of bhv_δ, if δ is clear from the context. In the predicate fibration $\mathsf{Pred} \to \mathsf{Set}$, we have

$$\mathsf{bhv}_\delta(P) = h^{-1}(h(P)) = h^{-1}(\{h(u) \mid u \in P\}) = \{x \mid \exists u \in P.\, h(x) = h(u)\} .$$

Our aim is to prove $\overline{B}_\delta$-compatibility of bhv_δ. This is an instance of the following result, which concerns a generalization of bhv_δ to arbitrary coalgebra morphisms (rather than the coinductive extension h).

Theorem 5.2.1. *Suppose that $(\overline{B}, B)$ is a fibration map. For any B-coalgebra morphism $h\colon (X,\delta) \to (Y,\vartheta)$, the functor $h^* \circ \coprod_h$ is $\overline{B}_\delta$-compatible.*

Proof. We exhibit a natural transformation

$$(h^* \circ \textstyle\coprod_h) \circ (\delta^* \circ \overline{B}_X) \Rightarrow (\delta^* \circ \overline{B}_X) \circ (h^* \circ \coprod_h)$$

obtained by pasting the 2-cells (natural transformations) $(a), (b), (c), (d)$ in the following diagram:

$$
\begin{tikzcd}
\mathcal{E}_X \arrow[rr, "\overline{B}"] \arrow[dd, equal] & & \mathcal{E}_{BX} \arrow[rr, "\delta^*"] \arrow[drr, "\coprod_{Bh}"'] & & \mathcal{E}_X \arrow[rr, "\coprod_h"] \arrow[d, phantom, "\Downarrow(d)"] & & \mathcal{E}_Y \arrow[rr, "h^*"] & & \mathcal{E}_X \arrow[dd, equal] \\
 & \Downarrow(b) & & & \mathcal{E}_{BY} \arrow[urr, "\vartheta^*"'] \arrow[drr, "(Bh)^*"] \arrow[d, phantom, "\Downarrow(a)"] & & & \Downarrow(c) & \\
\mathcal{E}_X \arrow[rr, "\coprod_h"'] & & \mathcal{E}_Y \arrow[rr, "h^*"'] \arrow[urr, "\overline{B}"] & & \mathcal{E}_X \arrow[rr, "\overline{B}"'] & & \mathcal{E}_{BX} \arrow[rr, "\delta^*"'] & & \mathcal{E}_X
\end{tikzcd}
$$

(a) $(\overline{B}, B)$ is a fibration map, so $\overline{B} \circ h^* \cong (Bh)^* \circ \overline{B}$.

(b) $\overline{B}$ is a lifting of B; this is an instance of Lemma 3.3.4.

(c) h is a coalgebra homomorphism, i.e., $\vartheta \circ h = Bh \circ \delta$, and consequently $(\vartheta \circ h)^* = (Bh \circ \delta)^*$. Combining this with the natural isomorphisms $h^* \circ \vartheta^* \cong (\vartheta \circ h)^*$ and $(Bh \circ \delta)^* \cong \delta^* \circ (Bh)^*$ shows that the required 2-cell is a natural isomorphism.

(d) follows from (c); see the proof of Proposition 3.3.7. For convenience we repeat the construction of the natural transformation:

$$\coprod_h \circ \delta^* \Longrightarrow \coprod_h \circ \delta^* \circ (Bh)^* \circ \coprod_{Bh} \Longrightarrow \coprod_h \circ h^* \circ \vartheta^* \circ \coprod_{Bh} \Longrightarrow \vartheta^* \circ \coprod_{Bh} .$$

The natural transformation on the left is the unit of the adjunction $\coprod_{Bh} \dashv (Bh)^*$, the middle is (c), and the one on the right is the counit of $\coprod_h \dashv h^*$. $\square$

We first instantiate this to the canonical relation lifting $\mathsf{Rel}(B)$ of a Set functor B. To this end, we use that $(\mathsf{Rel}(B), B)$ is a fibration map whenever B preserves weak pullbacks (Lemma 3.3.3). The functor $\mathsf{Rel}(B)_\delta$ coincides with b_δ, so from Theorem 5.2.1 we directly obtain:

Corollary 5.2.2. *If B is a* Set *functor preserving weak pullbacks then the behavioural equivalence closure functor* bhv_δ *is* b_δ*-compatible.*

If (X, δ) is a coalgebra for a functor B that preserves weak pullbacks, then behavioural equivalence $\approx$ coincides with bisimilarity $\sim$ (Lemma 3.1.6). Hence, in that case, the bisimilarity closure bis_δ defined in Section 4.2 coincides with the behavioural equivalence closure bhv_δ:

$$\mathsf{bis}_\delta(R) = (\sim \circ R \circ \sim) = (\approx \circ R \circ \approx) = \mathsf{bhv}_\delta(R) .$$

Thus, the fact that bis_δ is b_δ-compatible if B preserves weak pullbacks (Theorem 4.4.6) follows from Corollary 5.2.2, and hence is a special case of Theorem 5.2.1.

From Theorem 5.2.1 we also derive the soundness of up-to bhv for unary predicates that are defined by a *modality* $m \colon B2 \to 2$, where B is a functor on Set.

Modalities are in one-to-one correspondence to *predicate liftings*, which are natural transformations of the form $2^{\mathsf{Id}} \Rightarrow 2^B$ [Sch05, Proposition 20]. If such a predicate lifting is monotone, then it defines a lifting $\overline{B}\colon \mathsf{Pred} \to \mathsf{Pred}$ of B, which maps a predicate $X \to 2$ to $BX \to B2 \xrightarrow{m} 2$. Recall that with predicates viewed as functions $X \to 2$ reindexing is precomposition; then it is easy to show that the lifting induced by a modality is a fibration map. Consequently, we have:

Corollary 5.2.3. *If* $\overline{B}\colon \mathsf{Pred} \to \mathsf{Pred}$ *arises from a modality* $m\colon B2 \to 2$ *as explained above, then* bhv *is* $\overline{B}_\delta$*-compatible.*

5.2.2 Relational composition and equivalence

We propose a general approach for deriving the compatibility of the reflexive, symmetric and transitive closure. Composing these functors yields compatibility of the equivalence closure.

For transitive closure, it suffices to show that relational composition is compatible. Relational composition can be expressed in a fibrational setting by considering the category $\mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel}$ obtained as a pullback (in the category Cat of categories[1]) of the fibration $\mathsf{Rel} \to \mathsf{Set}$ along itself:

$$\begin{tikzcd}
\mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel} \arrow[r] \arrow[d] & \mathsf{Rel} \arrow[d] \\
\mathsf{Rel} \arrow[r] & \mathsf{Set}
\end{tikzcd}$$

The objects of $\mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel}$ are pairs of relations $R, S \subseteq X \times X$ on a common carrier X. An arrow from $R, S \subseteq X \times X$ to $R', S' \subseteq Y \times Y$ is a pair of morphisms in Rel above a common $f\colon X \to Y$; thus, it is a map $f\colon X \to Y$ such that $f(R) \subseteq R'$ and $f(S) \subseteq S'$. Then relational composition can be presented as a functor

$$\otimes\colon \mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel} \to \mathsf{Rel}$$

mapping relations $R, S \subseteq X \times X$ to their composition.

The pullback $\mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel}$ above is, in fact, a product in the category $\mathsf{Fib}(\mathsf{Set})$ of fibrations over Set. Indeed, $\mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel} \to \mathsf{Set}$ is again a fibration. In order to treat not only relational composition but also, e.g., symmetric and reflexive closure, we move to a more general setting of n-fold products. Consider for an arbitrary fibration $\mathcal{E} \to \mathcal{A}$ its n-fold product in $\mathsf{Fib}(\mathcal{A})$ (see [Jac99, Lemma 1.7.4]), denoted by $\mathcal{E}^{\times^n_{\mathcal{A}}} \to \mathcal{A}$ and defined by pullback in Cat. We have

$$(\mathcal{E}^{\times^n_{\mathcal{A}}})_X = (\mathcal{E}_X)^n \quad \text{and} \quad \mathcal{E}^0 = \mathcal{A}\,.$$

Concretely, the objects in $\mathcal{E}^{\times^n_{\mathcal{A}}}$ are n-tuples of objects in $\mathcal{E}$ belonging to the same fibre, and an arrow from $(R_1, \ldots, R_n)$ above X to $(S_1, \ldots, S_n)$ above Y consists

[1] We assume that Cat contains large categories such as Set and Rel; see [Lan98] for various ways to justify this at a foundational level.

of a tuple of arrows $(f_1 \colon R_1 \to S_1, \ldots, f_n \colon R_n \to S_n)$ that sit above a common $f \colon X \to Y$.

Hereafter, we are interested in functors $G \colon \mathcal{E}^{\times^n_{\mathcal{A}}} \to \mathcal{E}$ that are liftings of the identity functor on $\mathcal{A}$, meaning that the following diagram commutes:

$$\begin{tikzcd} \mathcal{E}^{\times^n_{\mathcal{A}}} \arrow[rr, "G"] \arrow[dr] & & \mathcal{E} \arrow[dl] \\ & \mathcal{A} & \end{tikzcd}$$

Given such a functor G, for each X in $\mathcal{A}$ we have functors $G_X \colon (\mathcal{E}_X)^n \to \mathcal{E}_X$.

For the relation fibration $\mathsf{Rel} \to \mathsf{Set}$, we have three interesting instances of such functors G:

- $(n = 0)$: $\mathsf{diag} \colon \mathsf{Set} \to \mathsf{Rel}$ mapping a set X to the diagonal relation Δ_X;
- $(n = 1)$: $\mathsf{inv} \colon \mathcal{E} \to \mathcal{E}$ mapping a relation R to its converse R^{op};
- $(n = 2)$: $\otimes \colon \mathcal{E} \times_{\mathsf{Set}} \mathcal{E} \to \mathcal{E}$ mapping relations R, S to their composition $R \circ S$.

Next, we provide a general condition on functors $G \colon \mathcal{E}^{\times^n_{\mathcal{A}}} \to \mathcal{E}$ as above and on the lifting $\overline{B}$ that guarantees G_X to be $\overline{B}_\delta$-compatible.

Theorem 5.2.4. *Let $\delta \colon X \to BX$ be a coalgebra. Let $G \colon \mathcal{E}^{\times^n_{\mathcal{A}}} \to \mathcal{E}$ be a lifting of the identity functor on $\mathcal{A}$ such that there exists a natural transformation*

$$\gamma \colon G_{BX} \circ (\overline{B}_X)^n \Rightarrow \overline{B}_X \circ G_X : (\mathcal{E}_X)^n \to \mathcal{E}_{BX} \, .$$

Then G_X is $\overline{B}_\delta$-compatible.

Proof. The goal is to construct a natural transformation of the form

$$G_X \circ (\delta^* \circ \overline{B}_X)^n \Rightarrow (\delta^* \circ \overline{B}_X) \circ G_X \, .$$

First, observe that there is a natural transformation

$$\theta \colon G_X \circ (\delta^*)^n \Rightarrow \delta^* \circ G_{BX} : (\mathcal{E}_{BX})^n \to \mathcal{E}_X \, .$$

by Lemma 3.3.4 (instantiated to $B = \mathsf{Id}$ and $\overline{B} = G$), using that reindexing along an $\mathcal{A}$-morphism f in $\mathcal{E}^{\times^n_{\mathcal{A}}}$ is $(f^*)^n$, where f^* is the reindexing functor in $\mathcal{E}$. (To see this, one can use the characterization of Cartesian morphisms in fibrations obtained by change-of-base and composition, which are the basic operations used to construct the fibration $\mathcal{E}^{\times^n_{\mathcal{A}}} \to \mathcal{A}$ [Jac99, Lemma 1.7.4].)

The desired natural transformation is now obtained as follows:

$$\begin{tikzcd} G_X \circ (\delta^* \circ \overline{B}_X)^n \arrow[r, equal] & G_X \circ (\delta^*)^n \circ (\overline{B}_X)^n \arrow[d, Rightarrow, "\theta(\overline{B}_X)^n"] & \\ & \delta^* \circ G_{BX} \circ (\overline{B}_X)^n \arrow[r, Rightarrow, "\delta^*\gamma"] & \delta^* \circ \overline{B}_X \circ G_X \end{tikzcd}$$

The first equality follows from the definition of $(-)^n$ as the mediating arrow into the product $(\mathcal{E}_X)^n$. $\square$

The use of the above theorem is that compatibility is reduced to checking the existence of a natural transformation that does not mention the coalgebra under consideration. We list several applications of the theorem for the fibration $\mathsf{Rel} \to \mathsf{Set}$. In this case, a natural transformation $G_{BX} \circ (\overline{B}_X)^n \Rightarrow \overline{B}_X \circ G_X$ exists precisely if for all relations $R_1, \ldots, R_n$ on the carrier X:

$$G(\overline{B}(R_1), \ldots, \overline{B}(R_n)) \subseteq \overline{B}G(R_1, \ldots, R_n)\,.$$

Instantiating this, we obtain concrete compatibility results for functors $\mathsf{Rel}^{\times^n_{\mathsf{Set}}} \to \mathsf{Rel}$, including relational composition.

Corollary 5.2.5. *Suppose $\overline{B}\colon \mathsf{Rel} \to \mathsf{Rel}$ is a lifting of B, and $\delta\colon X \to BX$ a B-coalgebra.*

1. *Let $\mathsf{diag}\colon \mathsf{Set} \to \mathsf{Rel}$ be the functor mapping each set to the associated diagonal relation. The functor $\mathsf{diag}_X : 1 \to \mathsf{Rel}_X$ is $\overline{B}_\delta$-compatible if:*

$$\Delta_{BX} \subseteq \overline{B}(\Delta_X)\,. \tag{5.1}$$

2. *Let $\mathsf{inv}\colon \mathsf{Rel} \to \mathsf{Rel}$ be the functor mapping each relation to its converse. The functor $\mathsf{inv}_X : \mathsf{Rel}_X \to \mathsf{Rel}_X$ is $\overline{B}_\delta$-compatible if for all relations $R \subseteq X^2$:*

$$(\overline{B}R)^{op} \subseteq \overline{B}(R^{op})\,. \tag{5.2}$$

3. *Let $\otimes\colon \mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel} \to \mathsf{Rel}$ be the relational composition functor. The functor $\otimes_X : \mathsf{Rel}_X \times \mathsf{Rel}_X \to \mathsf{Rel}_X$ is $\overline{B}_\delta$-compatible if for all $R, S \subseteq X^2$:*

$$\overline{B}(R) \otimes \overline{B}(S) \subseteq \overline{B}(R \otimes S)\,. \tag{5.3}$$

Note that $\overline{B}_\delta$-compatibility of diag_X simply means that $\Delta_X \subseteq \overline{B}_\delta(\Delta_X)$, i.e., the diagonal is a $\overline{B}_\delta$-invariant.

If relational composition is $\overline{B}_\delta$-compatible, and $F_1, F_2\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ are two $\overline{B}_\delta$-compatible functors, then their pointwise composition

$$F_1 \bullet F_2 = \otimes_X \circ \langle F_1, F_2 \rangle$$

is $\overline{B}_\delta$-compatible. This way of combining compatible functors corresponds to the operator $\bullet$ in Section 4.3 (4.2).

This operator $\bullet$ was used to prove the compatibility of transitive closure in the more concrete setting of the previous chapter (Theorem 4.4.6). We follow the same reasoning and define the transitive closure functor as follows:

$$\mathsf{tra} = \coprod \circ \langle (-)^{\bullet i} \rangle_{i \geq 1} : \mathsf{Rel}_X \to \mathsf{Rel}_X$$

where $(-)^{\bullet i}\colon \mathsf{Rel} \to \mathsf{Rel}$ is defined inductively: $(-)^{\bullet 1} = \mathsf{Id}$ and $(-)^{\bullet n+1} = \mathsf{Id} \bullet (-)^{\bullet n}$. By Proposition 5.1.3, compatibility of $\bullet$ implies compatibility of tra.

The above conditions (5.1) and (5.2) always hold for the canonical lifting $\overline{B} = \mathsf{Rel}(B)$; (5.3) holds for $\mathsf{Rel}(B)$ when B preserves weak pullbacks (Theorem 3.2.5). Thus, we retrieve the b_δ-compatibility of reflexive, symmetric and transitive closure (and hence also the equivalence closure eq), as proved in Theorem 4.4.6, as a special case of Corollary 5.2.5.

When $\overline{B}_\delta$ has a final coalgebra with carrier Z, one can define a *self closure* functor $\mathsf{slf}\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ by

$$\mathsf{slf}_\delta(R) = (\mathsf{cst}_Z \bullet \mathsf{Id} \bullet \mathsf{cst}_Z)(R) = Z \otimes R \otimes Z$$

where $\mathsf{cst}_Z\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ is the constant-to-Z functor. By Proposition 5.1.3 and the above, the functor slf is compatible whenever $\otimes$ is. If B is a Set functor and $\overline{B}$ is instantiated to the canonical relation lifting, then Z is the bisimilarity relation $\sim$, so

$$\mathsf{slf}_\delta(R) = \sim \circ R \circ \sim = \mathsf{bis}_\delta(R)$$

where bis_δ is the bisimilarity closure, defined in Section 4.2.

5.2.3 Contextual closure

In this section, we study the compatibility of the contextual closure. To this end, we assume an algebra $\alpha\colon TX \to X$ for some functor $T\colon \mathcal{A} \to \mathcal{A}$. Then contextual closure is defined using the bifibrational structure of p, parameterized by a lifting $\overline{T}$ of T:

$$\mathcal{E}_X \xrightarrow{\overline{T}_X} \mathcal{E}_{TX} \xrightarrow{\coprod_\alpha} \mathcal{E}_X$$

If T is a Set functor, then instantiating $\overline{T}$ to the canonical relation lifting $\mathsf{Rel}(T)$ yields the usual contextual closure, denoted ctx_α, as defined in Section 4.2.

However, taking different liftings of $\overline{T}$ yields different types of contextual closure, similar to the fact that taking different liftings of $\overline{B}$ to define $\overline{B}_\delta$ yields different coinductive predicates. Indeed, in the next section we consider the *left contextual closure* for reasoning about divergence, and the *monotone contextual closure* for weighted automata; both contextual closures differ from ctx_α.

Given liftings of T and B, compatibility of the associated contextual closure requires a λ-bialgebra, similar to the case of bisimulation up to context in Theorem 4.4.7. Additionally, it is required that λ lifts to a natural transformation between the lifted functors. All this is stated in Theorem 5.2.7 below; we require the following basic result for its proof.

Lemma 5.2.6. *Let $p\colon \mathcal{E} \to \mathcal{A}$ be a fibration, and F, G endofunctors on $\mathcal{A}$ with liftings $\overline{F}$ and $\overline{G}$ respectively. Given a natural transformation $\overline{\lambda}\colon \overline{F} \Rightarrow \overline{G}$ above some $\lambda\colon F \Rightarrow G$, there exists for every object X in $\mathcal{A}$ a natural transformation*

$$\theta\colon \overline{F}_X \Rightarrow (\lambda_X)^* \circ \overline{G}_X : \mathcal{E}_X \to \mathcal{E}_{FX}$$

Proof. For any R in $\mathcal{E}_X$ we use the universal property of the Cartesian lifting $\widetilde{(\lambda_X)}_{\overline{G}R}$ to define θ_R:

$$
\begin{tikzcd}
\overline{F}(R) \arrow[d, dashed, "\theta_R"'] \arrow[drr, "\overline{\lambda}_R"] & & \\
\lambda_X^*(\overline{G}(R)) \arrow[rr, "\widetilde{(\lambda_X)}_{\overline{G}R}"'] & & \overline{G}(R)
\end{tikzcd}
$$

$$
\begin{tikzcd}
FX \arrow[r, "\lambda_X"] & GX
\end{tikzcd}
$$

Naturality is straightforward using the uniqueness of the factorisation and the definition of the reindexing functor on morphisms. □

Theorem 5.2.7. *Suppose (X, α, δ) is a λ-bialgebra for some natural transformation $\lambda\colon TB \Rightarrow BT$, and suppose there exists a natural transformation $\overline{\lambda}\colon \overline{T}\,\overline{B} \Rightarrow \overline{B}\,\overline{T}$ sitting above λ. Then $\coprod_\alpha \circ \overline{T}$ is $\overline{B}_\delta$-compatible.*

Proof. The desired natural transformation is formed by composing basic pieces:

$$
\begin{tikzcd}
\mathcal{E}_X \arrow[r, "\overline{B}"] \arrow[ddd, equal] & \mathcal{E}_{BX} \arrow[rr, "\delta^*"] \arrow[drr, "\overline{T}"'] & & \mathcal{E}_X \arrow[r, "\overline{T}"] \arrow[d, phantom, "\Downarrow(d)"] & & \mathcal{E}_{TX} \arrow[r, "\coprod_\alpha"] & \mathcal{E}_X \arrow[ddd, equal] \\
& {} \arrow[d, phantom, "\Downarrow(b)"] & & \mathcal{E}_{TBX} \arrow[urr, "(T\delta)^*"'] \arrow[dr, "\coprod_{\lambda_X}"] \arrow[d, phantom, "\Downarrow(a)"] & & {} \arrow[d, phantom, "\Downarrow(c)"] & \\
& {} & \mathcal{E}_{BTX} \arrow[ur, "\lambda_X^*"] \arrow[rr, equal] & {} \arrow[d, phantom, "\Downarrow(e)"] & \mathcal{E}_{BTX} \arrow[dr, "\coprod_{B\alpha}"] & {} & \\
\mathcal{E}_X \arrow[r, "\overline{T}"'] & \mathcal{E}_{TX} \arrow[ur, "\overline{B}"] \arrow[rr, "\coprod_\alpha"'] & & \mathcal{E}_X \arrow[rr, "\overline{B}"'] & & \mathcal{E}_{BX} \arrow[r, "\delta^*"'] & \mathcal{E}_X
\end{tikzcd}
$$

The pieces (natural transformations) are obtained as follows:

(a) This is the counit of the adjunction $\coprod_{\lambda_X} \dashv \lambda_X^*$.

(b) $\overline{\lambda}$ is a lifting of λ, see Lemma 5.2.6.

(c) (X, α, δ) is a bialgebra, which implies that $(B\alpha \circ \lambda_X \circ T\delta)^* = (\delta \circ \alpha)^*$ and thus there is a natural isomorphism

$$(T\delta)^* \circ \lambda_X^* \circ (B\alpha)^* \cong \alpha^* \circ \delta^* \,. \tag{5.4}$$

The desired natural transformation (b) is defined from (5.4):

$$
\begin{aligned}
&\textstyle\coprod_\alpha \circ (T\delta)^* \Rightarrow \coprod_\alpha \circ (T\delta)^* \circ \lambda_X^* \circ (B\alpha)^* \circ \coprod_{B\alpha} \circ \coprod_{\lambda_X} \\
&\qquad\qquad\qquad\qquad \Big\Downarrow \text{(5.4)} \\
&\qquad\qquad\textstyle\coprod_\alpha \circ \alpha^* \circ \delta^* \circ \coprod_{B\alpha} \circ \coprod_{\lambda_X} \Longrightarrow \delta^* \circ \coprod_{B\alpha} \circ \coprod_{\lambda_X}
\end{aligned}
$$

using the unit of the composite adjunction $\coprod_{B\alpha} \circ \coprod_{\lambda_X} \dashv \lambda_X^* \circ (B\alpha)^*$ and the counit of $\coprod_\alpha \dashv \alpha^*$.

(d) This is an instance of Lemma 3.3.4, using that $\overline{T}$ is a lifting of T.

(e) This is an instance of Lemma 3.3.4, using that $\overline{B}$ is a lifting of B. □

The canonical relation lifting $\mathsf{Rel}(-)$ of a Set functor preserves natural transformations [Jac12, Exercise 4.4.6]. Therefore, if $\overline{T}$ and $\overline{B}$ are instantiated to $\mathsf{Rel}(T)$ and $\mathsf{Rel}(B)$ respectively, then the condition that there exists a $\overline{\lambda}$ above λ is satisfied. Thus we obtain the b_δ-compatibility of the contextual closure (Theorem 4.4.7) as a special case of Theorem 5.2.7.

In order to apply Theorem 5.2.7 for situations when either $\overline{T}$ or $\overline{B}$ is not the canonical relation lifting, one has to exhibit a $\overline{\lambda}$ sitting above λ. In Rel, such a $\overline{\lambda}$ exists if and only if for all relations $R \subseteq X^2$, the restriction of $\lambda_X \times \lambda_X$ to $\overline{T}\,\overline{B}R$ corestricts to $\overline{B}\,\overline{T}R$:

$$(\lambda_X \times \lambda_X)(\overline{T}\,\overline{B}(R)) \subseteq \overline{B}\,\overline{T}(R)\,.$$

A similar condition has to be checked for $\mathsf{Pred} \to \mathsf{Set}$. In Section 5.3 we consider several examples for which we check the above condition.

Abstract GSOS

Recall from Section 3.5.2 that an *abstract GSOS specification* is a natural transformation of the form $\rho\colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$, where Σ^* is the free monad for $\Sigma\colon \mathcal{A} \to \mathcal{A}$ (the $(-)^*$ notation is used both to denote reindexing functors of morphisms in $\mathcal{A}$ and to denote free monads of endofunctors, but the distinction should be clear). Any such specification induces a distributive law $\rho^\dagger\colon \Sigma^*(B \times \mathsf{Id}) \Rightarrow (B \times \mathsf{Id})\Sigma^*$.

To prove compatibility of the contextual closure for bialgebras for a distributive law $\rho^\dagger$ generated from an abstract GSOS specification, one could exhibit a natural transformation $\overline{\rho^\dagger}\colon \overline{\Sigma^*}(\overline{B} \times \overline{\mathsf{Id}}) \Rightarrow (\overline{B} \times \overline{\mathsf{Id}})\overline{\Sigma^*}$ above $\rho^\dagger$ directly, and then apply Theorem 5.2.7. We next show how to simplify such a task by proving that, under mild additional conditions, it suffices to show that there exists $\overline{\rho}\colon \overline{\Sigma}(\overline{B} \times \overline{\mathsf{Id}}) \Rightarrow \overline{B}\,\overline{\Sigma^*}$ above ρ. The lifting of $\overline{\Sigma^*}$ here is induced by the given lifting of $\overline{\Sigma}$; the functor $\overline{\mathsf{Id}}$ lifts the identity (it does not need to be the identity itself), and will be subject to a condition involving $\overline{\Sigma}$.

The construction of $\overline{\rho^\dagger}$ from $\overline{\rho}$ is similar to the construction of $\rho^\dagger$ from ρ. In order to show that it is a lifting, we need some properties relating algebras in the total category $\mathcal{E}$ to those in the base category $\mathcal{A}$.

Lemma 5.2.8. *Consider a lifting $\overline{\Sigma}$ of an $\mathcal{A}$-endofunctor Σ and assume $\overline{\Sigma}$ has free algebras.*

1. *The functor $p\colon \mathcal{E} \to \mathcal{A}$ has a right adjoint $\mathbf{1}\colon \mathcal{A} \to \mathcal{E}$, and this adjunction lifts*

as follows:

$$\begin{tikzcd} \overline{\Sigma}\text{-alg} \arrow[r, bend left, "\overline{p}"] \arrow[r, phantom, "\perp"] \arrow[d] & \Sigma\text{-alg} \arrow[l, bend left, "\overline{\mathbf{1}}"] \arrow[d] \\ \mathcal{E} \arrow[r, bend left, "p"] \arrow[r, phantom, "\perp"] & \mathcal{A} \arrow[l, bend left, "\mathbf{1}"] \end{tikzcd}$$

2. *The functor $\overline{p}$ preserves initial algebras.*

3. *When $P \in \mathcal{E}_X$ for some X in $\mathcal{A}$, the functor $\overline{p}$ maps the free $\overline{\Sigma}$-algebra for P to the free Σ-algebra for X.*

4. *The free monad $\overline{\Sigma}^*$ over $\overline{\Sigma}$ exists and is a lifting of the free monad Σ^* over Σ.*

Proof. 1. By assumption, the fibration considered here has fibred finite products, so one can define $\mathbf{1}(X)$ as the terminal object 1_X in $\mathcal{E}_X$, and $\mathbf{1}(f\colon X \to Y)$ as the Cartesian lifting $\overline{f}_{1_Y}\colon (1_Y)^* \to 1_Y$ which is well-defined since the p preserves terminal objects by assumption; thus $(1_Y)^* = 1_X$.

The functor $\overline{p}$ maps an algebra $\alpha\colon \overline{\Sigma}P \to P$ to $p(\alpha)\colon p(\overline{\Sigma}(P)) \to p(P)$ which is indeed a Σ-algebra since $\overline{\Sigma}$ lifts Σ, i.e., $\Sigma p(P) = p(\overline{\Sigma}(P))$. The existence of a right adjoint $\overline{\mathbf{1}}$ to $\overline{p}$ is a consequence of [HJ98, Theorem 2.14].

2. Since $\overline{p}$ is a left adjoint, it preserves initial objects.

3. This follows from item 2 applied to the lifting $\overline{\Sigma} + P$ of $\Sigma + X$.

4. This is a consequence of item 3. □

Lemma 5.2.8 allows us to prove the desired result on lifting distributive laws induced by GSOS specifications. Rather than assuming that $\overline{\mathsf{Id}}$ is itself the identity (so that the lifted natural transformation is itself an abstract GSOS specification), we assume that $\overline{\mathsf{Id}}$ is a lifting that comes together with a natural transformation $\gamma\colon \overline{\Sigma}\,\overline{\mathsf{Id}} \Rightarrow \overline{\mathsf{Id}}\,\overline{\Sigma}$. We shall apply this result in Section 5.4.1, involving the relation fibration, where $\overline{\mathsf{Id}}$ maps any relation $R \subseteq X \times X$ to the diagonal Δ_X.

Theorem 5.2.9. *Suppose:*

- $\overline{\Sigma}$ *is a lifting of an $\mathcal{A}$-endofunctor Σ;*
- $\overline{\Sigma}$ *has free algebras;*
- $\overline{\mathsf{Id}}$ *is a lifting of the identity functor;*
- *there is a natural transformation $\gamma\colon \overline{\Sigma}\,\overline{\mathsf{Id}} \Rightarrow \overline{\mathsf{Id}}\,\overline{\Sigma}$ that sits above the identity;*
- *there is a natural transformation $\overline{\rho}\colon \overline{\Sigma}(\overline{B} \times \overline{\mathsf{Id}}) \Rightarrow \overline{B\Sigma}^*$ above $\rho\colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$, where $\overline{\Sigma}^*$ is the lifting of Σ^* induced by $\overline{\Sigma}$ as in Lemma 5.2.8.*

Then there is a natural transformation $\overline{\rho^\dagger}\colon \overline{\Sigma}^*\,(\overline{B}\times\overline{\mathsf{Id}}) \Rightarrow (\overline{B}\times\overline{\mathsf{Id}})\overline{\Sigma}^*$ *that sits above* $\rho^\dagger$.

Proof. The idea of the proof is to construct $\overline{\rho^\dagger}$ from the given natural transformation $\overline{\rho}$, by initiality, similar to the construction of a distributive law from a GSOS law (in this case, $\overline{\rho}$ is not a GSOS law in general since $\overline{\mathsf{Id}}$ does not need to be the identity functor in $\mathcal{E}$). Using Lemma 5.2.8 we can then show that this resulting distributive law (between functors) sits above $\rho^\dagger$.

For an object X in $\mathcal{A}$, we know that $\Sigma^* X$ is the free Σ-algebra on X. Let

$$[\kappa_X, \eta_X]\colon \Sigma\Sigma^* X + X \to \Sigma^* X$$

denote the initial $\Sigma + X$-algebra. Similarly, let

$$[\overline{\kappa}_P, \overline{\eta}_P]\colon \overline{\Sigma}\,\overline{\Sigma}^* P + P \to \overline{\Sigma}^* P$$

denote the initial $\overline{\Sigma} + P$-algebra, where P is in $\mathcal{E}_X$. By Lemma 5.2.8 we know that $[\overline{\kappa}_P, \overline{\eta}_P]$ is above $[\kappa_X, \eta_X]$.

For $P \in \mathcal{E}_X$ the map $\overline{\rho}^\dagger_P$ is defined similarly to the construction of $\rho^\dagger_X$ from ρ_X (see (3.15) in Section 3.5.2); the difference is that it involves the natural transformation $\gamma\colon \overline{\Sigma}\,\overline{\mathsf{Id}} \Rightarrow \overline{\mathsf{Id}}\,\overline{\Sigma}$. Indeed, $\overline{\rho}^\dagger_P$ is the unique map arising from initiality:

$$\begin{tikzcd}
\overline{\Sigma}\,\overline{\Sigma}^*(\overline{B}\times\overline{\mathsf{Id}})P \arrow[r, "\overline{\Sigma}(\overline{\rho}^\dagger_P)"] \arrow[ddd, "\overline{\kappa}_{(\overline{B}\times\overline{\mathsf{Id}})P}"'] & \overline{\Sigma}(\overline{B}\times\overline{\mathsf{Id}})\overline{\Sigma}^* P \arrow[d, "\langle \overline{\rho}_{\overline{\Sigma}^* P}, \overline{\Sigma}\pi_2\rangle"] \\
& \overline{B}\,\overline{\Sigma}^*\,\overline{\Sigma}^* P \times \overline{\Sigma}\,\overline{\mathsf{Id}}\,\overline{\Sigma}^* P \arrow[d, "\mathsf{id}\times\gamma_{\overline{\Sigma}^* P}"] \\
& \overline{B}\,\overline{\Sigma}^*\,\overline{\Sigma}^* P \times \overline{\mathsf{Id}}\,\overline{\Sigma}\,\overline{\Sigma}^* P \arrow[d, "\overline{B}\overline{\mu}_P\times\overline{\mathsf{Id}}\overline{\kappa}_P"] \\
\overline{\Sigma}^*(\overline{B}\times\overline{\mathsf{Id}})P \arrow[r, dashed, "\overline{\rho}^\dagger_P"] & (\overline{B}\times\overline{\mathsf{Id}})\overline{\Sigma}^* P \\
(\overline{B}\times\overline{\mathsf{Id}})P \arrow[u, "\overline{\eta}_{(\overline{B}\times\overline{\mathsf{Id}})P}"] \arrow[ur, "(\overline{B}\times\overline{\mathsf{Id}})\overline{\eta}_P"'] &
\end{tikzcd} \tag{5.5}$$

By Lemma 5.2.8, and using that γ sits above the identity, we have that the $\overline{\Sigma} + (\overline{B}\times\overline{\mathsf{Id}})P$-algebras in the above diagram (5.5) sit above the $\Sigma + (B\times\mathsf{Id})X$-algebras defining $\rho^\dagger_X$ from ρ_X. By uniqueness of $\rho^\dagger_X$ it follows that $\overline{\rho}^\dagger_P$ sits above $\rho^\dagger_X$. $\square$

For a ρ-model (X, α, δ), the existence of $\overline{\rho^\dagger}$ above ρ ensures, via the above result and Theorem 5.2.7, compatibility of the contextual closure on the bialgebra $(X, \widehat{\alpha}, \langle\delta, \mathsf{id}\rangle)$ corresponding to the ρ-model. More precisely, it shows that $\coprod_{\widehat{\alpha}} \circ \overline{\Sigma^*}_X$

is $(\overline{B} \times \overline{\mathsf{Id}})_{\langle \delta, \mathsf{id} \rangle}$-compatible. In the remainder of this section, we address two technical issues regarding this result, which arise due to the fact that we present distributive laws by abstract GSOS specifications.

First, the above results provide compatibility for a contextual closure defined based on the free monad $\overline{\Sigma}^*$ rather than the lifted functor $\overline{\Sigma}$ itself, which is the one supplied in concrete examples. However, it turns out that the contextual closure defined by $\overline{\Sigma}$ is, in fibrations whose fibres are preorders, below the one defined by $\overline{\Sigma}^*$ (shown below in Lemma 5.2.10), so if the latter is compatible, the former is sound. Moreover, if the lifting $\overline{\Sigma}$ is given by a modality, then the lifting $\overline{\Sigma}^*$ is given in terms of the inductive extension of this modality (Lemma 5.2.11).

Second, $\overline{B} \times \overline{\mathsf{Id}}_{\langle \delta, \mathsf{id} \rangle}$-compatibility is not exactly $\overline{B}_\delta$-compatibility (the same phenomenon was discussed at a more concrete level at the end of Section 4.4.2). However, under some assumptions, any $\overline{B}_\delta$-invariant is also a $(\overline{B} \times \overline{\mathsf{Id}})_{\langle \delta, \mathsf{id} \rangle}$-invariant (shown below in Lemma 5.2.12).

Lemma 5.2.10. *Let $\Sigma, \overline{\Sigma}$, $\overline{\Sigma}^*$ and Σ^* be as in Lemma 5.2.8. Given an algebra $\alpha\colon \Sigma A \to A$ with induced algebra $\widehat{\alpha}\colon \Sigma^* A \to A$ for the monad Σ^*, there exists a natural transformation of type $\coprod_\alpha \circ \overline{\Sigma} \Rightarrow \coprod_{\widehat{\alpha}} \circ \overline{\Sigma}^*$.*

Proof. Let $\eta\colon \mathsf{Id} \Rightarrow \Sigma^*$ and $\kappa\colon \Sigma\Sigma^* \Rightarrow \Sigma^*$ be the canonical natural transformations defined by initiality; composing them yields a natural transformation $\iota\colon \Sigma \Rightarrow \Sigma^*$. Similarly, we can construct a natural transformation $\overline{\iota}\colon \overline{\Sigma} \Rightarrow \overline{\Sigma}^*$ sitting above ι.

The desired natural transformation consists of two pieces:

$$\begin{tikzcd}
\mathcal{E}_X \arrow[r, "\overline{\Sigma}"] \arrow[d, equal] \arrow[dr, phantom, "\Downarrow(a)"] & \mathcal{E}_{\Sigma X} \arrow[r, "\coprod_\alpha"] \arrow[d, "\coprod_{\iota_X}"] \arrow[dr, phantom, "\Downarrow(b)"] & \mathcal{E}_X \arrow[d, equal] \\
\mathcal{E}_X \arrow[r, "\overline{\Sigma}^*"'] & \mathcal{E}_{\Sigma^* X} \arrow[r, "\coprod_{\widehat{\alpha}}"'] & \mathcal{E}_X
\end{tikzcd}$$

(a) Since $\overline{\iota}$ sits above ι, by Lemma 5.2.6 there is a natural transformation $\theta\colon \overline{\Sigma} \Rightarrow \iota_X^* \circ \overline{\Sigma}^*$. The natural transformation for (a) is its mate:

$$\coprod_{\iota_X} \circ \overline{\Sigma} \Longrightarrow \coprod_{\iota_X} \circ \iota_X^* \circ \overline{\Sigma}^* \Longrightarrow \overline{\Sigma}^*$$

using the counit of $\coprod_{\iota_X} \dashv \iota_X^*$.

(b) We have $\alpha = \widehat{\alpha} \circ \iota_X$, so $\coprod_\alpha = \coprod_{\widehat{\alpha} \circ \iota_X} \cong \coprod_{\widehat{\alpha}} \circ \coprod_{\iota_X}$. □

Lemma 5.2.11. *Suppose $\overline{\Sigma}$: $\mathsf{Pred} \to \mathsf{Pred}$ is a lifting of Σ: $\mathsf{Set} \to \mathsf{Set}$, given by a modality $n\colon \Sigma 2 \to 2$ (see the end of Section 5.2.1), and suppose $\overline{\Sigma}$ has free algebras. Then the lifting $\overline{\Sigma^*}$ of the free monad Σ^* (Lemma 5.2.8) is given by the modality $\widehat{n}\colon \Sigma^* 2 \to 2$.*

Proof. The lifting $\overline{\Sigma^*}$ of the free monad is itself a free monad $\overline{\Sigma}^*$, for $\overline{\Sigma}$ (see Lemma 5.2.8). We need to show that, for any $p\colon X \to 2$: $\overline{\Sigma}^* p = \widehat{n} \circ \Sigma^* p$.

First, observe that $\overline{\Sigma}^* p$ is the initial $\overline{\Sigma} + p$-algebra. By Lemma 5.2.8 it sits above the initial $\Sigma + X$-algebra $[\kappa_X, \eta_X]\colon \Sigma\Sigma X^* + X \to \Sigma^* X$. Let $q\colon \Sigma^* X \to 2$ be the carrier of the initial $\overline{\Sigma} + p$-algebra; then by definition of $\overline{\Sigma}$ and morphisms in Pred it makes the following diagram commute laxly:

$$\begin{tikzcd}
\Sigma\Sigma^* X + X \arrow[r, "{[\kappa_X, \eta_X]}"] \arrow[d, "\Sigma q + \mathsf{id}"'] & \Sigma^* X \arrow[dd, "q"] \\
\Sigma 2 + X \arrow[d, "{[n,p]}"'] \arrow[r, phantom, "\leq"] & \\
2 \arrow[r, equal] & 2
\end{tikzcd}$$

Since the initial algebra is an isomorphism, this is actually strict commutativity. Thus, we have a Σ-algebra morphism:

$$\begin{tikzcd}
\Sigma\Sigma^* X + X \arrow[r, "\Sigma q + \mathsf{id}"] \arrow[d, "{[\kappa_X, \eta_X]}"'] & \Sigma 2 + X \arrow[d, "{[n,p]}"] \\
\Sigma^* X \arrow[r, "q"'] & 2
\end{tikzcd}$$

But this is the unique Σ-algebra morphism from the initial algebra, so if we can prove that filling in $\widehat{n} \circ \Sigma^* p$ for q makes the above diagram commute, then we are done. Indeed, this follows from the commutativity of:

$$\begin{tikzcd}
\Sigma\Sigma^* X + X \arrow[r, "\Sigma\Sigma^* p + \mathsf{id}"] \arrow[d, "{[\kappa_X, \eta_X]}"'] & \Sigma\Sigma^* 2 + X \arrow[r, "\Sigma\widehat{n} + \mathsf{id}"] \arrow[d, "{[\kappa_2, \eta_2 \circ p]}"] & \Sigma 2 + X \arrow[d, "{[n,p]}"] \\
\Sigma^* X \arrow[r, "\Sigma^* p"'] & \Sigma^* 2 \arrow[r, "\widehat{n}"'] & 2
\end{tikzcd}$$

The left-hand square commutes by naturality of κ and the definition of Σ^* on morphisms, the right-hand square commutes by definition of $\widehat{n}$. $\square$

Lemma 5.2.12. *Suppose G is an $\mathcal{E}$-endofunctor such that there exists a natural transformation $\eta\colon \mathsf{Id} \Rightarrow \overline{\mathsf{Id}} \circ G$ that sits above the identity. If R is a $\overline{B}_\delta$-invariant up to G then it is a $(\overline{B} \times \overline{\mathsf{Id}})_{\langle \delta, \mathsf{id}\rangle}$-invariant up to G.*

Proof. Given $R \to \delta^* \overline{B} GR$ and the natural transformation η we construct a morphism h using the universal property of the product $(\overline{B} \times \overline{\mathsf{Id}})(GR) = \overline{B}GR \times \overline{\mathsf{Id}}GR$:

$$\begin{tikzcd}
\delta^*(\overline{B}GR) \arrow[d, "\widetilde{\delta}_{\overline{B}GR}"'] & R \arrow[l] \arrow[d, dashed, "h"] \arrow[dr, "\eta_R"] & \\
\overline{B}GR & (\overline{B} \times \overline{\mathsf{Id}})GR \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & \overline{\mathsf{Id}}GR
\end{tikzcd}$$

The morphism h sits above $\langle \delta, \mathsf{id} \rangle$ (using that η sits above the identity). Thus we can use a Cartesian lifting of $\langle \delta, \mathsf{id} \rangle$ to get the desired invariant:

$$
\begin{tikzcd}
R \arrow[d, dashed] \arrow[dr, "h"] & \\
\langle \delta, \mathsf{id} \rangle^*((\overline{B} \times \overline{\mathsf{Id}})(GR)) \arrow[r, "\widetilde{\langle \delta, \mathsf{id} \rangle}_{(\overline{B} \times \overline{\mathsf{Id}})(GR)}"'] & (\overline{B} \times \overline{\mathsf{Id}})(GR) \\
X \arrow[r, "\langle \delta, \mathsf{id} \rangle"] & BX \times X
\end{tikzcd}
$$

□

If $\mathcal{A} = \mathsf{Rel}$ or $\mathcal{A} = \mathsf{Pred}$, then the existence of η means that $R \subseteq \overline{\mathsf{Id}} \circ G(R)$. A special case is when $\overline{\mathsf{Id}}$ is itself the identity and G is pointed; this holds, for instance, if G is the (canonical) contextual closure of a monad with respect to an algebra for that monad, see the end of Section 4.4.2.

5.3 Examples

We give examples of up-to techniques for several coinductive predicates, and prove their soundness by instantiating the results of Section 5.2.

5.3.1 Weighted language inclusion

Consider the Set functor $BX = \mathbb{S} \times X^A$, where $\mathbb{S}$ is a semiring. Recall that a B-coalgebra is a Moore automaton with output in $\mathbb{S}$, and that the final semantics assigns to every state a weighted language, i.e., a function in $\mathbb{S}^{A^*}$ (Example 3.1.1).

Suppose $\mathbb{S}$ carries a partial order $\leq$. This can be extended pointwise to an order on weighted languages. For instance, if $\mathbb{S}$ is a two-element set of truth values then this order corresponds to plain language inclusion.

To obtain such a notion of inclusion as a coinductive predicate on any B-coalgebra, we define a lifting $\overline{B} \colon \mathsf{Rel} \to \mathsf{Rel}$ of B that maps a relation R on X to a relation on $\mathbb{S} \times X^A$:

$$\overline{B}(R) = \{((p, \varphi), (q, \psi)) \mid p \leq q \text{ and } \forall a \in A. \, (\varphi(a), \psi(a)) \in R\} \, . \tag{5.6}$$

Given a coalgebra $\langle o, t \rangle \colon X \to \mathbb{S} \times X^A$, a relation $R \subseteq X \times X$ is a $\overline{B}_{\langle o,t \rangle}$-invariant iff for every pair $(x, y) \in R$: $o(x) \leq o(y)$ and for all $a \in A$: $(t(x), t(y)) \in R$. Notice that this generalizes simulation of deterministic automata (Definition 2.4.1, Example 3.1.2). The coinductive predicate defined by $\overline{B}_{\langle o,t \rangle}$, that is, the carrier of the final $\overline{B}_{\langle o,t \rangle}$-coalgebra, is the largest $\overline{B}_{\langle o,t \rangle}$-invariant. We call it *inclusion*, and denote it by $\precsim$. Thus, to prove that $x \precsim y$ it suffices to construct a $\overline{B}_{\langle o,t \rangle}$-invariant that contains (x, y).

Let $\langle o, t\rangle \colon X \to \mathbb{S} \times (\mathcal{M}X)^A$ be a weighted automaton (Example 3.1.1). Determinizing it yields a Moore automaton $\langle o^\sharp, t^\sharp\rangle \colon \mathcal{M}X \to \mathbb{S} \times (\mathcal{M}X)^A$, where the final semantics of a state x (viewed as a linear combination) is precisely the weighted language accepted by x on the original automaton (Example 3.5.2). Indeed, given states x and y, we have $x \precsim y$ if the weighted language accepted by x is (pointwise) less than the language accepted by y, and proving $x \precsim y$ amounts to exhibiting a $\overline{B}_{\langle o^\sharp, t^\sharp\rangle}$-invariant that contains (x, y).

As an example, consider the following weighted automaton, where $\mathbb{S} = \mathbb{R}^+$ is the semiring of non-negative real numbers and A is the singleton $\{a\}$:

$$\begin{tikzcd} x \downarrow 0 \arrow[r, bend left, "{a,1}"] & y \downarrow 1 \arrow[l, bend left, "{a,1}"] \arrow[loop above, "{a,1}"] \end{tikzcd}$$

Since the alphabet is a singleton, the language semantics assigns a sequence (of zeros and ones) to each state. To show that the semantics of x is pointwise less than that of y (i.e., the sequence generated by x is increasing) one can establish an invariant on the states of the determinized B-coalgebra associated to the above weighted automaton, as follows:

$$\begin{tikzcd} x \downarrow 0 \arrow[r, "a"] \arrow[d, dash, dashed] & y \downarrow 1 \arrow[r, "a"] \arrow[d, dash, dashed] & x{+}y \downarrow 1 \arrow[r, "a"] \arrow[d, dash, dashed] & \cdots \\ y \downarrow 1 \arrow[r, "a"'] & x{+}y \downarrow 1 \arrow[r, "a"'] & x{+}2y \downarrow 2 \arrow[r, "a"'] & \cdots \end{tikzcd} \tag{5.7}$$

where the solid arrows are transitions, and the dashed lines represent the relation. It is straightforward to see that this requires an infinite relation.

Now consider the finite relation $R = \{(x, y), (y, x{+}y)\}$. This is not an invariant, since $x + y$ is not related to $x + 2y$. However, $x + 2y$ is obtained from $x + y$ by substituting x for y and y for $x + y$, which means that $(x + y, x + 2y)$ is in the contextual closure $\mathsf{ctx}(R)$ as defined in Section 4.2, and thus R is an invariant up to ctx. Below, we define ctx properly and show that it is compatible. As a consequence, the relation R suffices to prove that $x \precsim y$.

Consider a determinized weighted automaton $(\mathcal{M}X, \langle o^\sharp, t^\sharp\rangle)$. The associated contextual closure ctx is formally defined by $\mathsf{ctx} = \coprod_{\mu_X} \circ \mathsf{Rel}(\mathcal{M})$, where μ_X is the multiplication of the monad $\mathcal{M}$ (Example 3.4.1). The canonical relation lifting $\mathsf{Rel}(\mathcal{M})$ is given on a relation $R \subseteq X \times X$ by

$$\mathsf{Rel}(\mathcal{M})(R) = \left\{ \left(\sum r_i x_i,\ \sum r_i y_i\right) \mid \forall i.\, (x_i, y_i) \in R \right\} .$$

To prove that ctx is $\overline{B}_\delta$-compatible, recall that $(\mathcal{M}X, \mu_X, \langle o^\sharp, t^\sharp\rangle)$ is a λ-bialgebra for some λ (Chapter 3). Compatibility follows from Theorem 5.2.7, if we show that there is a natural transformation $\overline{\lambda} \colon \mathsf{Rel}(\mathcal{M})\overline{B} \Rightarrow \overline{B}\mathsf{Rel}(\mathcal{M})$ sitting above λ. Concretely, this amounts to proving that

$$(\lambda_X \times \lambda_X)(\mathsf{Rel}(\mathcal{M})(\overline{B}(R))) \subseteq \overline{B}(\mathsf{Rel}(\mathcal{M})(R)) \tag{5.8}$$

for any relation $R \subseteq X \times X$ and any X. First, we compute $\mathsf{Rel}(\mathcal{M})(\overline{B}(R))$:

$$\Big\{\Big(\sum r_i(p_i,\varphi_i), \sum r_i(q_i,\psi_i)\Big) \mid \forall i.\, p_i \le q_i \text{ and } \forall a.\, (\varphi_i(a),\psi_i(a)) \in R\Big\}$$

Applying $\lambda_X \times \lambda_X$ yields a relation on $B\mathcal{M}X$:

$$\Big\{\Big(\Big(\sum r_i \cdot p_i, \lambda a. \sum r_i \cdot \varphi_i(a)\Big), \Big(\sum r_i \cdot q_i, \lambda a. \sum r_i \cdot \psi_i(a)\Big)\Big) \\ \mid \forall i.\, p_i \le q_i \text{ and } \forall a.\, (\varphi_i(a),\psi_i(a)) \in R\Big\}$$

Now we compute $\overline{B}(\mathsf{Rel}(\mathcal{M})(R))$:

$$\Big\{\Big(\Big(p, \lambda a. \sum r_{a,i}x_{a,i}\Big), \Big(q, \lambda a. \sum r_{a,i}y_{a,i}\Big)\Big) \mid p \le q \text{ and } \forall a.\, \forall i.\, (x_{a,i},y_{a,i}) \in R\Big\}$$

It follows that the inclusion (5.8) holds whenever $\sum r_i \cdot p_i \le \sum r_i \cdot q_i$ given that $p_i \le q_i$ for all i. This is the case when for all $n_1, m_1, n_2, m_2 \in \mathbb{S}$ such that $n_1 \le n_2$ and $m_1 \le m_2$, we have

(a) $n_1 + m_1 \le n_2 + m_2$, and

(b) $n_1 \cdot m_1 \le n_1 \cdot m_2$.

These two conditions are satisfied, for instance, in the Boolean semiring or in $\mathbb{R}^+$. Thus, in these cases, the construction of invariants up to ctx is a sound proof technique for inclusion.

The above argument can possibly be reformulated by using the category Pos of posets and monotone functions as a base category rather than Set, since the conditions (a) and (b) assert that addition and multiplication are monotone. We leave this for future work.

Monotone contextual closure

Condition (b) fails for the semiring $\mathbb{R}$ of (all) real numbers. Nevertheless, our framework allows us to prove compatibility for a different up-to technique, based on a variant of contextual closure. The *monotone* contextual closure is obtained as the composition $\coprod_\mu \circ \overline{\mathcal{M}}$ involving the non-canonical lifting of the functor $\mathcal{M}$ (for the semiring $\mathbb{R}$) defined as follows:

$$\overline{\mathcal{M}}(R) = \left\{\Big(\sum r_i x_i, \sum r_i y_i\Big) \mid \forall i. \begin{array}{lcl} r_i \ge 0 & \Rightarrow & (x_i,y_i) \in R \\ r_i < 0 & \Rightarrow & (y_i,x_i) \in R \end{array}\right\}$$

The rule-based inductive characterization of the monotone contextual closure differs from the standard contextual closure (presented in Example 4.2.5) in the rule for scalar multiplication, which now splits into two rules:

$$\frac{v \;\mathsf{ctx}(R)\; w \quad r \ge 0}{r \cdot v \;\mathsf{ctx}(R)\; r \cdot w} \qquad \frac{v \;\mathsf{ctx}(R)\; w \quad r < 0}{r \cdot w \;\mathsf{ctx}(R)\; r \cdot v}$$

To prove that this is compatible, we prove the inclusion

$$(\lambda_X \times \lambda_X)(\overline{\mathcal{M}}(\overline{B}(R))) \subseteq \overline{B}(\overline{\mathcal{M}}(R)) \,. \tag{5.9}$$

We first compute $\overline{\mathcal{M}}(\overline{B}(R))$:

$$\left\{ \left(\sum r_i(p_i, \varphi_i), \sum r_i(q_i, \psi_i) \right) \mid \forall i. \begin{array}{lcl} r_i \geq 0 & \Rightarrow & p_i \leq q_i \text{ and } \forall a. \varphi_i(a) \; R \; \psi_i(a)) \\ r_i < 0 & \Rightarrow & q_i \leq p_i \text{ and } \forall a. \psi_i(a) \; R \; \varphi_i(a)) \end{array} \right\}$$

Then $(\lambda_X \times \lambda_X)(\overline{\mathcal{M}}(\overline{B}(R)))$ is:

$$\Big\{ \Big(\Big(\sum r_i \cdot p_i, \lambda a. \sum r_i \cdot \varphi_i(a) \Big), \Big(\sum r_i \cdot q_i, \lambda a. \sum r_i \cdot \psi_i(a) \Big) \Big) \\ \mid \forall i. \begin{array}{l} r_i \geq 0 \Rightarrow p_i \leq q_i \text{ and } \forall a. (\varphi_i(a), \psi_i(a)) \in R \\ r_i < 0 \Rightarrow q_i \leq p_i \text{ and } \forall a. (\psi_i(a), \varphi_i(a)) \in R \end{array} \Big\}$$

Finally $\overline{B}(\overline{\mathcal{M}}(R))$ is

$$\Big\{ \Big(\Big(p, \lambda a. \sum r_{a,i} x_{a,i} \Big), \Big(q, \lambda a. \sum r_{a,i} y_{a,i} \Big) \Big) \\ \mid p \leq q; \forall a. \forall i. \begin{array}{l} r_{a,i} \geq 0 \Rightarrow (x_{a,i}, y_{a,i}) \in R \\ r_{a,i} < 0 \Rightarrow (y_{a,i}, x_{a,i}) \in R \end{array} \Big\}$$

The desired inclusion (5.9) holds, since $r_i \cdot p_i \leq r_i \cdot q_i$ for all i. The reason is that $p_i \leq q_i$ when $r_i \geq 0$, whereas $q_i \leq p_i$ if $r_i < 0$.

Reflexive and transitive closure

Contextual closure can be combined with reflexive, transitive and symmetric closure to obtain the *congruence* closure (see Section 4.2), which is a useful technique for bisimulation up-to. For the lifting $\overline{B}$ of B (5.6) (with $BX = \mathbb{S} \times X^A$), we can not expect symmetric closure to be compatible, but we can nevertheless prove compatibility of reflexive and transitive closure.

By reflexivity of $\leq$ it follows that $\Delta_{BX} \subseteq \overline{B}(\Delta_X)$, and thus by Corollary 5.2.5 the functor $\mathsf{diag}_X \colon 1 \to \mathsf{Rel}$ is $\overline{B}_\delta$-compatible, i.e., Δ_X is a $\overline{B}_\delta$-invariant (this amounts to the elementary fact that the diagonal on any Moore automaton is a simulation). By Proposition 5.1.3 this implies that the endofunctor on Rel_X that maps any relation to Δ_X is $\overline{B}_\delta$-compatible. For the transitive closure, by transitivity of $\leq$ it follows that $\overline{B}(R) \otimes \overline{B}(S) \subseteq \overline{B}(R \otimes S)$, where $\otimes$ is relational composition. Again by Corollary 5.2.5 we obtain $\overline{B}_\delta$-compatibility of $\otimes_X$, and thus also of the transitive closure.

5.3.2 Divergence of processes

Consider the functor $BX = (\mathcal{P}_\omega X)^A$, where A is a set of labels that contains a distinguished $\tau \in A$. Let $\overline{B} \colon \mathsf{Pred} \to \mathsf{Pred}$ be the predicate lifting for divergence (Example 3.2.6), and recall that a process diverges if it has an infinite outgoing path labelled by τ-actions. In this section, we establish compatibility of the behavioural equivalence closure, and of a variant of the contextual closure.

As a motivating example, consider the processes p and q given by

$$p \xrightarrow{a} p|p \qquad q \xrightarrow{\overline{a}} q$$

where the parallel composition $|$ is defined as usual (Example 3.5.4). To prove that the process $p|q$ diverges, one can establish an invariant containing $p|q$. But this invariant should then contain all states occurring on the infinite path

$$p|q \xrightarrow{\tau} (p|p)|q \xrightarrow{\tau} \dots$$

and thus it needs to contain infinitely many states.

Instead, an informal proof might go as follows: $p|q$ makes a τ-step to the process $(p|p)|q$. But $(p|p)|q$ is bisimilar to $(p|q)|p$, and now we would like to conclude that this suffices, since we have already inspected $p|q$. Formally, this argument corresponds to establishing an invariant up to the composition of the behavioural equivalence closure and a particular type of contextual closure.

More precisely, recall from Section 5.2.1 that the functor bhv closes a given predicate under behaviourally equivalent (i.e., bisimilar) states. Further, we define the *left contextual closure* as

$$\mathsf{ctx}^l(P \subseteq X) = \{(p|x) \mid p \in P,\ x \in X\}\,.$$

Then $P = \{p|q\}$ is a $\overline{B}_\delta$-invariant up to $\mathsf{bhv} \circ \mathsf{ctx}^l$ (where δ is the model). To conclude from this argument that $p|q$ diverges, we need to prove the soundness of $\mathsf{bhv} \circ \mathsf{ctx}^l$. We do this by proving the compatibility of bhv and ctx^l separately.

Observe that the lifting $\overline{B}$ is determined by a modality $m\colon (\mathcal{P}_\omega 2)^A \to 2$ (as in the end of Section 5.2.1). This modality is defined by: $m(f) = 1$ iff $1 \in f(\tau)$. It induces a monotone predicate lifting, so by Corollary 5.2.2, bhv is $\overline{B}_\delta$-compatible on any B-coalgebra δ.

For the contextual closure, we use a functor $\Sigma X = X \times X$ to syntactically represent the composition operator. Let $\rho\colon \Sigma(B \times \mathsf{Id}) \to B\Sigma^*$ be the GSOS specification giving its semantics, and ρ^* the induced distributive law (Example 3.5.4). We define the left contextual closure of a Σ-algebra α as the composite functor $\mathsf{ctx}^l = \coprod_\alpha \circ \overline{\Sigma}$. The lifting $\overline{\Sigma}$ is given by the modality $n\colon \Sigma 2 \to 2$, defined by $n(b, c) = b$.

Using Theorem 5.2.9, we prove the compatibility of the (left) contextual closure $\coprod_{\widehat{\alpha}} \circ \overline{\Sigma}^*$, involving the free monad for $\overline{\Sigma}$ (by Lemma 5.2.10, ctx^l is contained in this contextual closure). The main step is to show that there exists $\overline{\rho}\colon \overline{\Sigma}(\overline{B} \times \mathsf{Id}) \Rightarrow \overline{B}\,\overline{\Sigma^*}$ that sits above ρ (notice that we use the identity functor on the total category Pred as the lifting of the identity functor on Set).

The existence of $\overline{\rho}$ above ρ amounts to the inclusion

$$\rho(\overline{\Sigma}(\overline{B} \times \mathsf{Id})) \subseteq \overline{B}\,\overline{\Sigma^*} \tag{5.10}$$

which can be proved by hand, based on a careful analysis of ρ and the liftings. However, in the present situation, where both $\overline{B}$ and $\overline{\Sigma}$ are given by modalities (m

and n respectively), this condition can be proved in a neater way. Using the definition of the liftings $\overline{B}$ and $\overline{\Sigma}$ in terms of modalities, the inclusion (5.10) amounts to (lax) commutativity of the outside of the following diagram, for any predicate $p\colon X \to 2$:

$$\begin{tikzcd}
\Sigma(BX \times X) \arrow[r, "\rho_X"] \arrow[d, "\Sigma(Bp\times p)"'] & B\Sigma^* X \arrow[d, "B\Sigma^* p"] \\
\Sigma(B2 \times 2) \arrow[r, "\rho_2"] \arrow[d, "\Sigma m \circ \Sigma\pi_1"'] & B\Sigma^* 2 \arrow[d, "B\widehat{n}"] \\
\Sigma 2 \arrow[r, phantom, "\leq"] \arrow[d, "n"'] & B2 \arrow[d, "m"] \\
2 \arrow[r, equal] & 2
\end{tikzcd}$$

(The lifting $\overline{\Sigma^*}$ is given by $\widehat{n}$; this is Lemma 5.2.11.) The upper square commutes by naturality, which means that lax commutativity of the lower square suffices. To see that this requirement is satisfied, let $f, g \in B2 = (\mathcal{P}2)^A$. If $1 \in f(\tau)$ (which is the only situation where $n \circ \Sigma m \circ \Sigma\pi_1((f,x),(g,y)) = 1$) then $1|y \in \rho_2((f,x),(g,y))(\tau)$, which implies that $m \circ Bn \circ \rho_2((f,x),(g,y)) = 1$ holds, as required.

More interestingly, the property that ρ lifts reduces to checking an inclusion that only involves finite sets (given that the set of labels is finite). This suggest that in general, if B and Σ both preserve finite sets and the liftings are presented by modalities, then this property is decidable. We leave a general investigation for future work.

5.4 Compositional predicates

In this section, we describe a way of defining functor liftings by *composing* simpler liftings, using a generalization of relational composition. We show that proving compatibility of the contextual closure for such a composite lifting reduces to proving compatibility for its constituents. We instantiate this to relational composition in the fibration Rel $\to$ Set, and apply it to derive sound up-to techniques for the notion of *similarity*, studied in [HJ04].

Assume a fibration $p\colon \mathcal{E} \to \mathcal{A}$ and a functor $\otimes\colon \mathcal{E} \times_{\mathcal{A}} \mathcal{E} \to \mathcal{E}$ that lifts the identity functor (see Section 5.2.2). Suppose we have two liftings $\overline{B}_1, \overline{B}_2\colon \mathcal{E} \to \mathcal{E}$ of the same functor $B\colon \mathcal{A} \to \mathcal{A}$. One can then define a *composite* lifting

$$\overline{B}_1 \otimes \overline{B}_2 = \otimes \circ \langle \overline{B}_1, \overline{B}_2 \rangle\,. \tag{5.11}$$

Notice that $\overline{B}_1 \otimes \overline{B}_2$ is a lifting of B. This follows from the fact that $\langle \overline{B}_1, \overline{B}_2 \rangle$ lifts B and that $\otimes$ lifts the identity functor.

Let $\overline{T}\colon \mathcal{E} \to \mathcal{E}$ be a lifting of a functor $T\colon \mathcal{A} \to \mathcal{A}$. To obtain the compatibility of the contextual closure for a composite lifting $\overline{B}_1 \otimes \overline{B}_2$ using Theorem 5.2.7, one needs to prove that a distributive law $\lambda\colon TB \Rightarrow BT$ under consideration lifts to a distributive law of $\overline{T}$ over $\overline{B}_1 \otimes \overline{B}_2$. As a consequence of Theorem 5.4.1 below, it suffices to show that there are distributive laws for the two liftings $\overline{B}_1$ and $\overline{B}_2$ separately, both sitting above λ.

This additionally requires a natural transformation $\gamma\colon \overline{T}\otimes \Rightarrow \otimes\overline{T}^2$. Here

$$\overline{T}^2\colon \mathcal{E} \times_{\mathcal{A}} \mathcal{E} \to \mathcal{E} \times_{\mathcal{A}} \mathcal{E}\,,$$

defined by the universal property of the pullback $\mathcal{E} \times_{\mathcal{A}} \mathcal{E}$, is simply the restriction of the functor $\overline{T}^2\colon \mathcal{E} \times \mathcal{E} \to \mathcal{E} \times \mathcal{E}$. If $\overline{T}$ is the canonical relation lifting $\mathsf{Rel}(T)$ and $\otimes$ is relational composition, then the existence of γ in the theorem amounts to the inclusion $\mathsf{Rel}(T)(R \otimes S) \subseteq \mathsf{Rel}(T)(R) \otimes \mathsf{Rel}(T)(S)$, which holds for any Set functor T (Lemma 3.2.4).

Theorem 5.4.1. *Suppose we have*

1. *a lifting $\overline{T}$ of T;*
2. *a natural transformation $\gamma\colon \overline{T}\otimes \Rightarrow \otimes\overline{T}^2$ above $\mathsf{id}\colon T \Rightarrow T$;*
3. *two liftings $\overline{B}_1$ and $\overline{B}_2$ of B;*
4. *two natural transformations $\overline{\lambda}_1\colon \overline{T}\,\overline{B}_1 \Rightarrow \overline{B}_1\overline{T}$ and $\overline{\lambda}_2\colon \overline{T}\,\overline{B}_2 \Rightarrow \overline{B}_2\overline{T}$ sitting above the same $\lambda\colon TB \Rightarrow BT$.*

Then there exists $\overline{\lambda}\colon \overline{T}(\overline{B}_1 \otimes \overline{B}_2) \Rightarrow (\overline{B}_1 \otimes \overline{B}_2)\overline{T}$ above λ.

Proof. Define $\overline{\lambda}$ on a component P in $\mathcal{E}$ as follows:

$$\overline{T}(\overline{B}_1 P \otimes \overline{B}_2 P) \xrightarrow{\gamma_{\langle \overline{B}_1, \overline{B}_2\rangle P}} (\overline{T}\,\overline{B}_1 P) \otimes (\overline{T}\,\overline{B}_2 P) \xrightarrow{(\overline{\lambda}_1)_P \otimes (\overline{\lambda}_2)_P} \overline{B}_1\overline{T}P \otimes \overline{B}_2\,\overline{T}P$$

Notice that $((\overline{\lambda}_1)_P, (\overline{\lambda}_2)_P)$ is indeed a morphism in $\mathcal{E} \times_{\mathcal{A}} \mathcal{E}$ since $\overline{\lambda}_1$ and $\overline{\lambda}_2$ sit above a common λ. Naturality of $\overline{\lambda}$ follows from naturality of $\overline{\lambda}_1$, $\overline{\lambda}_2$ and γ. Finally, $\overline{\lambda}$ sits above λ since γ sits above $\mathsf{id}\colon T \Rightarrow T$ and $\otimes$ is a lifting of the identity functor. $\square$

5.4.1 Simulation up-to

We recall simulations for coalgebras as introduced in [HJ04]. An *ordered* functor is a pair $(B, \sqsubseteq)$ consisting of a functor $B\colon \mathsf{Set} \to \mathsf{Set}$ with a factorization through the category Pre of preorders and monotone maps:

$$\begin{tikzcd} & \mathsf{Pre} \arrow[d] \\ \mathsf{Set} \arrow[ur, "\sqsubseteq"] \arrow[r, "B"'] & \mathsf{Set} \end{tikzcd}$$

Such an ordered functor gives rise to a *constant* relation lifting $\overline{\sqsubseteq}$ of B defined by $\overline{\sqsubseteq}(R \subseteq X \times X) = \sqsubseteq_{BX}$. Then the *lax relation lifting* $\mathsf{Rel}(B)^{\sqsubseteq}$ is defined compositionally by

$$\mathsf{Rel}(B)^{\sqsubseteq} = \overline{\sqsubseteq} \otimes \mathsf{Rel}(B) \otimes \overline{\sqsubseteq}$$

where $\otimes \colon \mathcal{E} \times_{\mathcal{A}} \mathcal{E} \to \mathcal{E}$ is the relational composition functor (using the notation of (5.11) above).

Let $\delta \colon X \to BX$ be a B-coalgebra. A $\mathsf{Rel}(B)^{\sqsubseteq}_{\delta}$-invariant, where $\mathsf{Rel}(B)^{\sqsubseteq}_{\delta}$ abbreviates $\delta^* \circ \mathsf{Rel}(B)^{\sqsubseteq}_{X}$, is called a *simulation*. The coinductive predicate defined by $\mathsf{Rel}(B)^{\sqsubseteq}_{\delta}$ is called *similarity*.

Example 5.4.2. We list a few examples of ordered functors and their associated notion of simulations, and refer to [HJ04] for many more.

1. Let $\mathbb{S}$ be a semiring equipped with a partial order $\leq$. The functor $BX = \mathbb{S} \times X^A$ is ordered, with $\sqsubseteq_{BX}$ defined as $(p, \varphi) \sqsubseteq_{BX} (q, \psi)$ iff $p \leq q$ and $\varphi = \psi$. Then $\mathsf{Rel}(B)^{\sqsubseteq}$ coincides with the lifting $\overline{B}$ defined in Section 5.3.1.

2. The functor $BX = (\mathcal{P}_\omega X)^A$ is ordered by pointwise subset inclusion. In this case, a simulation is the standard notion on transition systems: a relation $R \subseteq X \times X$ such that for all $(x, y) \in R$: if $x \xrightarrow{a} x'$ then there exists y' such that $y \xrightarrow{a} y'$ and $(x', y') \in R$. Given a transition system, similarity is the greatest simulation.

An ordered functor B is called *stable* if $(\mathsf{Rel}(B)^{\sqsubseteq}, B)$ is a fibration map [HJ04]. Since polynomial functors, as well as the one for LTSs, are stable [HJ04], the following results hold for the coalgebras in Example 5.4.2.

Proposition 5.4.3. *If B is a stable ordered functor, then the behavioural equivalence closure* bhv*, the self closure* slf *and the transitive closure* tra *(all defined in Section 5.2.2) are* $\mathsf{Rel}(B)^{\sqsubseteq}_{\delta}$*-compatible.*

Proof. Compatibility of bhv comes from Theorem 5.2.1, which only requires that $(\mathsf{Rel}(B)^{\sqsubseteq}, B)$ is a fibration map. Compatibility of slf and tra comes from Corollary 5.2.5: as shown in [HJ04, Lemma 5.3], stable functors satisfy condition (5.3), i.e., for all relations $R, S \subseteq X^2$: $\mathsf{Rel}(B)^{\sqsubseteq}(R) \otimes \mathsf{Rel}(B)^{\sqsubseteq}(S) \subseteq \mathsf{Rel}(B)^{\sqsubseteq}(R \otimes S)$. ☐

If $BX = (\mathcal{P}_\omega X)^A$ then bhv maps a relation R to $\sim \circ R \circ \sim$ where $\sim$ is bisimilarity, whereas slf maps R to $\lesssim \circ R \circ \lesssim$, where $\lesssim$ is similarity.

We proceed to consider the compatibility of the contextual closure, for which we assume an abstract GSOS specification $\rho \colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$. Such a specification ρ is *monotone* if, for any X, the restriction of $\rho_X \times \rho_X$ to $\mathsf{Rel}(\Sigma)(\sqsubseteq_{BX} \times \Delta_X)$ corestricts to $\sqsubseteq_{B\Sigma^* X}$. If Σ is a polynomial functor representing a signature, then this means that for any operator σ (of arity n) we have

$$\frac{b_1 \sqsubseteq_{BX} c_1 \quad \dots \quad b_n \sqsubseteq_{BX} c_n}{\rho_X(\sigma(\mathbf{b}, \mathbf{x})) \sqsubseteq_{B\Sigma^* X} \rho_X(\sigma(\mathbf{c}, \mathbf{x}))}$$

where $\mathbf{b}, \mathbf{x} = (b_1, x_1), \ldots, (b_n, x_n)$ with $x_i \in X$ and similarly for $\mathbf{c}, \mathbf{x}$. If $\sqsubseteq$ is the order on the functor for LTSs, then monotonicity corresponds to the *positive GSOS* format [FS10], which is GSOS without negative premises. Monotonicity turns out to be precisely the condition needed to apply Theorem 5.2.9.

Proposition 5.4.4. *Let $\rho\colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$ be a monotone abstract GSOS specification and $(X, \alpha, \langle \delta, \mathsf{id}\rangle)$ be a $\rho^\dagger$-bialgebra. Then* $\mathsf{ctx} = \coprod_\alpha \circ \mathsf{Rel}(\Sigma^*)$ *is* $(\mathsf{Rel}(B)^{\sqsubseteq} \times \mathsf{Id})_{\langle \delta, \mathsf{id}\rangle}$*-compatible.*

Proof. To obtain the desired compatibility from Theorem 5.2.7, we need to prove that there exists a distributive law $\overline{\rho^\dagger}$ of $\mathsf{Rel}(\Sigma^*)$ over $\mathsf{Rel}(B)^{\sqsubseteq} \times \mathsf{Id}$, sitting above $\rho^\dagger$.

First, observe that the lifting $\mathsf{Rel}(B)^{\sqsubseteq} \times \mathsf{Id}$ of $B \times \mathsf{Id}$ decomposes as

$$(\sqsubseteq \times \overline{\mathsf{Id}}) \otimes (\mathsf{Rel}(B) \times \mathsf{Id}) \otimes (\sqsubseteq \times \overline{\mathsf{Id}})$$

where $\overline{\mathsf{Id}}$ is the constant functor mapping $R \subseteq X \times X$ to Δ_X. Notice that $\overline{\mathsf{Id}}$ is a lifting of the identity functor (but it is not the identity functor itself).

By Theorem 5.4.1, proving the existence of $\overline{\rho^\dagger}$ above $\rho^\dagger$ reduces to proving that there exist two natural transformations

1. $\overline{\rho^\dagger}_1\colon \mathsf{Rel}(\Sigma^*)(\mathsf{Rel}(B) \times \mathsf{Id}) \Rightarrow (\mathsf{Rel}(B) \times \mathsf{Id})\mathsf{Rel}(\Sigma^*)$, and
2. $\overline{\rho^\dagger}_2\colon \mathsf{Rel}(\Sigma^*)(\sqsubseteq \times \overline{\mathsf{Id}}) \Rightarrow (\sqsubseteq \times \overline{\mathsf{Id}})\mathsf{Rel}(\Sigma^*)$,

both sitting above $\rho^\dagger$. (Notice that since the functor $\overline{T}$ of the theorem is a canonical relation lifting, the required γ exists.)

For item 1, observe that the required natural transformation exists since both functor liftings are canonical; see Section 5.2.3 (below Theorem 5.2.7).

For item 2, the task reduces by Theorem 5.2.9 to showing that there is

$$\overline{\rho}\colon \mathsf{Rel}(\Sigma)(\sqsubseteq \times \overline{\mathsf{Id}}) \Rightarrow \sqsubseteq \circ \mathsf{Rel}(\Sigma^*)$$

above ρ. But this is precisely monotonicity, as introduced above. Further, Theorem 5.2.9 requires that there exists a natural transformation $\gamma\colon \mathsf{Rel}(\Sigma) \circ \overline{\mathsf{Id}} \Rightarrow \overline{\mathsf{Id}} \circ \mathsf{Rel}(\Sigma)$. Since $\overline{\mathsf{Id}}$ is the functor mapping any relation to the diagonal over its carrier, γ exists if $\mathsf{Rel}(\Sigma)(\Delta_X) \subseteq \Delta_{\Sigma X}$, which holds for any Σ (Lemma 3.2.4). Thus, as a consequence of Theorem 5.2.9, we obtain the desired natural transformation.

The existence of $\overline{\rho^\dagger}_1$ and $\overline{\rho^\dagger}_2$ ensures, by Theorem 5.4.1 and Theorem 5.2.7, that ctx is $(\mathsf{Rel}(B)^{\sqsubseteq} \times \mathsf{Id})_{\langle \delta, \mathsf{id}\rangle}$-compatible. □

A direct consequence of this result is that simulation up-to is compatible on any model of a positive GSOS specification.

Further, Theorem 2.4.6 states that simulation up-to (precongruence) for languages is sound whenever the operations under consideration are given by monotone behavioural differential equations. But any such operation can also be expressed in monotone GSOS for the ordered functor $BX = 2 \times X^A$ (see Example 5.4.2). Thus, we obtain the compatibility of the contextual closure by Proposition 5.4.4, and since the reflexive and transitive closure are compatible as well (Section 5.3.1), composing them together yields an alternative proof of Theorem 2.4.6.

5.5 Discussion and related work

We showed how up-to techniques fit into the setting of coinduction in a fibration, yielding a general and modular theory of coinduction up-to. This goes beyond the previous chapter in several ways: first, it allows other predicates than bisimilarity, including other binary predicates but also, e.g., unary predicates. Second, it can be instantiated to different base categories (in [BPPR14] an example of this is given by up-to-congruence for nominal automata).

Bisimulation up-to at the level of coalgebras was studied by Lenisa [Len99, LPW00]. The up-to-context technique for coalgebraic bisimulation was later derived as a special case of so-called λ-coinduction [Bar04]. Combining up-to techniques remained an open problem. In [Luo06], Sangiorgi's framework of up-to techniques [San98] is adapted to prove soundness of several up-to techniques for bisimulation, based on relation lifting and thus strongly related to the development in Chapter 4, but combinations of enhancements are not considered there. Finally [ZLL$^+$10] introduces bisimulation up-to where the notion of bisimulation is based on a specification language for polynomial functors. All of the above works focus on bisimulation, rather than general coinductive predicates.

We conclude with a short, technical summary of the main soundness results of this chapter. The up-to techniques and soundness results are all formulated in terms of a bifibration $p\colon \mathcal{E} \to \mathcal{A}$, a coalgebra $\delta\colon X \to BX$ for a functor $B\colon \mathcal{A} \to \mathcal{A}$ (that models the system of interest) and a lifting $\overline{B}\colon \mathcal{E} \to \mathcal{E}$ of $\mathcal{A}$ (that determines the coinductive predicate of interest). By proving a functor G to be $\overline{B}_\delta$-*compatible*, the construction of invariants up to G is a sound proof technique for the coinductive predicate determined by the lifting $\overline{B}$ on the coalgebra δ. The table below lists the main compatibility results, based on conditions on the functors involved. For ctx_α, we assume an algebra $\alpha\colon TX \to X$ for a functor T with a lifting $\overline{T}$, and a distributive law of the functor T over the functor B.

Name	Notation	Condition $\overline{B}_\delta$-compatibility
Behavioural equivalence	bhv_δ	$(\overline{B}, B)$ is a fibration map
Contextual closure	ctx_α	(X, α, δ) is a λ-bialgebra, and there is a distributive law of $\overline{T}$ over $\overline{B}$ above λ

If $p\colon \mathsf{Rel} \to \mathsf{Set}$ is the relation fibration, then we have the following additional results.

Name	Notation	Condition $\overline{B}_\delta$-compatibility
Diagonal functor	diag	$\Delta_{BX} \subseteq \overline{B}(\Delta_X)$
Inverse functor	inv	$(\overline{B}R)^{op} \subseteq \overline{B}(R^{op})$ for all $R \subseteq X^2$
Relational comp.	$\otimes$	$\overline{B}(R) \otimes \overline{B}(S) \subseteq \overline{B}(R \otimes S)$ for all $R, S \subseteq X^2$
Self closure	slf_δ	$\otimes$ is $\overline{B}_\delta$-compatible
Transitive closure	tra	$\otimes$ is $\overline{B}_\delta$-compatible
Equivalence closure	eq	diag, inv and $\otimes$ are $\overline{B}_\delta$-compatible

While the techniques introduced in this chapter are very general, they are also quite technical and require significant background knowledge to be understood. It would be a worthwile effort to develop natural specification techniques for coinductive predicates, in which compatibility can be established easily, or even automatically. In this chapter we have suggested one approach in this direction: the use of *modalities* to specify coinductive predicates, so that, under suitable assumptions, the required condition for compatibility of the contextual closure is a decidable property. We leave a more extensive investigation for future work.

Chapter 6

Bialgebraic semantics with equations

In this chapter, we focus on structural operational semantics, in the setting of abstract GSOS specifications as introduced by Turi and Plotkin. As explained in Section 3.5 and the introduction, their approach provides a general perspective on well-behaved, compositional calculi and languages, parametric in the type of behaviour and the type of syntax. Moreover, in the previous chapters we have seen that bisimulation up to context is a sound (even compatible) proof technique on models of abstract GSOS specifications.

Given a GSOS specification, the behaviour of terms is computed inductively, which is possible since each operator is defined directly in terms of the behaviour of its arguments. An example of a rule that does not fit the GSOS format is the following:

$$\frac{!x|x \xrightarrow{a} t}{!x \xrightarrow{a} t} \tag{6.1}$$

This rule properly defines the replication operator in CCS[1]: intuitively, $!x$ represents $x|x|x|\ldots$, i.e., the infinite parallel composition of x with itself. In fact, the above rule can be seen as assigning the behaviour of the term $!x|x$ to the simpler term $!x$, therefore we call it an *assignment rule*.

We show how to interpret assignment rules together with abstract GSOS specifications. Our approach is based on the assumption that the functor which represents the type of coalgebra is *ordered* as a complete lattice; for example, for the functor $(\mathcal{P}-)^A$ of labelled transition systems this order is simply pointwise set inclusion. The operational model on closed terms is then defined as the *least* model such that every transition can either can be derived from a rule in the specification or from

[1]The simpler rule $\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} !x|x'}$ is problematic in the presence of the sum operator, since it does not allow to derive τ-transitions from a process such as $!(a.P + \overline{a}.Q)$ [PS12, SW01].

an assignment rule. To ensure the existence of such least models, we disallow negative premises by using *monotone* abstract GSOS specifications, a generalization of the positive GSOS format for transition systems (see Section 5.4.1).

The main result of this chapter is that the interpretation of a monotone abstract GSOS specification together with a set of assignment rules is itself the operational model of another (typically larger) abstract GSOS specification. Like the interpretation of a GSOS specification with assignment rules, we construct this latter specification by fixed point induction. As a direct consequence of this alternative representation of the interpretation, we obtain that bisimilarity is a congruence and that bisimulation up to context is sound and even compatible—properties that do not follow from bisimilarity being a congruence [PS12]. As an example, we obtain the compatibility of bisimulation up to context for CCS with replication, which was shown earlier with an ad-hoc argument (see, e.g., [PS12]).

In the second part of this chapter, we combine *structural congruences* with the bialgebraic framework, using assignment rules. Structural congruences have been widely used in concurrency theory ever since their introduction in the operational semantics of the π-calculus in [Mil92]. The basic idea is that SOS specifications are extended with *equations* $\equiv$ on terms, which are then linked by a special deduction rule:

$$\frac{t \equiv u \qquad u \xrightarrow{a} u' \qquad u' \equiv v}{t \xrightarrow{a} v}$$

This rule essentially states that if two processes are equated by the congruence generated by the set of equations, then they can perform the same transitions. Prototypical examples are the specification of the parallel operator by combining a single rule with commutativity, and the specification of the replication operator by an equation, both shown below:

$$\frac{x \xrightarrow{a} x'}{x|y \xrightarrow{a} x'|y} \quad x|y = y|x \qquad\qquad !x = !x|x \tag{6.2}$$

Even though structural congruences are standard in concurrency theory, a systematic study of their properties was missing until the work of Mousavi and Reniers, who show how to interpret SOS rules with structural congruences in various equivalent ways [MR05]. Mousavi and Reniers exhibit very simple examples of equations and SOS rules for which bisimilarity is not a congruence, even when the SOS rules are in the tyft (or the GSOS) format. As a solution to this problem they introduce a restricted format for equations, called cfsc, for which bisimilarity is a congruence when combined with tyft specifications.

In the current chapter, we show how to interpret structural congruences at the general level of coalgebras, in terms of an operational model on closed terms. We prove that when the equations are in the cfsc format then they can be encoded by assignment rules, in such a way that their respective interpretations coincide up to bisimilarity. Consequently, not only is bisimilarity a congruence for monotone abstract GSOS combined with cfsc equations, but we also obtain the compatibility of bisimulation up to context and bisimilarity. From a technical point of view, structural congruences have not been developed outside the work of Mousavi and Reniers, and have not at all been explored in the theory of bialgebraic semantics [Bar04, Kli07]. Here, we develop the basic theory of monotone abstract GSOS specifications for ordered functors, and use it to obtain a bialgebraic perspective on structural congruences (assuming an ordered behaviour functor).

Outline In Section 6.1, we introduce assignment rules and their interpretation. In Section 6.2, we show that this interpretation can be obtained as the operational model of another abstract GSOS specification. Section 6.3 contains the integration of structural congruence with the bialgebraic framework. In Section 6.4, we conclude and discuss related work.

6.1 Assignment rules

We consider the interpretation of abstract GSOS specifications (without negative premises) together with *assignment rules* of the form

$$\sigma(x_1, \ldots, x_n) := t \tag{6.3}$$

where t is a term over the variables $x_1, \ldots, x_n$. Assignment rules will be interpreted as a kind of rewriting rules: the behaviour of t induces behaviour of $\sigma(x_1, \ldots, x_n)$. An example is the replication operator given in equation (6.1) of the introduction; this can be given by the assignment rule $!x := !x|x$. Notice that assignment rules do not fit directly into the bialgebraic framework, since they are inherently non-structural: they do not satisfy the property of GSOS specifications that the behaviour of terms in the operational model is computed directly from the behaviour of their subterms.

In the case of labelled transition systems, given a GSOS specification and a set of rules of the above form, the desired interpretation is informally as follows (this is formalized below): every transition from a term $\sigma(t_1, \ldots, t_n)$ should either be derived from the transitions of $t_1, \ldots, t_n$ and a rule in the specification, or from an assignment rule which has σ on the left-hand side. However, such an interpretation is not necessarily unique, since there may be infinite inferences caused by the assignment rules. For example, the rule $\sigma(x) := \sigma(x)$ does not have a unique solution. In order to rule out infinite inferences, one is interested in the *least* transition system on closed terms which is a model in the above sense. Such a least model does not necessarily exist in general because of negative premises. Therefore, we will restrict to GSOS specifications without negative premises.

To interpret specifications which involve assignment rules at the general level of a functor $B \colon \mathsf{Set} \to \mathsf{Set}$ one needs a notion of order on B. In the case of labelled transition systems, this order is clear and often left implicit: in that case $BX = (\mathcal{P}X)^A$, and the order is simply the (pointwise) subset order. To allow the desired generalization, we assume that our behaviour functor B is ordered (cf. Section 5.4.1). We will need the existence of fixed points of monotone functions.

To this end, let CJSL be the category of complete (join semi-)lattices and join-preserving functions. We define a CJSL-*ordered functor* to be a functor $B\colon \mathsf{Set} \to \mathsf{Set}$ with a factorization $\sqsubseteq$ through CJSL:

$$\begin{tikzcd} & \mathsf{CJSL} \arrow[d, "U"] \\ \mathsf{Set} \arrow[ur, "\sqsubseteq"] \arrow[r, "B"'] & \mathsf{Set} \end{tikzcd}$$

where U is the forgetful functor. If B is a CJSL-ordered functor, then for any set X, BX is a complete lattice. We denote the join of a set $S \subseteq BX$ in this lattice by $\bigvee S$, and we write $\perp$ for $\bigvee \emptyset$ and $x \leq y$ if $x \vee y = y$, for $x, y \in BX$. Moreover, for any function $f\colon X \to Y$, Bf is join-preserving. Consequently, Bf is also monotone, i.e., for any $x, y \in BX$: $x \leq y$ implies $(Bf)(x) \leq (Bf)(y)$.

Example 6.1.1. The functor $(\mathcal{P}-)^A$ of labelled transition systems is CJSL-ordered, with the order on $(\mathcal{P}X)^A$ given by pointwise subset inclusion.

Example 6.1.2. In Chapter 3 we defined weighted transition systems for a semiring as coalgebras for the functor $(\mathcal{M}-)^A$, where $\mathcal{M}X$ consists of (finite) linear combinations with coefficients in the semiring. Here, we consider weighted transition systems for a *complete monoid* M, i.e., a monoid with an infinitary sum operation consistent with the finite sum [DK09]. These are coalgebras for the functor $(M^-)^A$ where $M^-\colon \mathsf{Set} \to \mathsf{Set}$ is defined as follows:

- For each set X, M^X is the set of functions from X to M.
- For each function $h\colon X \to Y$, $M^h\colon M^X \to M^Y$ is the function mapping each $\varphi \in M^X$ into $\varphi^h \in M^Y$ defined, for all $y \in Y$, by $\varphi^h(y) = \sum_{x' \in h^{-1}(y)} \varphi(x')$.

By taking the Boolean monoid, we retrieve infinitely branching labelled transition systems. As another example, consider the set $\mathbb{R}^+ \cup \{\infty\}$ of positive reals, ordered as usual and extended with a top element ∞. Together with the supremum operation, $\mathbb{R}^+ \cup \{\infty\}$ forms a complete ordered monoid, with 0 as unit. The order on $\mathbb{R}^+ \cup \{\infty\}$ extends to an order on the functor for weighted transition systems over this monoid, where joins are calculated pointwise.

Example 6.1.3. For a non-example: we can try to extend a functor $B\colon \mathsf{Set} \to \mathsf{Set}$ to a CJSL-ordered functor B' by defining $B'X = BX + 2$, putting the discrete order on BX and taking the elements of $2 = \{\top, \perp\}$ to be the top and the bottom element respectively. Contrary to what is stated in [RB14, Example 2], such a functor B' is not CJSL-ordered, in general. Indeed $B'X$ is a complete lattice, but the functor B' is not well-defined on morphisms: given a function f, $B'f$ need not be join-preserving. For instance, if we take $B = \mathsf{Id}$, a set X with two distinct elements $x, y \in X$ and a function $f\colon X \to X$ such that $f(x) = f(y)$, we have $(B'f)(x) \vee (B'f)(y) = (B'f)(x) = f(x) \neq \top$ whereas $(B'f)(x \vee y) = (B'f)(\top) = \top$.

Given arbitrary sets X and Y, the complete lattice on BY lifts pointwise to a complete lattice on functions of type $X \to BY$, i.e., for a collection $\{f_i\}_{i \in I}$ of functions of the form $f_i \colon X \to BY$ we define $(\bigvee\{f_i\}_{i\in I})(x) = \bigvee_{i \in I}(f_i(x))$. This induces in particular a complete lattice on the set of all coalgebras on closed terms over a signature. Given a polynomial functor $\Sigma \colon \mathsf{Set} \to \mathsf{Set}$ corresponding to a signature (Section 3.4), we denote this set by

$$\mathbb{M} = \{f \mid f \colon \Sigma^*\emptyset \to B\Sigma^*\emptyset\}\,. \tag{6.4}$$

The order on B lifts to an order on $B \times \mathsf{Id}$ by defining $(b_1, x_1) \leq (b_2, x_2)$ iff $b_1 \leq b_2$ and $x_1 = x_2$ for $(b_1, x_1), (b_2, x_2) \in BX \times X$. Moreover, given Σ as above, the order lifts componentwise to ΣBX (and also to $\Sigma(BX \times X)$) for any set X, by defining, for any operators σ, τ of arity n and m respectively: $\sigma(k_1, \ldots, k_n) \leq \tau(l_1, \ldots, l_m)$ iff $\sigma = \tau$ (so also $n = m$) and $k_i \leq l_i$ for all $i \leq n$.

Definition 6.1.4. Using the above lifting of the order on B to $\Sigma(B \times \mathsf{Id})$, a specification $\rho \colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$ is said to be *monotone* if all its components are monotone.

Definition 6.1.4 is a special case of monotone abstract GSOS specifications defined in terms of relation lifting, as introduced in Section 5.4.1. For the functor $BX = (\mathcal{P}X)^A$ of labelled transition systems, monotone specifications correspond to specifications in (an infinitary version of) the *positive GSOS* format [FS10].

Assignment rules (6.3) can be formalized in terms of natural transformations, which are independent of the behaviour functor B.

Definition 6.1.5. An *assignment rule* is a natural transformation $d \colon \Sigma \Rightarrow \Sigma^*$.

If there is no intended assignment for an operator $\sigma \in \Sigma$, this is modelled by defining $d_X(\sigma(x_1, \ldots, x_n)) = \sigma(x_1, \ldots, x_n)$ for every X. For example, the assignment rule for the replication operator is the natural transformation that sends $!x$ to $!x|x$ for any x, and is the identity on all other operators in Σ.

Assumption 6.1.6. In the remainder of this chapter, we assume:

1. A CJSL-ordered functor B.
2. A functor Σ defined from a signature (see Section 3.4), with free monad (Σ^*, η, μ).
3. A monotone GSOS specification $\rho \colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$.
4. A set Δ of assignment rules, ranged over by $d \colon \Sigma \Rightarrow \Sigma^*$.

Throughout this chapter we denote by $M(\rho)$ the *operational model* of ρ. As explained in Section 3.5.2, the operational model $M(\rho) \colon \Sigma^*\emptyset \to B\Sigma^*\emptyset$ is the unique coalgebra that makes the following diagram commute:

$$\begin{tikzcd}
\Sigma\Sigma^*\emptyset \arrow[r, "\Sigma\langle M(\rho),\mathsf{id}\rangle"] \arrow[dd, "\kappa_\emptyset"'] & \Sigma(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\rho_{\Sigma^*\emptyset}"] \\
 & B\Sigma^*\Sigma^*\emptyset \arrow[d, "B\mu_\emptyset"] \\
\Sigma^*\emptyset \arrow[r, "M(\rho)"'] & B\Sigma^*\emptyset
\end{tikzcd} \tag{6.5}$$

where $\kappa \colon \Sigma\Sigma^* \Rightarrow \Sigma^*$ is the natural transformation such that, for a component X, the copairing $[\kappa_X, \eta_X]$ is the initial $\Sigma + X$-algebra (Equation (3.11) in Section 3.4). Observe that the operational model is the unique $f \in \mathbb{M}$ (see Equation 6.4) satisfying the equation

$$f \circ \kappa_\emptyset = B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \,.$$

The definition below extends this equation to incorporate assignment rules.

Definition 6.1.7. Let $\psi \colon \mathbb{M} \to \mathbb{M}$ be the (unique) function such that

$$\psi(f) \circ \kappa_\emptyset = B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d \in \Delta} f \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} \,.$$

A (ρ, Δ)-*model* is a coalgebra $f \in \mathbb{M}$ such that $\psi(f) = f$.

The function ψ is indeed uniquely defined, since $\kappa_\emptyset \colon \Sigma\Sigma^*\emptyset \to \Sigma^*\emptyset$ is an initial algebra and therefore an isomorphism. As argued in the beginning of this section, in general there may be more than one model for a fixed ρ and Δ, and we regard the *least* (ρ, Δ)-model to be the intended interpretation. In order to show that a least model exists, we need the following.

Lemma 6.1.8. *The function* $\psi \colon \mathbb{M} \to \mathbb{M}$ *is monotone.*

Proof. Let $f, g \in \mathbb{M}$ with $f \leq g$. By monotonicity of ρ, we have $\rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq \rho_{\Sigma^*\emptyset} \circ \Sigma\langle g, \mathsf{id}\rangle$, and since $B\mu_\emptyset$ is monotone then $B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle g, \mathsf{id}\rangle$. It follows that $\psi(f) \circ \kappa_\emptyset \leq \psi(g) \circ \kappa_\emptyset$ and thus also $\psi(f) \leq \psi(g)$ because $\kappa_\emptyset$ is an isomorphism. $\square$

Since ψ is monotone and $\mathbb{M}$ is a complete lattice, by the Knaster-Tarski theorem ψ has a least fixed point.

Definition 6.1.9. The *interpretation* of ρ and Δ is the least (ρ, Δ)-model, i.e., $\mathsf{lfp}(\psi)$.

Example 6.1.10. For a GSOS specification together with assignment rules, the interpretation is the least transition system on closed terms so that $\sigma(t_1, \ldots, t_n) \xrightarrow{a} t'$ if and only if:

1. it can be obtained by instantiating a rule in the specification, or
2. there is an assignment of t to σ, and $t \xrightarrow{a} t'$.

This is a recursive definition; being the least such transition system has the desired consequence that every derivation of a transition $t \xrightarrow{a} t'$ is finite.

6.2 Integrating assignment rules in abstract GSOS

In the previous section, we have seen how to interpret a monotone abstract GSOS specification ρ together with a set of assignment rules Δ as a coalgebra on closed terms. In this section, we show that we can alternatively construct this coalgebra as the operational model of another specification (without assignment rules), which is constructed as the least fixed point of a function on the complete lattice of specifications. The consequence of this alternative representation is that the well-behavedness properties of the operational model of a specification, such as bisimilarity being a congruence and the compatibility of bisimulation up to context, carry over to the interpretation of ρ and Δ.

Let $\mathbb{G}$ be the set of all monotone abstract GSOS specifications of Σ over B (Definition 6.1.4). We turn $\mathbb{G}$ into a complete lattice by defining the order componentwise, i.e., for any $L \subseteq \mathbb{G}$ and any set X: $(\bigvee L)_X = \bigvee_{\rho \in L} \rho_X$. The join is well-defined:

Lemma 6.2.1. *For any $L \subseteq \mathbb{G}$: the family of functions $\bigvee L$ as defined above is a monotone specification.*

Proof. Let $f \colon X \to Y$ be a function. For any $k \in \Sigma(BX \times X)$:

$$\begin{aligned}
B\Sigma^* f \circ (\textstyle\bigvee L)_X(k) &= B\Sigma^* f \circ (\textstyle\bigvee_{\rho \in L}(\rho_X(k))) && \text{definition of } \textstyle\bigvee L \\
&= \textstyle\bigvee_{\rho \in L}(B\Sigma^* f \circ \rho_X(k)) && B\Sigma^* f \text{ is join-preserving} \\
&= \textstyle\bigvee_{\rho \in L}(\rho_Y \circ \Sigma(Bf \times f)(k)) && \text{naturality of } \rho \\
&= (\textstyle\bigvee L)_Y(\Sigma(Bf \times f)(k)) && \text{definition of } \textstyle\bigvee L
\end{aligned}$$

which proves naturality. Monotonicity is straightforward as well. □

The lattice structure of $\mathbb{G}$ provides a way of combining specifications. Consider, for an assignment rule $d \in \Delta$ and specification τ, the following natural transformation:

$$\Sigma(B \times \mathsf{Id}) \xrightarrow{d_{B \times \mathsf{Id}}} \Sigma^*(B \times \mathsf{Id}) \xrightarrow{\tau^\dagger} B\Sigma^* \times \Sigma^* \xrightarrow{\pi_1} B\Sigma^* \tag{6.6}$$

Recall from Section 3.5.2 that $\tau^\dagger$ is the extension of τ to a distributive law; intuitively, it is the inductive extension of τ to terms. Informally, the above natural transformation acts as follows. For an operator σ of arity n, given behaviour $k_1, \ldots, k_n \in BX \times X$ of its arguments, it first applies the assignment rule d to obtain a term $t(k_1, \ldots, k_n)$. Subsequently $\tau^\dagger$ is used to compute the behaviour of t given the behaviour $k_1, \ldots, k_n$. In short, the above transformation computes the behaviour of an operator by using rules from τ and a single application of the assignment rule d.

Example 6.2.2. Suppose the signature Σ contains a binary operator and a unary operator (to be interpreted as parallel composition | and replication ! respectively). Further, let ρ be a GSOS specification defined as usual for | (Example 3.5.4), and without any rules for the replication operator $!x$. Let d be the assignment rule associated to the replication, i.e., the identity on all operators except $!x$, which is mapped to $!x|x$.

Then the natural transformation in (6.6) corresponds to a specification in which there is a rule that concludes with $!x \to t$ for some t if and only if there is a derivation of $!x|x \to t$ in the GSOS specification ρ, from the same premises. Since there are no rules for $!x$ in ρ, the only possible derivation is

$$\frac{x \xrightarrow{a} x'}{!x|x \xrightarrow{a} !x|x'}$$

and therefore, the only rule for $!x$ is

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} !x|x'}$$

The natural transformation in (6.6) is unchanged on all other operators.

As explained in the introduction of this chapter, this is not quite the correct specification of replication yet, but it is a first step. To obtain the correct specification, we need to apply such a construction recursively, which we will do below. First we define a function φ on $\mathbb{G}$ which uses the above construction to build, from an argument specification τ (of Σ over B), the specification containing all rules from the fixed specification ρ and all rules which can be formed as in (6.6).

Definition 6.2.3. Given our fixed ρ and Δ (Assumption 6.1.6), the map $\varphi \colon \mathbb{G} \to \mathbb{G}$ is defined as

$$\varphi(\tau) = \rho \vee \bigvee_{d \in \Delta} (\pi_1 \circ \tau^\dagger \circ d_{B \times \mathsf{Id}}) .$$

For well-definedness, we need to check that φ preserves monotonicity. To this end, it is convenient to speak about monotonicity of a distributive law $\tau^\dagger$, which requires an order on Σ^*. Any partial order $(X, \leq)$ inductively extends to an order on $\Sigma^* X$ by defining

$$\sigma(t_1, \ldots, t_n) \leq \tau(u_1, \ldots, u_m)$$

iff $\sigma = \tau$ (so also $n = m$) and $t_i \leq u_i$ for all $i \leq n$. We thus get a notion of monotonicity of distributive laws (this can be defined more generally using relation lifting, see Section 5.4.1; here, we provide a concrete, self-contained exposition).

Lemma 6.2.4. *If τ is a monotone specification, then $\varphi(\tau)$ is monotone as well.*

Proof. We prove that if τ is monotone then the induced distributive law $\tau^\dagger\colon \Sigma^*(B\times \mathsf{Id}) \Rightarrow B\Sigma^*\times\Sigma^*$ is also monotone, by induction on pairs of terms $t,u \in \Sigma^*(BX\times X)$ with $t \leq u$ (note that this order is defined inductively). The desired result that $\varphi(\tau)$ is monotone then follows, since assignment rules d are clearly monotone.

For the base case, if $(b,x),(c,y) \in BX \times X$ with $(b,x) \leq (c,y)$ (so $b \leq c$ and $x = y$) then

$$\tau^\dagger_X \circ \eta_{BX\times X}(b,x) = (B\eta_X \times \eta_X)(b,x) \leq (B\eta_X \times \eta_X)(c,y) = \tau^\dagger_X \circ \eta_{BX\times X}(c,y)$$

where the inequality holds by monotonicity of $B\eta_X$ and since $x = y$, and the equalities by definition of $\tau^\dagger$ (Equation (3.15) in Section 3.5.2).

Suppose σ is an operator of arity n, and $t_1,\ldots,t_n,u_1,\ldots,u_n \in \Sigma^*(BX\times X)$ with $\tau^\dagger_X(t_i) \leq \tau^\dagger_X(u_i)$ for all i. Then

$$\begin{array}{ll}
\tau^\dagger_X \circ \kappa_{BX\times X}(\sigma(t_1,\ldots,t_n)) & \\
= (B\mu_X \times \kappa_X) \circ \langle \tau_{\Sigma^* X}, \Sigma\pi_2\rangle \circ \Sigma\tau^\dagger_X(\sigma(t_1,\ldots,t_n)) & \text{definition } \tau^\dagger \\
= (B\mu_X \times \kappa_X) \circ \langle \tau_{\Sigma^* X}, \Sigma\pi_2\rangle(\sigma(\tau^\dagger_X(t_1),\ldots,\tau^\dagger_X(t_n))) & \text{definition } \Sigma \\
\leq (B\mu_X \times \kappa_X) \circ \langle \tau_{\Sigma^* X}, \Sigma\pi_2\rangle(\sigma(\tau^\dagger_X(u_1),\ldots,\tau^\dagger_X(u_n))) & \text{see below} \\
= \tau^\dagger_X \circ \kappa_{BX\times X}(\sigma(u_1,\ldots,u_n)) &
\end{array}$$

The inequality holds by monotonicity of $B\mu_X$ and τ, and the induction hypothesis; note that the induction hypothesis implies $\pi_2 \circ \tau^\dagger_X(t_i) = \pi_2 \circ \tau^\dagger_X(u_i)$ for all i. ☐

Moreover, φ is monotone on $\mathbb{G}$:

Lemma 6.2.5. *The function $\varphi\colon \mathbb{G} \to \mathbb{G}$ is monotone.*

The main step in the proof of Lemma 6.2.5 is to show that the extension $(-)^\dagger$ of abstract GSOS specifications to distributive laws is monotone.

Lemma 6.2.6. *Let τ_1,τ_2 be specifications. If $\tau_1 \leq \tau_2$ then $\pi_1 \circ (\tau_1^\dagger) \leq \pi_1 \circ (\tau_2^\dagger)$.*

Proof. We have

$$(\tau_1^\dagger)_X \circ \eta_{BX\times X} = B\eta_X \times \eta_X = (\tau_2^\dagger)_X \circ \eta_{BX\times X}$$

by definition of $(-)^\dagger$ (Equation (3.15) in Section 3.5.2). Moreover

$$(B\mu_X \times \kappa_X) \circ \langle (\tau_1)_{\Sigma^* X}, \Sigma\pi_2\rangle \leq (B\mu_X \times \kappa_X) \circ \langle (\tau_2)_{\Sigma^* X}, \Sigma\pi_2\rangle$$

by monotonicity of $B\mu$ and assumption. Now using the definition of $(\tau_1^\dagger)_X$, it easily follows by induction on terms in $\Sigma^*(BX \times X)$ that $(\tau_1^\dagger)_X \leq (\tau_2^\dagger)_X$, and thus $\pi_1 \circ (\tau_1^\dagger)_X \leq \pi_1 \circ (\tau_2^\dagger)_X$. ☐

Because φ is monotone, it has a least fixed point, which we denote by $\mathsf{lfp}(\varphi)$. Further, since φ preserves monotonicity we obtain monotonicity of $\mathsf{lfp}(\varphi)$ by transfinite induction (the base case and limit steps are rather easy). The proof technique of transfinite induction, which we also use several times below, is justified by the fact that the least fixed point of a monotone function in a complete lattice can be constructed as the supremum of an ascending chain obtained by iterating the function over the ordinals (see, e.g., [San12a]).

Corollary 6.2.7. *The abstract GSOS specification* $\mathsf{lfp}(\varphi)$ *is monotone.*

Informally, $\mathsf{lfp}(\varphi)$ is the specification consisting of rules from ρ and Δ. We proceed to prove that the operational model of the least fixed point of φ is precisely the interpretation of ρ and Δ (the least fixed point of ψ as given in Definition 6.1.7), i.e., that $M(\mathsf{lfp}(\varphi)) = \mathsf{lfp}(\psi)$. First, we show that $M(\mathsf{lfp}(\varphi))$ is a fixed point of ψ.

Lemma 6.2.8. *The operational model* $M(\mathsf{lfp}(\varphi))$ *of the specification* $\mathsf{lfp}(\varphi)$ *is a* (ρ, Δ)*-model, i.e.,* $\psi(M(\mathsf{lfp}(\varphi))) = M(\mathsf{lfp}(\varphi))$.

Proof. Let $f = M(\mathsf{lfp}(\varphi))$. We must show that $\psi(f) = f$.

$$\begin{aligned}
& f \circ \kappa_\emptyset \\
&= B\mu_\emptyset \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \\
&= B\mu_\emptyset \circ (\rho \vee \textstyle\bigvee_{d\in\Delta} \pi_1 \circ (\mathsf{lfp}(\varphi))^\dagger \circ d_{B\times\mathsf{Id}})_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \\
&= B\mu_\emptyset \circ (\rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \textstyle\bigvee_{d\in\Delta} \pi_1 \circ (\mathsf{lfp}(\varphi))^\dagger_{\Sigma^*\emptyset} \circ d_{B\Sigma^*\emptyset\times\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle) \\
&= B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \textstyle\bigvee_{d\in\Delta} B\mu_\emptyset \circ \pi_1 \circ (\mathsf{lfp}(\varphi))^\dagger_{\Sigma^*\emptyset} \circ d_{B\Sigma^*\emptyset\times\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle
\end{aligned}$$

where the first equality holds by definition of M, the second since $\mathsf{lfp}(\varphi)$ is a fixed point of φ, the third holds by the definition of the join on natural transformations and the last one holds by the fact the $B\mu_\emptyset$ preserves joins. For the right-hand part, we have

$$\begin{array}{cll}
& \bigvee_{d\in\Delta} B\mu_\emptyset \circ \pi_1 \circ (\mathsf{lfp}(\varphi))^\dagger_{\Sigma^*\emptyset} \circ d_{B\Sigma^*\emptyset\times\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle & \\
= & \bigvee_{d\in\Delta} \pi_1 \circ B\mu_\emptyset \times \mu_\emptyset \circ (\mathsf{lfp}(\varphi))^\dagger_{\Sigma^*\emptyset} \circ \Sigma^*\langle f, \mathsf{id}\rangle \circ d_{\Sigma^*\emptyset} & \text{naturality of } d,\ \pi_1 \\
= & \bigvee_{d\in\Delta} \pi_1 \circ \langle f, \mathsf{id}\rangle \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} & (\Sigma\emptyset^*, \mu_\emptyset, \langle f, \mathsf{id}\rangle) \text{ is a} \\
= & \bigvee_{d\in\Delta} f \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} & (\mathsf{lfp}(\varphi))^\dagger\text{-bialg.}
\end{array}$$

Thus $f \circ \kappa_\emptyset = B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} f \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} = \psi(f) \circ \kappa_\emptyset$ and consequently $\psi(f) = f$, since $\kappa_\emptyset$ is an isomorphism. □

We proceed to show that $M(\mathsf{lfp}(\varphi)) \leq \mathsf{lfp}(\psi)$. Since $\psi(M(\mathsf{lfp}(\varphi))) = M(\mathsf{lfp}(\varphi))$ by the above Lemma 6.2.8, we then have $M(\mathsf{lfp}(\varphi)) = \mathsf{lfp}(\psi)$ (Theorem 6.2.14). The main step is that any fixed point of ψ is "closed under ρ", i.e., that in such a model, all the behaviour that we can derive by the specification is already there. This result is the contents of Corollary 6.2.13 below; it follows by transfinite induction from Lemma 6.2.11 and 6.2.12. But first, we need a few technical tools (Lemma 6.2.9 and 6.2.10). Recall from Section 3.4 that a Σ-algebra $\alpha\colon \Sigma X \to X$ induces an algebra $\widehat{\alpha}\colon \Sigma^* X \to X$ for the free monad. This construction preserves algebra morphisms. We prove a lax version of this fact.

Lemma 6.2.9. *Let* $\alpha\colon \Sigma X \to X$ *and* $\beta\colon \Sigma Y \to Y$ *be algebras, such that* Y *carries a partial order* $\leq$ *and* β *is monotone. Then for any function* $f\colon X \to Y$*:*

$$\begin{tikzcd}
\Sigma X \arrow[r, "\Sigma f"] \arrow[d, "\alpha"'] \arrow[dr, phantom, "\geq"] & \Sigma Y \arrow[d, "\beta"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}
\quad \textit{implies} \quad
\begin{tikzcd}
\Sigma^* X \arrow[r, "\Sigma^* f"] \arrow[d, "\widehat{\alpha}"'] \arrow[dr, phantom, "\geq"] & \Sigma^* Y \arrow[d, "\widehat{\beta}"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

Proof. Suppose $\beta \circ \Sigma f \leq f \circ \alpha$. The proof is by induction on $t \in \Sigma^* X$. For the base case $t = \eta_X(s) \in \Sigma^* X$, we have an equality, without using the assumption:

$$\widehat{\beta} \circ \Sigma^* f \circ \eta_X(s) = \widehat{\beta} \circ \eta_Y \circ f(s) = f(s) = f \circ \widehat{\alpha} \circ \eta_X(s)\,.$$

Now suppose $\sigma \in \Sigma$ is of arity n, and for some $t_1, \ldots, t_n \in \Sigma^* X$, we have $\widehat{\beta} \circ (\Sigma^* f)(t_i) \leq f \circ \widehat{\alpha}(t_i)$ for all i with $1 \leq i \leq n$. Then

$$\begin{array}{ll}
\widehat{\beta} \circ \Sigma^* f \circ \kappa_X(\sigma(t_1, \ldots, t_n)) & \\
= \widehat{\beta} \circ \kappa_Y \circ \Sigma\Sigma^* f(\sigma(t_1, \ldots, t_n)) & \text{naturality } \kappa \\
= \widehat{\beta} \circ \kappa_Y(\sigma(\Sigma^* f(t_1), \ldots, \Sigma^* f(t_n))) & \text{definition } \Sigma \\
= \beta \circ \Sigma\widehat{\beta}(\sigma(\Sigma^* f(t_1), \ldots, \Sigma^* f(t_n))) & \text{definition } \widehat{\beta} \\
= \beta(\sigma(\widehat{\beta} \circ \Sigma^* f(t_1), \ldots, \widehat{\beta} \circ \Sigma^* f(t_n))) & \text{definition } \Sigma \\
\leq \beta(\sigma(f \circ \widehat{\alpha}(t_1), \ldots, f \circ \widehat{\alpha}(t_n))) & \text{ind. hypothesis, monotonicity } \beta \\
= \beta \circ \Sigma f \circ \Sigma\widehat{\alpha}(\sigma(t_1, \ldots, t_n)) & \text{definition } \Sigma \\
\leq f \circ \alpha \circ \Sigma\widehat{\alpha}(\sigma(t_1, \ldots, t_n)) & \text{assumption} \\
= f \circ \widehat{\alpha} \circ \kappa_X(\sigma(t_1, \ldots, t_n)) & \text{definition } \widehat{\alpha}
\end{array}$$

which concludes the induction step. □

We instantiate the above lemma to the definition of $\tau^\dagger$.

Lemma 6.2.10. *Let τ be a monotone abstract GSOS specification of Σ over B. Then for any $f \colon \Sigma^*\emptyset \to B\Sigma^*\emptyset$:*

$$\begin{tikzcd}
\Sigma\Sigma^*\emptyset \arrow[r, "\Sigma\langle f, \mathsf{id}\rangle"] \arrow[dd, "\kappa_\emptyset"'] & \Sigma(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\tau_{\Sigma^*\emptyset}"] & & \Sigma^*\Sigma^*\emptyset \arrow[r, "\Sigma^*\langle f, \mathsf{id}\rangle"] \arrow[dd, "\mu_\emptyset"'] & \Sigma^*(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\tau^\dagger_{\Sigma^*\emptyset}"] \\
{} \arrow[r, phantom, "\geq"] & B\Sigma^*\Sigma^*\emptyset \arrow[d, "B\mu_\emptyset"] & \textit{implies} & {} \arrow[r, phantom, "\geq"] & B\Sigma^*\Sigma^*\emptyset \times \Sigma^*\Sigma^*\emptyset \arrow[d, "B\mu_\emptyset \times \mu_\emptyset"] \\
\Sigma^*\emptyset \arrow[r, "f"'] & B\Sigma^*\emptyset & & \Sigma^*\emptyset \arrow[r, "\langle f, \mathsf{id}\rangle"'] & B\Sigma^*\emptyset \times \Sigma^*\emptyset
\end{tikzcd}$$

Proof. From the assumption it follows that

$$(B\mu_\emptyset \times \kappa_\emptyset) \circ \langle \tau_{\Sigma^*\emptyset}, \Sigma\pi_2 \rangle \circ \Sigma\langle f, \mathsf{id}\rangle \leq \langle f, \mathsf{id}\rangle \circ \kappa_\emptyset\,.$$

Let $\beta = (B\mu_\emptyset \times \kappa_\emptyset) \circ \langle \tau_{\Sigma^*\emptyset}, \Sigma\pi_2 \rangle$, then by Lemma 6.2.9 we get

$$\begin{tikzcd}
\Sigma^*\Sigma^*\emptyset \arrow[r, "\Sigma^*\langle f, \mathsf{id}\rangle"] \arrow[d, "\mu_\emptyset"'] \arrow[dr, phantom, "\geq"] & \Sigma^*(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\widehat{\beta}"] \\
\Sigma^*\emptyset \arrow[r, "\langle f, \mathsf{id}\rangle"'] & B\Sigma^*\emptyset \times \Sigma^*\emptyset
\end{tikzcd}$$

where $\widehat{\beta}$ is the Σ^*-algebra induced by the Σ-algebra $\beta = (B\mu_\emptyset \times \kappa_\emptyset) \circ \langle \tau_{\Sigma^*\emptyset}, \Sigma\pi_2 \rangle$. Thus, it only remains to prove that $\widehat{\beta} = (B\mu_\emptyset \times \mu_\emptyset) \circ \tau^\dagger_{\Sigma^*\emptyset}$.

To this end, consider the following diagram:

$$
\begin{tikzcd}
\Sigma\Sigma^*(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[r, "\Sigma\tau^\dagger_{\Sigma^*\emptyset}"] \arrow[dd, "\kappa_{B\Sigma^*\emptyset\times\Sigma^*\emptyset}"'] & \Sigma(B\Sigma^*\Sigma^*\emptyset \times \Sigma^*\Sigma^*\emptyset) \arrow[r, "\Sigma(B\mu_\emptyset\times\mu_\emptyset)"] \arrow[d, "\langle \tau_{\Sigma^*\Sigma^*\emptyset}, \Sigma\pi_2\rangle"] & \Sigma(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\langle \tau_{\Sigma^*\emptyset}, \Sigma\pi_2\rangle"] \\
 & B\Sigma^*\Sigma^*\Sigma^*\emptyset \times \Sigma\Sigma^*\Sigma^*\emptyset \arrow[r, "B\Sigma^*\mu_\emptyset\times\Sigma\mu_\emptyset"] \arrow[d, "B\mu_{\Sigma^*\emptyset}\times\kappa_{\Sigma^*\emptyset}"] & B\Sigma^*\Sigma^*\emptyset \times \Sigma\Sigma^*\emptyset \arrow[d, "B\mu_\emptyset\times\kappa_\emptyset"] \\
\Sigma^*(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[r, "\tau^\dagger_{\Sigma^*\emptyset}"] & B\Sigma^*\Sigma^*\emptyset \times \Sigma^*\Sigma^*\emptyset \arrow[r, "B\mu_\emptyset\times\mu_\emptyset"] & B\Sigma^*\emptyset \times \Sigma^*\emptyset \\
B\Sigma^*\emptyset \times \Sigma^*\emptyset \arrow[u, "\eta_{B\Sigma^*\emptyset\times\Sigma^*\emptyset}"] \arrow[ur, "B\eta_{\Sigma^*\emptyset}\times\eta_{\Sigma^*\emptyset}"'] & &
\end{tikzcd}
$$

The upper right rectangle commutes by naturality, the lower right rectangle commutes by the multiplication law of the monad and since $\mu_\emptyset = \widehat{\kappa_\emptyset}$. The left square and triangle commute by definition of $\tau^\dagger$ (Equation (3.15) in Section 3.5.2). Thus $(B\mu_\emptyset \times \mu_\emptyset) \circ \tau^\dagger_{\Sigma^*\emptyset}$ is an algebra homomorphism extending id, and since $\widehat{\beta}$ is by definition an algebra homomorphism extending id and homomorphic extensions are unique, we have $\widehat{\beta} = B\mu_\emptyset \times \mu_\emptyset \circ \tau^\dagger_{\Sigma^*\emptyset}$. □

Lemma 6.2.11. *Let τ be a specification, and $f \in \mathbb{M}$ a fixed point of ψ. If $B\mu_\emptyset \circ \tau_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq f \circ \kappa_\emptyset$ then $B\mu_\emptyset \circ \varphi(\tau)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq f \circ \kappa_\emptyset$.*

Proof.

$$
\begin{aligned}
& B\mu_\emptyset \circ \varphi(\tau)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \\
&= B\mu_\emptyset \circ (\rho \vee \bigvee_{d\in\Delta} \pi_1 \circ \tau^\dagger \circ d_{B\times\mathsf{Id}})_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \\
&= B\mu_\emptyset \circ (\rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} \pi_1 \circ \tau^\dagger_{\Sigma^*\emptyset} \circ d_{B\Sigma^*\emptyset\times\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle) \\
&= B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} B\mu_\emptyset \circ \pi_1 \circ \tau^\dagger_{\Sigma^*\emptyset} \circ d_{B\Sigma^*\emptyset\times\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \\
&= B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} \pi_1 \circ (B\mu_\emptyset \times \mu_\emptyset) \circ \tau^\dagger_{\Sigma^*\emptyset} \circ \Sigma^*\langle f, \mathsf{id}\rangle \circ d_{\Sigma^*\emptyset} \\
&\leq B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} \pi_1 \circ \langle f, \mathsf{id}\rangle \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} \\
&= B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{d\in\Delta} f \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} \\
&= \psi(f) \circ \kappa_\emptyset = f \circ \kappa_\emptyset
\end{aligned}
$$

The first equality holds by definition of φ, the second by definition of the join of specifications, the third since $B\mu_\emptyset$ is join-preserving, and the fourth equality by naturality of d and π_1. The inequality holds by assumption and Lemma 6.2.10. The last equality holds by definition of ψ. □

Lemma 6.2.12. *Let $f \in \mathbb{M}$ such that $\psi(f) = f$, and suppose we have a family $\{\tau_i\}_{i \in I}$ of specifications, for some index set I. If $B\mu_\emptyset \circ (\tau_i)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq f \circ \kappa_\emptyset$ for all $i \in I$, then $B\mu_\emptyset \circ (\bigvee_{i\in I} \tau_i)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq f \circ \kappa_\emptyset$.*

Proof. Since $B\mu_\emptyset$ preserves joins we have

$$B\mu_\emptyset \circ (\bigvee_{i\in I} \tau_i)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle = \bigvee_{i\in I} B\mu_\emptyset \circ (\tau_i)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle$$

and the result now follows by the assumption that $B\mu_\emptyset \circ (\tau_i)_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \leq f \circ \kappa_\emptyset$ for each i. □

Corollary 6.2.13. *For any $f \in \mathbb{M}$: if $\psi(f) = f$ then*

$$\begin{tikzcd}
\Sigma\Sigma^* X \arrow[r, "\Sigma\langle f, \mathsf{id}\rangle"] \arrow[dd, "\kappa_\emptyset"'] & \Sigma(B\Sigma^*\emptyset \times \Sigma^*\emptyset) \arrow[d, "\mathsf{lfp}(\varphi)_{\Sigma^*\emptyset}"] \\
\geq & B\Sigma^*\Sigma^*\emptyset \arrow[d, "B\mu_\emptyset"] \\
\Sigma^* X \arrow[r, "f"'] & B\Sigma^*\emptyset
\end{tikzcd}$$

Proof. By transfinite induction. For the base case we have $B\mu_\emptyset \circ \bot \circ \Sigma\langle f, \mathsf{id}\rangle = \bot \leq f \circ \kappa_\emptyset$. The successor step is given by Lemma 6.2.11 and the limit step by Lemma 6.2.12. □

This allows to prove the main result of this chapter.

Theorem 6.2.14. *The interpretation of ρ and Δ coincides with the operational model of the abstract GSOS specification $\mathsf{lfp}(\varphi)$, i.e., $M(\mathsf{lfp}(\varphi)) = \mathsf{lfp}(\psi)$.*

Proof. By Lemma 6.2.8, $M(\mathsf{lfp}(\varphi))$ is a fixed point of ψ. To show it is the least one, let f be any fixed point of ψ; we proceed to prove $M(\mathsf{lfp}(\varphi)) \leq f$ by structural induction on closed terms. Suppose $\sigma \in \Sigma$ is an operator of arity n, and suppose we have $t_1, \ldots, t_n \in \Sigma^*\emptyset$ such that $M(\mathsf{lfp}(\varphi))(t_i) \leq f(t_i)$ for all i with $1 \leq i \leq n$ (note that this trivially holds in the base case, when $n = 0$). Then

$$\begin{aligned}
M(\mathsf{lfp}(\varphi))(\sigma(t_1, \ldots, t_n)) &= B\mu_\emptyset \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle M(\mathsf{lfp}(\varphi)), \mathsf{id}\rangle(\sigma(t_1, \ldots, t_n)) \\
&\leq B\mu_\emptyset \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle(\sigma(t_1, \ldots, t_n)) \\
&\leq f(\sigma(t_1, \ldots, t_n))
\end{aligned}$$

where the first inequality holds by assumption and monotonicity of $B\mu_\emptyset$ and $\mathsf{lfp}(\varphi)$ (Corollary 6.2.7) and the second by Corollary 6.2.13. □

As a consequence, the interpretation of ρ and Δ is well-behaved.

Corollary 6.2.15. *Bisimilarity is a congruence on the interpretation* $\mathsf{lfp}(\psi)$ *of* ρ *and* Δ*, and bisimulation up to context is compatible (i.e., the contextual closure is* $\mathsf{b}_{\mathsf{lfp}(\psi)}$*-compatible).*

Example 6.2.16. The parallel composition can be given by a positive GSOS specification, and Equation (6.1) of the introduction contains a rule for the replication operator. Thus, by the above Corollary, bisimilarity is a congruence on the operational model of CCS with replication, and bisimulation up to context is compatible; this is known (see, e.g., [San12a]), but here we obtain it directly from the format and the above results.

Example 6.2.17. We revisit the general process algebra with transition costs (GPA) (see Example 4.5.5, [BK01]). We consider basic GPA processes with procedures, defined by the grammar $t ::= 0 \mid t + t \mid (a, r).t \mid p$ where a ranges over the set of actions A, r ranges over the positive real numbers $\mathbb{R}^+$ and p ranges over a fixed set of procedure names $PNames$. We assume that each procedure name $p_i \in PNames$ has a body $t_i \in P$.

The operational semantics of the operators of basic GPA processes on the complete monoid $\mathbb{R}^+ \cup \{\infty\}$ (with supremum) is similar to the semantics in Example 4.5.5. The semantics corresponds to a GSOS specification; see [Kli11] for details. This specification is monotone. The (recursive) procedures can now be interpreted by assignment rules: for each $p_i \in PNames$ we add an assignment rule $p_i := t_i$. Intuitively this means that the procedure call p_i is given by the behaviour of its body t_i, as expected. By Theorem 6.2.14, bisimilarity is a congruence on the interpretation.

6.3 Structural congruences

The assignment rules considered in the theory of the previous sections copy behaviour from a term to an operator, but this assignment goes one way only. In this section, we consider the combination of abstract GSOS specifications with actual *equations*, interpreted by the structural congruence rule. By encoding equations in a restricted format as assignment rules, we obtain that the interpretation of any specification with equations in this format is well-behaved.

Equations are elements of $\Sigma^*V \times \Sigma^*V$, where V is an arbitrary but fixed set of variables. A set of equations $E \subseteq \Sigma^*V \times \Sigma^*V$ induces a *congruence* $\equiv_E$:

Definition 6.3.1. Let $E \subseteq \Sigma^*V \times \Sigma^*V$ be a set of equations. The *congruence closure* $\equiv_E$ of E is the least relation on $\Sigma^*\emptyset$ satisfying the following rules:

$$\frac{t\,E\,u \quad s\colon V \to \Sigma^*\emptyset}{s^\sharp(t) \equiv_E s^\sharp(u)} \qquad \frac{}{t \equiv_E t} \qquad \frac{u \equiv_E t}{t \equiv_E u} \qquad \frac{t \equiv_E u \qquad u \equiv_E v}{t \equiv_E v}$$

$$\frac{t_1 \equiv_E u_1 \qquad \ldots \qquad t_n \equiv_E u_n}{\sigma(t_1, \ldots, t_n) \equiv_E \sigma(u_1, \ldots, u_n)} \qquad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

where $s^\sharp \colon \Sigma^* V \to \Sigma^* \emptyset$ is the inductive extension of s to terms (Section 3.4).

In the context of structural operational semantics, equations are often interpreted by the *structural congruence rule*:

$$\frac{t \equiv_E u \qquad u \xrightarrow{a} u' \qquad u' \equiv_E v}{t \xrightarrow{a} v} \tag{6.7}$$

Informally, this rule states that we can use the specification to derive transitions modulo the congruence generated by the equations. In fact, removing the part $u' \equiv_E v$ from the premise (and writing u' instead of v in the conclusion) does not affect the behaviour, modulo bisimilarity [MR05]. See [MR05] for details on the interpretation of structural congruences in the context of transition systems.

We denote by $(\Sigma^*\emptyset)/\equiv_E$ the set of equivalence classes, and by $q \colon \Sigma^*\emptyset \to (\Sigma^*\emptyset)/\equiv_E$ the quotient map of $\equiv_E$ (we remark that one can equip $(\Sigma^*\emptyset)/\equiv_E$ with an algebra structure μ' such that q is a Σ^*-algebra homomorphism). Thus $q(t) = q(u)$ iff $t \equiv_E u$. Assuming the axiom of choice, we have $t \equiv_E u$ iff there is a right inverse $r \colon (\Sigma^*\emptyset)/\equiv_E \to \Sigma^*\emptyset$ such that $r(q(t)) = u$. The latter fact is exploited in the interpretation of a specification together with a set of equations.

Definition 6.3.2. Let $\theta \colon \mathbb{M} \to \mathbb{M}$ be the (unique) function such that

$$\theta(f) \circ \kappa_\emptyset = B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle f, \mathsf{id}\rangle \vee \bigvee_{r \in R} f \circ r \circ q \circ \kappa_\emptyset \,.$$

where R is the set of right inverses of q. A (ρ, E)-model is a coalgebra $f \in \mathbb{M}$ such that $\theta(f) = f$.

Lemma 6.3.3. *The function $\theta \colon \mathbb{M} \to \mathbb{M}$ is monotone.*

Proof. Similar to the proof of Lemma 6.1.8. □

Definition 6.3.4. The *interpretation* of ρ and E is the least (ρ, E)-model, i.e., $\mathsf{lfp}(\theta)$.

Example 6.3.5. Consider the specification of the parallel composition $x|y$ as given in (6.2) in the introduction of this chapter, i.e., by a single rule and commutativity. In the interpretation, if $t \xrightarrow{a} t'$ then $t|u \xrightarrow{a} t'|u$, simply by the SOS rule. But also $u|t \xrightarrow{a} t'|u$, since $t|u \equiv_E u|t$. Concerning the definition of the replication operator by the equation $!x = !x|x$, for a term t the interpretation contains the least set of transitions from $!t$ which satisfy the equation, as desired.

In general, bisimilarity is not a congruence when equations are added. For convenience we recall a counterexample on transition systems [MR05].

Example 6.3.6. Consider rules $\overline{p \xrightarrow{a} p}$ and $\overline{q \xrightarrow{a} p}$ and the single equation $p = \sigma(q)$, where p, q are constants, σ is a unary operator and a is an arbitrary label. In the interpretation, p is bisimilar to q, but $\sigma(p)$ is not bisimilar to $\sigma(q)$.

The above counterexample is based on assigning behaviour to the term $\sigma(q)$, rather than defining each operator independently of its arguments. To rule out such assignments, a restricted format of equations was introduced in [MR05], called cfsc. The main result of [MR05] is that for any specification in the tyft format combined with cfsc equations, bisimilarity is a congruence.

Definition 6.3.7. A set of equations $E \subseteq \Sigma^* V \times \Sigma^* V$ is in cfsc with respect to ρ if every equation is of one of the following forms:

1. A *σx-equation*: $\sigma_1(x_1, \ldots, x_n) = \sigma_2(y_1, \ldots, y_n)$, where $\sigma_1, \sigma_2 \in \Sigma$ are of arity n (possibly $\sigma_1 = \sigma_2$), $x_1, \ldots, x_n$ are distinct variables and $y_1, \ldots, y_n$ is a permutation of $x_1, \ldots, x_n$.
2. A *defining equation*: $\sigma(x_1, \ldots, x_n) = t$ where $\sigma \in \Sigma$ and t is an arbitrary term (which may involve σ again); $x_1, \ldots, x_n$ are distinct variables, and all variables that occur in t are in $x_1, \ldots, x_n$. Moreover σ does not appear in any other equation in E, and $\rho_X(\sigma(u_1, \ldots, u_n)) = \perp$ for any set X and any $u_1, \ldots, u_n \in BX \times X$.

A *σx-equation* allows to assign simple algebraic properties to operators which already have behaviour; the prototypical example here is commutativity, like in the specification of the parallel composition in (6.2). With a *defining equation*, as the name suggests, one can define the behaviour of an operator. An example is $!x = !x|x$; another example is $p = q|z|a.p$ where p, q and z are constants. Further, the procedure declarations of Example 6.2.17 can be modelled by defining equations. Associativity of $|$ is neither a σx-equation nor a defining one. We refer to [MR05] for arguments that the cfsc format cannot be trivially extended. The cfsc format depends on an abstract GSOS specification: operators at the left hand side of a defining equation should not get any behaviour in the specification. This restriction ensures that one can not assign behaviour to complex terms, disallowing a situation such as in Example 6.3.6.

We proceed to show that the interpretation of an abstract GSOS specification ρ and a set of equations E in cfsc equals the operational model of a certain other specification, up to bisimilarity (Definition 4.4.10). This is done by encoding equations in this format as assignment rules, and using the theory of the previous section to obtain the desired result.

First, note that for any σx-equation $\sigma_1(x_1, \ldots, x_n) = \sigma_2(y_1, \ldots, y_n)$, the variables on one side are a permutation of the variables on the other, hence a σx-equation can equivalently be represented as a triple (σ_1, σ_2, p) where $p\colon \mathsf{Id}^n \to \mathsf{Id}^n$ is the natural transformation corresponding to the permutation of variables in the equation.

Definition 6.3.8. A set of equations E in cfsc defines a set of assignment rules Δ^E as follows:

1. For every σx-equation (σ_1, σ_2, p) we define d and d' on a component X as

$$d_X(\sigma(u_1, \ldots, u_n)) = \begin{cases} \sigma_2(p_X(u_1, \ldots, u_n)) & \text{if } \sigma = \sigma_1 \\ \sigma(u_1, \ldots, u_n) & \text{otherwise} \end{cases}$$

for all $u_1, \ldots, u_n \in X$, and d' is similarly defined using the inverse permutation p^{-1}, with and σ_1 and σ_2 swapped.

2. For every defining equation $\sigma_1(x_1, \ldots, x_n) = t$ we define a corresponding assignment rule

$$d_X(\sigma(u_1, \ldots, u_n)) = \begin{cases} t[x_1 := u_1, \ldots, x_n := u_n] & \text{if } \sigma = \sigma_1 \\ \sigma(u_1, \ldots, u_n) & \text{otherwise} \end{cases}$$

for any set X and all $u_1, \ldots, u_n \in X$.

Remark 6.3.9. In [MR05], σx-equations are a bit more liberal in that they do not require the arities of σ_1 and σ_2 to coincide, and do allow variables which only occur on one side of the equation. But in the interpretation these variables are quantified universally over closed terms; thus, we can encode this using infinitely many assignment rules. For example, an equation $\sigma_1(x) = \sigma_2(x, y)$ can be encoded by the set of assignment rules, one for each term $t \in \Sigma^*\emptyset$ mapping $\sigma_1(x)$ to $\sigma_2(x, t)$. We work with the simpler format above for technical convenience.

We prove that the encoding of equations as assignment rules is correct with respect to the interpretation of the equations (Theorem 6.3.13). First, we show that if $\sigma(x_1, \ldots, x_n) = t$ is a defining equation of a set of equations in the cfsc format, then the behaviour of $\sigma(x_1, \ldots, x_n)$ will be below that of t.

Lemma 6.3.10. *Let E be a set of equations in cfsc format w.r.t. ρ, and let ψ be as in Definition 6.1.7 for (ρ, Δ^E). Then for any defining equation $\sigma(x_1, \ldots, x_n) = t$ and any $t_1, \ldots, t_n \in \Sigma^*\emptyset$: $\mathsf{lfp}(\psi) \circ \kappa_\emptyset(\sigma(t_1, \ldots, t_n)) \leq \mathsf{lfp}(\psi) \circ \mu_\emptyset(t[x_1 := t_1, \ldots, x_n := t_n])$.*

Proof. Given a defining equation, let $d \in \Delta^E$ be the natural transformation that encodes it (see Definition 6.3.8(2)). We prove by transfinite induction that for any function $g \in \mathbb{M}$ that arises in the iterative construction of $\mathsf{lfp}(\psi)$ and for any $t_1, \ldots, t_n \in \Sigma^*\emptyset$ we have

$$g \circ \kappa_\emptyset(\sigma(t_1, \ldots, t_n)) \leq \mathsf{lfp}(\psi) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1, \ldots, t_n)) \,. \tag{6.8}$$

The base case is when $g = \bot$, which is trivial. Now suppose that (6.8) holds for some $g \leq \mathsf{lfp}(\psi)$. Then

$$\psi(g) \circ \kappa_\emptyset(\sigma(t_1, \ldots, t_n)) = (B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle g, \mathsf{id}\rangle \vee \bigvee_{d' \in \Delta^E} g \circ \mu_\emptyset \circ d'_{\Sigma^*\emptyset})(\sigma(t_1, \ldots, t_n)) \,.$$

But since the equations are in cfsc format, we have

$$B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle g, \mathsf{id}\rangle(\sigma(t_1, \ldots, t_n)) = \bot \,. \tag{6.9}$$

Moreover, again by the cfsc format, $\sigma(t_1, \ldots, t_n)$ does not occur in any equation other than the defining one in E, and thus for all $d' \in \Delta^E$ with $d' \neq d$ we have

$$g \circ \mu_\emptyset \circ d'_{\Sigma^*\emptyset}(\sigma(t_1, \ldots, t_n)) = g \circ \kappa_\emptyset(\sigma(t_1, \ldots, t_n))$$

which is below $\mathsf{lfp}(\psi) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n))$ by the induction hypothesis (6.8). Together with the assumption that $g \leq \mathsf{lfp}(\psi)$ this implies

$$\bigvee_{d' \in \Delta^E} g \circ \mu_\emptyset \circ d'_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n)) \leq \mathsf{lfp}(\psi) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n))\,.$$

By the above and (6.9), we may conclude

$$\psi(g) \circ \kappa_\emptyset(\sigma(t_1,\ldots,t_n)) \leq \mathsf{lfp}(\psi) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n))$$

as desired. This concludes the successor step; the limit step is again trivial (i.e., if we assume that (6.8) holds for a family of functions, then it also holds for the join of these functions). $\square$

The following lemma is the main step for the correctness of the encoding of equations as assignment rules.

Lemma 6.3.11. *Let E and ψ be as above. If $t \equiv_E u$ then $Bq \circ (\mathsf{lfp}(\psi))(t) = Bq \circ (\mathsf{lfp}(\psi))(u)$, where q is the quotient map of $\equiv_E$.*

Proof. The proof is by induction on $\equiv_E$, that is, we show that the set of pairs $t \equiv_E u$ that satisfy $Bq \circ (\mathsf{lfp}(\psi))(t) = Bq \circ (\mathsf{lfp}(\psi))(u)$ is closed under each of the defining rules of $\equiv_E$. For reflexivity, transitivity and symmetry this is easy. The important cases are the two types of cfsc equations from E, and congruence.

For a σx-equation $\sigma_1(t_1,\ldots,t_n) \equiv_E \sigma_2(u_1,\ldots,u_n)$, by definition of Δ^E there is an assignment rule d such that $\mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma_1(t_1,\ldots,t_n)) = \sigma_2(u_1,\ldots,u_n)$, and by definition of $\mathsf{lfp}(\psi)$ we have $\mathsf{lfp}(\psi) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} \leq \mathsf{lfp}(\psi)$; so $(\mathsf{lfp}(\psi))(\sigma_2(u_1,\ldots,u_n)) \leq (\mathsf{lfp}(\psi))(\sigma_1(t_1,\ldots,t_n))$. For the converse, there is another assignment rule d', and thus $(\mathsf{lfp}(\psi))(\sigma_1(t_1,\ldots,t_n)) \leq (\mathsf{lfp}(\psi))(\sigma_2(u_1,\ldots,u_n))$.

For a defining equation $\sigma(t_1,\ldots,t_n) \equiv_E t$ we have a natural transformation in d such that $\mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n)) = t$. Thus $(\mathsf{lfp}(\psi))(t) = (\mathsf{lfp}(\psi)) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset}(\sigma(t_1,\ldots,t_n)) \leq (\mathsf{lfp}(\psi))(\sigma(t_1,\ldots,t_n))$. The other way around follows by Lemma 6.3.10. So $(\mathsf{lfp}(\psi))(t) = (\mathsf{lfp}(\psi))(\sigma(t_1,\ldots,t_n))$.

Finally, for the congruence rule, suppose there are terms $t_1,\ldots,t_n$, $u_1,\ldots,u_n$ such that $t_i \equiv u_i$ and $Bq \circ (\mathsf{lfp}(\psi))(t_i) = Bq \circ (\mathsf{lfp}(\psi))(u_i)$ for all $i \leq n$, and σ is an operator of arity n. Notice that this implies

$$\langle Bq \circ \mathsf{lfp}(\psi), q\rangle(t_i) = \langle Bq \circ \mathsf{lfp}(\psi), q\rangle(u_i) \qquad \text{for all } i \leq n \tag{6.10}$$

since $q(t_i) = q(u_i)$ for each i. Now

$$\begin{array}{ll}
Bq \circ (\mathsf{lfp}(\psi))(\sigma(t_1,\ldots,t_n)) & \\
= Bq \circ B\mu_\emptyset \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(\sigma(t_1,\ldots,t_n)) & \text{Theorem 6.2.14} \\
= B\mu' \circ B\Sigma^* q \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(\sigma(t_1,\ldots,t_n)) & q \text{ alg. morphism} \\
= B\mu' \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma(Bq \times q) \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(\sigma(t_1,\ldots,t_n)) & \text{naturality} \\
= B\mu' \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle Bq \circ \mathsf{lfp}(\psi), q\rangle(\sigma(t_1,\ldots,t_n)) & \text{functoriality} \\
= B\mu' \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle Bq \circ \mathsf{lfp}(\psi), q\rangle(\sigma(u_1,\ldots,u_n)) & \text{ind. hypothesis} \\
= Bq \circ B\mu_\emptyset \circ (\mathsf{lfp}(\varphi))_{\Sigma^*\emptyset} \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(\sigma(u_1,\ldots,u_n)) & \\
= Bq \circ (\mathsf{lfp}(\psi))(\sigma(u_1,\ldots,u_n)) &
\end{array}$$

Notice that we used the fact that the quotient map q is an algebra morphism into some Σ^*-algebra μ'. It is worthwhile to note that we need to reason up to $\equiv_E$ to get (6.10). Indeed, $\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(t_i) = \langle \mathsf{lfp}(\psi), \mathsf{id}\rangle(u_i)$ does not hold in general, since t_i is only congruent to u_i, not necessary equal. $\square$

This allows to prove that $\mathsf{lfp}(\psi)$ and $\mathsf{lfp}(\theta)$ coincide "up to $\equiv_E$".

Lemma 6.3.12. *Let ψ and q be as above. Then $Bq \circ (\mathsf{lfp}(\theta)) = Bq \circ (\mathsf{lfp}(\psi))$.*

Proof. We first prove that $\psi(\mathsf{lfp}(\theta)) \leq \mathsf{lfp}(\theta)$. The interesting part is to show that $\mathsf{lfp}(\theta) \circ \mu_\emptyset \circ d_{\Sigma^*\emptyset} \leq \mathsf{lfp}(\theta) \circ \kappa_\emptyset$ for any $d \in \Delta^E$, given that $\bigvee_{r \in R} \mathsf{lfp}(\theta) \circ r \circ q \circ \kappa_\emptyset \leq \mathsf{lfp}(\theta) \circ \kappa_\emptyset$ (which holds since $\mathsf{lfp}(\theta)$ is a fixed point of θ). But this is simple, given that each d acts on an argument either as the identity or by an equation in E. Thus $\psi(\mathsf{lfp}(\theta)) \leq \mathsf{lfp}(\theta)$, and since $\mathsf{lfp}(\psi)$ is the least pre-fixed point of ψ we have $\mathsf{lfp}(\psi) \leq \mathsf{lfp}(\theta)$. Hence $Bq \circ \mathsf{lfp}(\psi) \leq Bq \circ \mathsf{lfp}(\theta)$.

We proceed to show $Bq \circ \mathsf{lfp}(\theta) \leq Bq \circ \mathsf{lfp}(\psi)$ by transfinite induction; the main step is to prove that $Bq \circ h \leq Bq \circ \mathsf{lfp}(\psi)$ implies $Bq \circ \theta(h) \leq Bq \circ \mathsf{lfp}(\psi)$. So suppose $Bq \circ h \leq Bq \circ \mathsf{lfp}(\psi)$. Then

$$\begin{aligned} Bq \circ \theta(h) \circ \kappa_\emptyset &= Bq \circ (B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle h, \mathsf{id}\rangle \vee \bigvee_{r \in R} h \circ r \circ q \circ \kappa_\emptyset) \\ &= Bq \circ B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle h, \mathsf{id}\rangle \vee \bigvee_{r \in R} Bq \circ h \circ r \circ q \circ \kappa_\emptyset \end{aligned}$$

Now

$$\begin{aligned} Bq \circ B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle h, \mathsf{id}\rangle &= B\mu' \circ B\Sigma^* q \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle h, \mathsf{id}\rangle \\ &= B\mu' \circ \rho_{\Sigma^*\emptyset} \circ \Sigma(Bq \times q) \circ \Sigma\langle h, \mathsf{id}\rangle \\ &\leq B\mu' \circ \rho_{\Sigma^*\emptyset} \circ \Sigma(Bq \times q) \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle \\ &= Bq \circ B\mu_\emptyset \circ \rho_{\Sigma^*\emptyset} \circ \Sigma\langle \mathsf{lfp}(\psi), \mathsf{id}\rangle \\ &\leq Bq \circ \mathsf{lfp}(\psi) \circ \kappa_\emptyset \end{aligned}$$

where μ' is the algebra structure induced by q. The first inequality holds by assumption ($Bq \circ h \leq Bq \circ \mathsf{lfp}(\psi)$) and the second one by the fact that $\mathsf{lfp}(\psi)$ is a fixed point of ψ and by monotonicity of Bq. Moreover

$$\bigvee_{r \in R} Bq \circ h \circ r \circ q \circ \kappa_\emptyset \leq \bigvee_{r \in R} Bq \circ \mathsf{lfp}(\psi) \circ r \circ q \circ \kappa_\emptyset = Bq \circ \mathsf{lfp}(\psi) \circ \kappa_\emptyset$$

by assumption and Lemma 6.3.11. Thus $Bq \circ \theta(h) \leq Bq \circ \mathsf{lfp}(\psi)$ as desired. $\square$

This implies that $\mathsf{lfp}(\theta)$ and $\mathsf{lfp}(\psi)$ are *behaviourally equivalent* up to $\equiv_E$. Recall that behavioural equivalence coincides with bisimilarity whenever the functor B preserves weak pullbacks (Lemma 3.1.6). Under this assumption one can prove that $\mathsf{lfp}(\theta)$ is equal to $\mathsf{lfp}(\psi)$ up to bisimilarity, and by Theorem 6.2.14 we then obtain our main result of this section.

Theorem 6.3.13. *Suppose E is a set of equations which is in cfsc format w.r.t. ρ, and suppose the behaviour functor B preserves weak pullbacks. Then the interpretation $\mathsf{lfp}(\theta)$ of ρ and E equals the operational model of a certain abstract GSOS specification, up to bisimilarity (Definition 4.4.10). Bisimilarity is a congruence, and* $\mathsf{bis} \circ \mathsf{ctx} \circ \mathsf{bis}$ *is* $\mathsf{b}_{\mathsf{lfp}(\theta)}$*-compatible.*

Proof. Using the universal property of the coequalizer $q\colon \Sigma^*\emptyset \to (\Sigma^*\emptyset)/\equiv_E$, by Lemma 6.3.11 we obtain a unique coalgebra structure on $(\Sigma^*\emptyset)/\equiv_E$ turning q into a homomorphism:

$$\begin{tikzcd} \equiv_E \arrow[r, shift left, "\pi_1"] \arrow[r, shift right, "\pi_2"'] & \Sigma^*\emptyset \arrow[r, "q"] \arrow[d, "\mathsf{lfp}(\psi)"'] & (\Sigma^*\emptyset)/\equiv_E \arrow[d, dashed] \\ & B\Sigma^*\emptyset \arrow[r, "Bq"] & B(\Sigma^*\emptyset)/\equiv_E \end{tikzcd}$$

Further, by Lemma 6.3.12, q is also a homomorphism from $\mathsf{lfp}(\theta)$ into the same coalgebra. Now the pullback (in Set) of q along itself is simply $\equiv_E$, and since B preserves weak pullbacks, $\equiv_E$ is a bisimulation between $\mathsf{lfp}(\psi)$ and $\mathsf{lfp}(\theta)$ [Rut00, Theorem 4.3]. Thus, in particular, $\mathsf{lfp}(\psi)$ and $\mathsf{lfp}(\theta)$ are equal up to bisimilarity, since $\equiv_E$ is reflexive.

By Theorem 6.2.14, bisimilarity is a congruence on $\mathsf{lfp}(\psi)$. Since $\mathsf{lfp}(\psi)$ and $\mathsf{lfp}(\theta)$ are equal up to bisimilarity, it follows from Lemma 4.4.11 that bisimilarity is a congruence on $\mathsf{lfp}(\theta)$. Finally, again by Theorem 6.2.14, ctx is $\mathsf{b}_{\mathsf{lfp}(\psi)}$-compatible. Thus, by Lemma 4.4.12, $\mathsf{bis} \circ \mathsf{ctx} \circ \mathsf{bis}$ is $\mathsf{b}_{\mathsf{lfp}(\theta)}$-compatible. □

6.4 Discussion and related work

We extended Turi and Plotkin's bialgebraic approach to operational semantics with non-structural assignment rules and structural congruence, providing a general coalgebraic framework for monotone abstract GSOS with equations. Technically, our results are based on the combination of bialgebraic semantics with order. Our main result is that the interpretation of a specification involving assignment rules is well-behaved, in the sense that bisimilarity is a congruence and bisimulation-up-to techniques are sound. This result carries over to specifications with structural congruence in the cfsc format proposed in [MR05].

The main work in the literature that treats the meta-theory of rule formats with structural congruences [MR05] focuses on labelled transition systems, whereas our results apply to coalgebras in general (for behaviour functors with a complete lattice structure). Concerning transition systems, the basic rule format in [MR05] is tyft/tyxt[2], which is more expressive than positive GSOS since it allows lookahead in the premises. However, while [MR05] proves congruence of bisimilarity this does not imply the compatibility (or even soundness) of bisimulation up to

[2] In [MR05], it is sketched how to extend the results to the ntyft/ntyxt, which involves however a complicated integration of the cfsc format with the notion of stable model.

context [PS12], which we obtain in the present work (and which is, in fact, problematic in the presence of lookahead).

Plotkin proposed to model recursion by interpreting abstract GSOS in the category of complete partial orders [Plo01]. Klin [Kli04] showed that by moving to categories enriched in complete partial orders, one can interpret recursive constructs which have a similar form as our assignment rules. Technically our approach is different as it is based on an order on the behaviour functor, rather than interpreting everything in an ordered setting and using an infinite unfolding of terms, as is done in [Kli04]. Further, in [Kli04] each operator is either specified by an equation or by operational rules, disallowing a specification such as that of the parallel composition in equation (6.2).

In [LPW04], various constructions on distributive laws are presented. Example 32 of that paper discusses the definition of the parallel composition as in (6.2) above, but a general theory for structural congruence is missing. Distributive laws are applied in [Jac06b] to find solutions of guarded recursive equations. Further, in [MMS13] recursive equations are interpreted in the context of iterative algebras, where operations of interest are given by an abstract GSOS specification. That work seems to focus mainly on solutions to guarded equations, but the precise connection to the present work remains to be understood. In [BM02, CHM02], it is shown how to lift calculi with structural axioms to coalgebraic models, but under the assumption that the equations already hold.

There are several directions for future work. First, our techniques can possibly be extended to allow lookahead in premises by using cofree comonads (see, e.g., [Kli11]). While in general the combined use of cofree comonads and free monads in specifications is known to be problematic [KN14], we expect that part of these problems may be addressed by considering only positive (monotone) specifications. In fact, this could form the basis for a bialgebraic account of the tyft format. Second, in the current work we only consider free monads. One may incorporate equations which already hold, for instance by using the theory of the next chapter.

At a more fundamental level, we believe that the combination of bialgebraic semantics with ordered structures is an exciting direction of research which is yet to be explored. In the current chapter, we developed this theory only in a relatively concrete manner, by focusing on Set functors and only specifications where the syntax is given by a signature. A more abstract categorical perspective, for instance in terms of order enriched categories, could potentially clean up and generalize some of the technical development of this chapter. Such a generalization could be of interest, for instance, to study structural congruences for calculi with names.

Chapter 7

Presenting distributive laws

In the current chapter, we study distributive laws of monads over functors. These capture interaction between algebraic structure and observable behaviour in a systematic way. There are several benefits of this approach, recalled in more detail in Section 3.5: a distributive law canonically induces an algebra on the final coalgebra, provides a compositional semantics, and yields solutions to recursive equations. Moreover, distributive laws play a central role in the framework of up-to techniques introduced in the first part of this thesis.

However, concretely describing a distributive law of a monad over a functor and proving the associated axioms can be rather complicated. Instead, one may try to use general methods for constructing distributive laws from simpler ingredients. An important example of this is given by abstract GSOS, where distributive laws are represented by plain natural transformations. Further, in [HK11] it was shown how an abstract GSOS specification for a functor B can be lifted to one for the functor $(B-)^A$ which describes B-systems with input in A. Another method, which works for all monads on Set but only for certain polynomial behaviour functors B, produces a distributive law inducing a "pointwise lifting" of the algebra structure to B-behaviours [Jac06b, SBBR13].

But many examples do not fit into the above mentioned settings. A motivating example for the current chapter is that of context-free grammars, where sequential composition is not a pointwise operation and whose formal semantics satisfies the axioms of idempotent semirings, which is not a free monad. More generally, one may be interested in distributive laws involving a monad that arises as the quotient of a free monad with respect to some equations.

We give a general approach for constructing a distributive law $\lambda^{\mathcal{E}}$ for a monad $\mathcal{T}^{\mathcal{E}}$, which is presented as a quotient of a monad $\mathcal{T}$ by some equations $\mathcal{E}$, from a distributive law λ for the monad $\mathcal{T}$. In the typical application of our result, $\mathcal{T}$ is a free monad, so that λ can in turn be defined in terms of an abstract GSOS specification. Then $\lambda^{\mathcal{E}}$ is obtained as a certain quotient of λ by the equations $\mathcal{E}$, hence we say that $\lambda^{\mathcal{E}}$ *is presented by λ and the equations $\mathcal{E}$*. We show that such quotients exist when the distributive law *preserves the equations $\mathcal{E}$*, which roughly means that congruences generated by the equations are bisimulations. We also discuss how these quotients of distributive laws give rise to quotients of bialgebras, thereby giving a concrete operational interpretation. As an illustration and application, we show the existence of a distributive law of the monad for idempotent semirings over the deterministic automata functor. This result yields the equivalence between the representation of context-free languages via grammars in Greibach normal form and the coalgebraic representation via context-free expressions given in [WBR13].

Outline. In the next section, we describe in detail how to construct the quotient of a monad with respect to some given equations. In Section 7.2, we prove our main results on quotients of distributive laws. Then, in Section 7.3 we show that such quotients induce quotients of bialgebras. Finally, in Section 7.4 we discuss related work, and provide some directions for future work.

7.1 Quotients of monads

Let $\mathcal{T} = (T, \eta, \mu)$ be a monad on a category $\mathcal{C}$. For a general notion of equations on a monad, we define *$\mathcal{T}$-equations* or *equations for $\mathcal{T}$* as a 3-tuple $\mathcal{E} = (E, l, r)$ where E is an endofunctor on $\mathcal{C}$ and $l, r \colon E \Rightarrow T$ are natural transformations. The intuition is that E models the arity of the equations, i.e., the (number of) variables occurring in each equation, and l and r give the left and right-hand side. The advantage of using natural transformations (over, say, a subset of $TV \times TV$ for some set of variables V, or a generalization thereof) is that this approach defines equations on TX uniformly over any set X.

Example 7.1.1. Consider the Set functor $\Sigma X = X \times X + 1$, modelling a binary operation and a constant, which we call $+$ and 0 respectively. The (underlying functor of the) free monad Σ^* for Σ sends a set X to the terms over X built from $+$ and 0. The equations $x + 0 = x$, $x + y = y + x$ and $(x + y) + z = x + (y + z)$ can be modelled as follows. The functor E is defined as $EX = X + (X \times X) + (X \times X \times X)$. The natural transformations $l, r \colon E \Rightarrow \Sigma^*$ are given by $l_X(x) = x + 0$ and $r_X(x) = x$ for all $x \in X$; $l_X(x, y) = x + y$ and $r_X(x, y) = y + x$ for all $(x, y) \in X \times X$; $l_X(x, y, z) = x + (y + z)$ and $r_X(x, y, z) = (x + y) + z$ for all $(x, y, z) \in X \times X \times X$. This defines l_X and r_X uniformly for any set X, which makes naturality of l and r easy to prove.

A $\mathcal{T}$-algebra (X, α) is said to *satisfy* $\mathcal{E}$ if $\alpha \circ l_X = \alpha \circ r_X$:

$$EX \underset{r_X}{\overset{l_X}{\rightrightarrows}} TX \xrightarrow{\alpha} X .$$

We denote the full subcategory of $\mathcal{T}$-algebras that satisfy $\mathcal{E}$ by $(\mathcal{T}, \mathcal{E})$-Alg.

Throughout this chapter we need assumptions on $\mathcal{C}$, $\mathcal{T}$, and $\mathcal{E}$. This involves regular epis: an epi is regular if it is the coequalizer of a pair of morphisms.

Assumption 7.1.2. We assume that $\mathcal{T} = (T, \eta, \mu)$ is a monad on $\mathcal{C}$, and $E \colon \mathcal{C} \to \mathcal{C}$ is a functor such that:

1. $\mathcal{T}$-Alg has coequalizers.

2. U maps regular epis in $\mathcal{T}$-Alg to epis in $\mathcal{C}$.

3. EU and TU map regular epis in $\mathcal{T}$-Alg to epis in $\mathcal{C}$.

The first condition is needed to construct quotients of free algebras modulo equations. The second condition relates quotients of algebras (regular epis) with quotients in the base category (epis). The last condition is satisfied if condition (2) holds and E and T preserve epimorphisms in $\mathcal{C}$. If $\mathcal{C} = \mathsf{Set}$ the conditions are satisfied for any monad $\mathcal{T}$ and endofunctor E. In that case, the first condition holds since $\mathcal{T}$-Alg is cocomplete if $\mathcal{C} = \mathsf{Set}$ (see, e.g., [BW05, Proposition 3.4]), the second condition holds since U preserves regular epis if $\mathcal{C} = \mathsf{Set}$ (see the proof of [BW05, Proposition 4.6]), and the third follows from the second, since any Set functor preserves epis.

Any $\mathcal{T}$-algebra (X, α) can be turned into an algebra that satisfies the equations, by taking the coequalizer s_α of $\alpha \circ l_X^\sharp$ and $\alpha \circ r_X^\sharp$ in $\mathcal{T}$-Alg, as depicted in the following diagram:

$$(TEX, \mu_{EX}) \underset{r_X^\sharp}{\overset{l_X^\sharp}{\rightrightarrows}} (TX, \mu_X) \xrightarrow{\alpha} (X, \alpha) \xrightarrow{s_\alpha} (X/\mathcal{E}, \alpha_{\mathcal{E}}) \,. \tag{7.1}$$

Since coequalizers are unique only up to isomorphism, we choose $s_\alpha = \mathsf{id}$ for every algebra in $(\mathcal{T}, \mathcal{E})$-Alg.

In the case $\mathcal{C} = \mathsf{Set}$, the definition of s_α (7.1) implies that $\mathsf{ker}(s_\alpha)$ is the *congruence* generated by the set $E_\alpha = \{(\alpha(l_X(e)), \alpha(r_X(e)) \mid e \in EX\}$, i.e., it is the least equivalence relation on X that includes E_α and is a subalgebra of $(X, \alpha) \times (X, \alpha)$. In this sense, the kernel pair of a morphism always yields a congruence, and conversely, every congruence relation on an algebra (X, α) is the kernel of the corresponding quotient homomorphism.

In general, the coequalizer (7.1) in $\mathcal{T}$-Alg differs from the one obtained by applying the forgetful functor U and then computing the coequalizer of $\alpha \circ l_X^\sharp$ and $\alpha \circ r_X^\sharp$ in Set. The coequalizers in $\mathcal{T}$-Alg and Set coincide if the equations are reflexive in the sense that the two parallel maps $\alpha \circ l_X$ and $\alpha \circ r_X$ from EX to X have a common section, and the forgetful functor U preserves reflexive coequalizers (sections and reflexive coequalizers are recalled in Section 4.5, above Theorem 4.5.4). If T is finitary, then U preserves reflexive coequalizers. Moreover, if U preserves reflexive coequalizers then T preserves them too, but not every Set-functor preserves reflexive coequalizers [AKV00, Example 4.3].

The main step to obtain the quotient monad is to show that $(\mathcal{T}, \mathcal{E})$-Alg is a reflective subcategory of $\mathcal{T}$-Alg, meaning that the inclusion functor has a left adjoint. This left adjoint uses the coequalizer in (7.1) to map an algebra to its quotient.

Lemma 7.1.3. *The inclusion* $V\colon (\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg}$ *has a left adjoint* $H\colon \mathcal{T}\text{-}\mathsf{Alg} \to (\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg}$ *with unit* $\overline{\eta}_\alpha = s_\alpha\colon (X,\alpha) \to (X/\mathcal{E}, \alpha_\mathcal{E})$ *for all* $\alpha\colon X \to TX$ *in* $\mathcal{T}\text{-}\mathsf{Alg}$*, and counit* $\overline{\epsilon}_\alpha = \mathsf{id}$ *the identity for all* $\alpha \in (\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg}$.

Proof. We first show that for any (X,α) in $\mathcal{T}\text{-}\mathsf{Alg}$, $(X/\mathcal{E},\alpha_\mathcal{E})$ is indeed an object in $(\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg}$, i.e., it satisfies the equations. Consider the following diagram:

$$\begin{tikzcd}
TEX \arrow[dr, bend left, shift left, "l^\sharp_X"] \arrow[dr, bend left, shift right, "r^\sharp_X"'] & & \\
EX \arrow[u, "\eta_{EX}"] \arrow[r, shift left, "l_X"] \arrow[r, shift right, "r_X"'] \arrow[d, "Es_\alpha"'] & TX \arrow[r, "\alpha"] \arrow[d, "Ts_\alpha"] & X \arrow[d, "s_\alpha"] \\
E(X/\mathcal{E}) \arrow[r, shift left, "l_{X/\mathcal{E}}"] \arrow[r, shift right, "r_{X/\mathcal{E}}"'] & T(X/\mathcal{E}) \arrow[r, "\alpha_\mathcal{E}"'] & X/\mathcal{E}
\end{tikzcd}$$

The right-hand square commutes by the definition of s_α as a coequalizer in $\mathcal{T}\text{-}\mathsf{Alg}$, see (7.1). The left-hand squares (for l and r respectively) commute by naturality of l and r. The upper two paths from TEX to $X/\mathcal{E}$ commute by definition of s_α. From the above diagram we obtain $\alpha_\mathcal{E} \circ l_{X/\mathcal{E}} \circ E(s_\alpha) = \alpha_\mathcal{E} \circ r_{X/\mathcal{E}} \circ E(s_\alpha)$. Since s_α is a regular epi, by Assumption 7.1.2 it follows that $E(s_\alpha)$ is an epi, and thus $\alpha_\mathcal{E} \circ l_{X/\mathcal{E}} = \alpha_\mathcal{E} \circ r_{X/\mathcal{E}}$.

It remains to show that if $f\colon X \to Y$ is an algebra homomorphism from (X,α) to an algebra (Y,β) in $(\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg}$, then there is a unique algebra homomorphism $g\colon X/\mathcal{E} \to Y$ such that $g \circ s_\alpha = f$. Since (Y,β) satisfies the equations we know $\beta \circ l_Y = \beta \circ r_Y$, and thus the following diagram commutes:

$$\begin{tikzcd}
TEX \arrow[dr, bend left, shift left, "l^\sharp_X"] \arrow[dr, bend left, shift right, "r^\sharp_X"'] & & & \\
EX \arrow[u, "\eta_{EX}"] \arrow[r, shift left, "l_X"] \arrow[r, shift right, "r_X"'] \arrow[d, "Ef"'] & TX \arrow[r, "\alpha"] \arrow[d, "Tf"] & X \arrow[r, "s_\alpha"] \arrow[d, "f"] & X/\mathcal{E} \arrow[dl, dashed, "g"] \\
EY \arrow[r, shift left, "l_Y"] \arrow[r, shift right, "r_Y"'] & TY \arrow[r, "\beta"'] & Y &
\end{tikzcd}$$

In particular, we have $f \circ \alpha \circ l_X = f \circ \alpha \circ r_X$. Thus $f \circ \alpha \circ l^\sharp_X = f \circ \alpha \circ r^\sharp_X$, hence the desired homomorphism g arises from the universal property of the coequalizer $s_\alpha\colon (X,\alpha) \to (X/\mathcal{E},\alpha_\mathcal{E})$.

By defining $H\colon \mathcal{T}\text{-}\mathsf{Alg} \to (\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg}$ as $H(X,\alpha) = (X/\mathcal{E},\alpha_\mathcal{E})$, H is left adjoint to V, and the unit of the adjunction is $\overline{\eta} = s$. For the counit, we have $V(\epsilon_\alpha) \circ s_{V\alpha} = \mathsf{id}_{V\alpha}$, and since $s_{V\alpha} = \mathsf{id}_{V\alpha}$ then $V(\epsilon_\alpha) = \mathsf{id}_{V\alpha} = V(\mathsf{id}_\alpha)$, which implies that $\epsilon_\alpha = \mathsf{id}_\alpha$ (V is an inclusion). ◻

By composition of adjoints, the functor $UV\colon (\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg} \to \mathcal{C}$ has a left adjoint given by $X \mapsto (TX/\mathcal{E}, (\mu_X)_\mathcal{E})$. In what follows, we will write $T^\mathcal{E}X$ for $TX/\mathcal{E}$.

Definition 7.1.4 (Quotient monad)**.** Given a monad $\mathcal{T} = (T, \eta, \mu)$ on $\mathcal{C}$ and $\mathcal{T}$-equations $\mathcal{E}$, we define the *quotient monad* $\mathcal{T}^{\mathcal{E}} = (T^{\mathcal{E}}, \eta^{\mathcal{E}}, \mu^{\mathcal{E}})$ as the monad on $\mathcal{C}$ arising from the composition of the adjunction $(H, V, \overline{\eta} = s, \overline{\epsilon} = \mathsf{id})$ of Lemma 7.1.3 and the Eilenberg-Moore adjunction (G, U, η, ϵ) of $\mathcal{T}$:

$$\begin{tikzcd}
(\mathcal{T},\mathcal{E})\text{-}\mathsf{Alg} \arrow[r, bend left, "V"] \arrow[r, phantom, "\top"] & \mathcal{T}\text{-}\mathsf{Alg} \arrow[l, bend left, "H"] \arrow[r, bend left, "U"] \arrow[r, phantom, "\top"] & \mathcal{C} \arrow[l, bend left, "G"] \arrow[loop right, "T^{\mathcal{E}}"]
\end{tikzcd}$$

We define the natural transformation $q\colon T \Rightarrow T^{\mathcal{E}}$ as the family of underlying $\mathcal{C}$-arrows of s for free algebras:

$$q_X = Us_{GX} = Us_{(TX,\mu_X)}\colon TX \to T^{\mathcal{E}}X \tag{7.2}$$

The next result summarizes what we need to know about q and the quotient monad.

Theorem 7.1.5. *Let $\mathcal{T}^{\mathcal{E}} = (T^{\mathcal{E}}, \eta^{\mathcal{E}}, \mu^{\mathcal{E}})$ be the quotient monad associated to a monad $\mathcal{T} = (T, \eta, \mu)$ on $\mathcal{C}$ with $\mathcal{T}$-equations $\mathcal{E}$. Define the natural transformation q as in* (7.2)*, so $q\colon T \Rightarrow \mathcal{T}^{\mathcal{E}}$ is defined on an object X as the coequalizer of $\mu_X \circ l^\sharp_{TX}$ and $\mu_X \circ r^\sharp_{TX}$:*

$$(TETX, \mu_{ETX}) \underset{r^\sharp_{TX}}{\overset{l^\sharp_{TX}}{\rightrightarrows}} (TTX, \mu_{TX}) \xrightarrow{\mu_X} (TX, \mu_X) \xrightarrow{q_X} (T^{\mathcal{E}}X, (\mu_X)_{\mathcal{E}})\,.$$

Then

1. *the components of q (as well as Tq and Eq) are epimorphisms in the underlying category $\mathcal{C}$,*
2. *the unit of the quotient monad is given by $\eta^{\mathcal{E}} = q \circ \eta$ and*
3. *q is a monad morphism from $\mathcal{T}$ to $\mathcal{T}^{\mathcal{E}}$.*

Proof. The first item follows from Assumption 7.1.2. For the second item, we have $\eta^{\mathcal{E}} = UsG \circ \eta = q \circ \eta$. The third item is proved below in Corollary 7.1.7. □

Next, we show that q is indeed a monad morphism from $\mathcal{T}$ to $\mathcal{T}^{\mathcal{E}}$. One way of doing so is to show that q is a coequalizer in the category of monads and monad morphisms. Kelly studied colimits in categories of monads, and proved their existence in the context of a certain adjunction [Kel80, Proposition 26.4]; with a bit of effort one can instantiate this to the adjunction constructed above. For a self-contained presentation in this section, we do not invoke Kelly's results but instead prove directly the part that shows the existence of a monad morphism. This is instantiated below to the adjunction of the quotient monad.

Lemma 7.1.6. *Let $\mathcal{A}$ be any subcategory of $\mathcal{T}$-*Alg*, and suppose the forgetful functor $U\colon \mathcal{A} \to \mathcal{C}$ has a left adjoint F, with unit and counit denoted by η' and ϵ' respectively. Then*

1. *F induces a natural transformation $\kappa\colon TUF \Rightarrow UF$ so that $\kappa \circ T\eta' \colon T \Rightarrow UF$ is a monad morphism.*

2. *Precomposing the functor UF-*Alg* $\to \mathcal{T}$-*Alg* induced by this monad morphism with the comparison functor $\mathcal{A} \to UF$-*Alg* yields the inclusion $\mathcal{A} \to \mathcal{T}$-*Alg*.*

The relevant categories and functors are summarized in the diagram below, where the functors in the right-hand triangle are given by 2 above (hence, this triangle commutes).

$$\begin{tikzcd}
\mathcal{C} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{A} \arrow[l, bend left, "U"] \arrow[r] \arrow[d, hook] & UF\text{-}\mathsf{Alg} \arrow[dl] \\
& \mathcal{T}\text{-}\mathsf{Alg} \arrow[ul] &
\end{tikzcd}$$

Proof. The functor F sends any $\mathcal{C}$-object X to a T-algebra structure on UFX; we define κ_X to be that algebra structure. Naturality of κ is immediate since Ff is a $\mathcal{T}$-algebra homomorphism for any $\mathcal{C}$-arrow f. To see that $\kappa \circ T\eta'$ is a monad morphism, we have to prove that the outside of the following diagram commutes:

$$\begin{tikzcd}
TT \arrow[r, Rightarrow, "TT\eta'"] \arrow[d, Rightarrow, "\mu"'] & TTUF \arrow[r, Rightarrow, "T\kappa"] \arrow[d, Rightarrow, "\mu UF"] & TUF \arrow[r, Rightarrow, "T\eta' UF"] \arrow[d, Rightarrow, "\kappa"] & TUFUF \arrow[d, Rightarrow, "\kappa UF"] \\
T \arrow[r, Rightarrow, "T\eta'"] & TUF \arrow[r, Rightarrow, "\kappa"] & UF & UFUF \arrow[l, Rightarrow, "\mu'"'] \\
\mathsf{Id} \arrow[r, Rightarrow, "\eta'"'] \arrow[u, Rightarrow, "\eta"] & UF \arrow[u, Rightarrow, "\eta UF"'] \arrow[ur, equal] & &
\end{tikzcd}$$

Here $\mu' = U\epsilon' F$ is the multiplication of the monad that arises from the adjunction $F \dashv U$. The top left square commutes by naturality and the middle square since any component of κ is a $\mathcal{T}$-algebra. For the right-hand square we have

$$\kappa = \kappa \circ TU\epsilon' F \circ T\eta' UF = U\epsilon' F \circ \kappa UF \circ T\eta' UF = \mu' \circ \kappa UF \circ T\eta' UF$$

where the first equality follows from the triangle identity $\mathsf{id}UF = U\epsilon' F \circ \eta' UF$ (and functoriality), and the second from the fact that, for any X, ϵ'_{FX} is a $\mathcal{T}$-algebra homomorphism from κ_{UFX} to κ_X. The bottom left square commutes by naturality, and the triangle since κ is an $\mathcal{T}$-algebra.

For item 2 of the statement of the theorem, we first note that the composite functor under consideration maps any T-algebra (A, α) in $\mathcal{A}$ to $U\epsilon'_{(A,\alpha)} \circ \kappa_A \circ T\eta'_A$.

But the following diagram commutes:

$$\begin{tikzcd} TA \arrow[r, "T\eta'_A"] \arrow[dr, equal] & TUFA \arrow[r, "\kappa_A"] \arrow[d, "TU\epsilon'_{(A,\alpha)}"] & UFA \arrow[d, "U\epsilon'_{(A,\alpha)}"] \\ & TA \arrow[r, "\alpha"'] & A \end{tikzcd}$$

by a triangle identity and the fact that $\epsilon'_{(A,\alpha)}$ is an algebra morphism. Hence $U\epsilon'_{(A,\alpha)} \circ \kappa_A \circ T\eta'_A = \alpha$, which means that the composite functor under consideration indeed coincides with the inclusion. □

Corollary 7.1.7. *Item 3 of Theorem 7.1.5 holds: $q\colon T \Rightarrow T^{\mathcal{E}}$ is a monad morphism.*

Proof. By Lemma 7.1.6, we only need to show that q coincides with $\kappa \circ T\eta^{\mathcal{E}}$, where $\eta^{\mathcal{E}}$ is the unit of the quotient monad. To this end, consider the following diagram:

$$\begin{tikzcd} T \arrow[r, Rightarrow, "T\eta^{\mathcal{E}}"] \arrow[dr, Rightarrow, "T\eta"'] & TT^{\mathcal{E}} \arrow[r, Rightarrow, "\kappa"] & T^{\mathcal{E}} \\ & TT \arrow[u, Rightarrow, "Tq"'] \arrow[r, Rightarrow, "\mu"'] & T \arrow[u, Rightarrow, "q"'] \end{tikzcd}$$

Commutativity of the triangle holds by Theorem 7.1.5. For the square, notice that the components of κ are simply the quotient algebras as constructed in the proof of Lemma 7.1.3, and q is an algebra morphism by construction. □

Remark 7.1.8. The monad morphism $q\colon T \Rightarrow T^{\mathcal{E}}$ induces a functor

$$\mathcal{T}^{\mathcal{E}}\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg}.$$

By Lemma 7.1.6 (2), the comparison functor $(\mathcal{T}, \mathcal{E})\text{-}\mathsf{Alg} \to \mathcal{T}^{\mathcal{E}}\text{-}\mathsf{Alg}$ followed by the functor $\mathcal{T}^{\mathcal{E}}\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg}$ coincides with the inclusion $(\mathcal{T}, \mathcal{E})\text{-}\mathsf{Alg} \to \mathcal{T}\text{-}\mathsf{Alg}$.

The above construction yields a monad $\mathcal{T}^{\mathcal{E}}$ given a set of operations and equations. Intuitively, any monad which is isomorphic to $\mathcal{T}^{\mathcal{E}}$ is presented by these same operations and equations; this is captured by the following definition.

Definition 7.1.9. Let Σ be an endofunctor on $\mathcal{C}$, Σ^* the free monad for Σ, and $\mathcal{T}^{\mathcal{E}}$ the quotient monad of Σ^* with respect to some Σ^*-equations $\mathcal{E}$. A monad $\mathcal{K} = (K, \theta, \nu)$ is *presented by* Σ *and* $\mathcal{E}$ if there is a monad isomorphism $i\colon (\Sigma^*)^{\mathcal{E}} \Rightarrow K$.

Example 7.1.10. The *idempotent semiring monad* is defined by the Set endofunctor that maps a set X to the set $\mathcal{P}_\omega(X^*)$ of finite languages over X and a function $f\colon X \to Y$ to $\mathcal{P}_\omega(f^*)(L) = \bigcup\{f(x_1)\cdots f(x_n) \mid x_1\cdots x_n \in L\}$. The unit $\eta_X : X \to \mathcal{P}_\omega(X^*)$ and the multiplication $\mu_X : \mathcal{P}_\omega(\mathcal{P}_\omega(X^*)^*) \to \mathcal{P}_\omega(X^*)$ are given by

$$\begin{aligned} \eta_X(x) &= \{x\}, \\ \mu_X(\mathcal{L}) &= \textstyle\bigcup_{L_1\cdots L_n \in \mathcal{L}}\{w_1\cdots w_n \mid w_i \in L_i\}. \end{aligned}$$

Consider the functor Σ and equations $\mathcal{E}$ for the free monad Σ^*, where

$$\Sigma X = X \times X + X \times X + 1 + 1$$

models two binary operators (to represent addition $+$ and multiplication $\cdot$) and two constants (to represent 0 and 1). The equations $\mathcal{E}$ for Σ^* are given by the idempotent semiring axioms. We obtain a quotient monad $(\Sigma^*)^{\mathcal{E}}$, and by Theorem 7.1.5 a monad morphism:

$$q \colon \Sigma^* \Rightarrow (\Sigma^*)^{\mathcal{E}}\,.$$

Since we have chosen $\mathcal{E}$ to be the idempotent semiring axioms, we have a monad isomorphism $(\Sigma^*)^{\mathcal{E}} \cong \mathcal{P}_\omega(\mathsf{Id}^*)$ (using these equations, every term is equivalent to a sum of products of variables). Thus, the monad $\mathcal{P}_\omega(\mathsf{Id}^*)$ is presented by the (semiring) signature Σ and the axioms for idempotent semirings.

7.2 Quotients of distributive laws

In the previous section, we saw how equations give rise to quotients of algebras, and we gave a construction of the resulting quotient monad. Next, we investigate conditions under which distributive laws and equations give rise to quotients of distributive laws.

7.2.1 Distributive laws over plain behaviour functors

Let $\lambda \colon TB \Rightarrow BT$ be a distributive law of a monad $\mathcal{T} = (T, \eta, \mu)$ over a (plain) behaviour functor B (Section 3.5). Given equations $\mathcal{E} = (E, l, r)$ for $\mathcal{T}$ we provide a condition on λ and $\mathcal{E}$ that ensures that we get a distributive law $\lambda^{\mathcal{E}} \colon T^{\mathcal{E}}B \Rightarrow BT^{\mathcal{E}}$ of the quotient monad over B. We use the notion of morphisms of distributive laws from [PW02, Wat02].

Definition 7.2.1. Let $\mathcal{T} = (T, \eta, \mu)$ and $\mathcal{K} = (K, \theta, \nu)$ be monads, and let $\lambda \colon TB \Rightarrow BT$ and $\kappa \colon KB \Rightarrow BK$ be distributive laws of $\mathcal{T}$ and $\mathcal{K}$ over B. A natural transformation $\tau \colon T \Rightarrow K$ is a *morphism of distributive laws* from λ to κ if τ is a monad morphism and the following square commutes:

$$\begin{tikzcd} TB \arrow[r, Rightarrow, "\tau B"] \arrow[d, Rightarrow, "\lambda"'] & KB \arrow[d, Rightarrow, "\kappa"] \\ BT \arrow[r, Rightarrow, "B\tau"'] & BK \end{tikzcd} \tag{7.3}$$

There are generalizations of the above definition that allow natural transformations between behaviour functors [Wat02]. For our purposes, we do not need to change the behaviour type.

Definition 7.2.2. We say that $\lambda \colon TB \Rightarrow BT$ *preserves (equations in)* $\mathcal{E}$ if for every object X in $\mathcal{C}$:

$$EBX \overset{l_{BX}}{\underset{r_{BX}}{\rightrightarrows}} TBX \xrightarrow{\lambda_X} BTX \xrightarrow{Bq_X} BT^{\mathcal{E}}X \tag{7.4}$$

commutes.

In Set, preservation of equations can be conveniently formulated in terms of relation lifting (Section 3.2.1).

Lemma 7.2.3. *Suppose* $B \colon \mathsf{Set} \to \mathsf{Set}$ *preserves weak pullbacks. Denote by* $\equiv_X$ *the congruence* $\mathsf{ker}(q_X)$ *on* TX *generated by the equations. Then* λ *preserves* $\mathcal{E}$ *if and only if for every set* X *and every* $b \in EBX$*:*

$$(\lambda_X(l_{BX}(b)), \lambda_X(r_{BX}(b))) \in \mathsf{Rel}(B)(\equiv_X)\,. \tag{7.5}$$

Proof. By Lemma 3.2.4:

- $\mathsf{Rel}(B)$ preserves diagonal relations, i.e., $\mathsf{Rel}(B)(\Delta_X) = \Delta_{BX}$, and
- $\mathsf{Rel}(B)$ preserves inverse images, since B preserves weak pullbacks.

Hence $\mathsf{Rel}(B)$ preserves kernel relations (cf. [Jac12, Lemma 3.2.5(i)]):

$$\begin{array}{lll}
\mathsf{Rel}(B)(\equiv_X) & = \mathsf{Rel}(B)(\mathsf{ker}(q_X)) & \text{definition } \equiv_X \\
& = \mathsf{Rel}(B)((q_X \times q_X)^{-1}(\Delta_X)) & \\
& = (Bq_X \times Bq_X)^{-1}(\mathsf{Rel}(B)(\Delta_X)) & \mathsf{Rel}(B) \text{ pres. inverse images} \\
& = (Bq_X \times Bq_X)^{-1}(\Delta_{BX}) & \mathsf{Rel}(B) \text{ pres. diagonals} \\
& = \mathsf{ker}(Bq_X) &
\end{array}$$

Thus, the condition from the statement of the lemma is satisfied if and only if for every X and every $b \in EBX$ we have

$$(\lambda_X(l_{BX}(b)), \lambda_X(r_{BX}(b))) \in \mathsf{ker}(Bq_X)$$

which coincides with preservation of equations. □

We now come to the main result of this chapter. It shows how to obtain a distributive law for the quotient monad under the assumption of preservation of equations. Preservation of equations can be proved by explicit calculations, as shown in several examples below.

Theorem 7.2.4. *If* $\lambda \colon TB \Rightarrow BT$ *preserves equations* $\mathcal{E}$ *then there is a (unique) distributive law* $\lambda^{\mathcal{E}} \colon T^{\mathcal{E}}B \Rightarrow BT^{\mathcal{E}}$ *such that* $q \colon T \Rightarrow T^{\mathcal{E}}$ *is a morphism of distributive laws from* λ *to* $\lambda^{\mathcal{E}}$.

Proof. Suppose λ preserves equations $\mathcal{E}$. We first prove that the top rows of the following diagram commute:

$$
\begin{tikzcd}
TETBX \arrow[r, shift left, "l^{\sharp}_{TBX}"] \arrow[r, shift right, "r^{\sharp}_{TBX}"'] & TTBX \arrow[r, "\mu_{BX}"] & TBX \arrow[r, "\lambda_X"] & BTX \arrow[r, "Bq_X"] & BT^{\mathcal{E}}X \\
ETBX \arrow[u, "\eta_{ETBX}"] \arrow[ur, bend left, "l_{TBX}"] \arrow[ur, bend right, "r_{TBX}"'] & & & &
\end{tikzcd}
\tag{7.6}
$$

In order to do so, we prove that

1. $Bq_X \circ \lambda_X$ is an algebra morphism from (TBX, μ_{BX}), and
2. the bottom two paths, i.e., from $ETBX$ to $BT^{\mathcal{E}}$, commute.

Commutativity of the top rows then follows from the fact that homomorphic extensions are unique.

For the first item, consider the following diagram:

$$
\begin{tikzcd}
TTBX \arrow[r, "T\lambda_X"] \arrow[dd, "\mu_{BX}"'] & TBTX \arrow[r, "TBq_X"] \arrow[d, "\lambda_{TX}"] & TBT^{\mathcal{E}}X \arrow[d, "\lambda_{T^{\mathcal{E}}X}"] \\
 & BTTX \arrow[r, "BTq_X"] \arrow[d, "B\mu_X"] & BTT^{\mathcal{E}}X \arrow[d, "B(\mu_X)_{\mathcal{E}}"] \\
TBX \arrow[r, "\lambda_X"'] & BTX \arrow[r, "Bq_X"'] & BT^{\mathcal{E}}X
\end{tikzcd}
$$

The rectangle (on the left) is the multiplication law for λ, which holds since λ is a distributive law of $\mathcal{T}$ over B (Section 3.5.1). The upper right square commutes by naturality, the lower by the fact that q_X is an algebra morphism.

For the second item, we need to prove that the top two rows in the following diagram commute:

$$
\begin{tikzcd}
ETBX \arrow[r, shift left, "l_{TBX}"] \arrow[r, shift right, "r_{TBX}"'] \arrow[d, "E\lambda_X"'] \arrow[dr, phantom, "\text{(nat. } l, r\text{)}"] & TTBX \arrow[r, "\mu_{BX}"] \arrow[d, "T\lambda_X"] \arrow[dr, phantom, "\text{(mult. } \lambda\text{)}"] & TBX \arrow[r, "\lambda_X"] & BTX \arrow[r, "Bq_X"] \arrow[dr, phantom, "(q \text{ monad morphism})"] & BT^{\mathcal{E}}X \\
EBTX \arrow[r, shift left, "l_{BTX}"] \arrow[r, shift right, "r_{BTX}"'] & TBTX \arrow[r, "\lambda_{TX}"'] & BTTX \arrow[r, "Bq_{TX}"'] \arrow[ur, "B\mu_X"] & BT^{\mathcal{E}}TX \arrow[r, "BT^{\mathcal{E}}q_X"'] & BT^{\mathcal{E}}T^{\mathcal{E}}X \arrow[u, "B\mu^{\mathcal{E}}_X"']
\end{tikzcd}
$$

The two squares on the left (for l, r respectively) commute by naturality of l and r. The two other shapes commute by the multiplication law of λ and the fact that q is a monad morphism (Corollary 7.1.7). The crucial point is that the two paths from $EBTX$ to $BT^{\mathcal{E}}TX$ commute by the assumption that λ preserves $\mathcal{E}$ (instantiated to the object TX). It follows that the top rows commute, as desired.

Thus, we have shown that (7.6) commutes. By the universal property of the coequalizer q_{BX} we obtain $\lambda^{\mathcal{E}}_X$:

$$\begin{tikzcd}
TETBX \arrow[r, shift left, "l^\sharp_{TBX}"] \arrow[r, shift right, "r^\sharp_{TBX}"'] & TTBX \arrow[r, "\mu_{BX}"] & TBX \arrow[r, "q_{BX}"] \arrow[d, "\lambda_X"] & T^{\mathcal{E}}BX \arrow[d, dashed, "\lambda^{\mathcal{E}}_X"] \\
 & & BTX \arrow[r, "Bq_X"] & BT^{\mathcal{E}}X
\end{tikzcd} \tag{7.7}$$

Naturality of $\lambda^{\mathcal{E}}$ follows from (7.7), naturality of λ and q, and the fact that q is componentwise epic in the underlying category $\mathcal{C}$ (Theorem 7.1.5). Due to the commutativity of the square in (7.7), q is a morphism of distributive laws from λ to $\lambda^{\mathcal{E}}$ once we show that $\lambda^{\mathcal{E}}$ is, in fact, a distributive law of monad over functor (Section 3.5.1).

The unit law for $\lambda^{\mathcal{E}}$ holds due to the unit law for λ, (7.7) and the fact that $\eta^{\mathcal{E}} = q \circ \eta$ (Theorem 7.1.5):

$$\begin{tikzcd}
BX \arrow[r, "\eta_{BX}"] \arrow[rr, bend left, "\eta^{\mathcal{E}}_{BX}"] \arrow[d, equal] & TBX \arrow[r, "q_{BX}"] \arrow[d, "\lambda_X"] \arrow[dr, phantom, "(7.7)"] & T^{\mathcal{E}}BX \arrow[d, "\lambda^{\mathcal{E}}_X"] \\
BX \arrow[r, "B\eta_X"'] \arrow[rr, bend right, "B\eta^{\mathcal{E}}_X"'] & BTX \arrow[r, "Bq_X"'] & BT^{\mathcal{E}}X
\end{tikzcd} \tag{7.8}$$

Multiplication law for $\lambda^{\mathcal{E}}$:

$$\begin{tikzcd}
TBX \arrow[rr, "\lambda_X"] \arrow[dddd, bend right, "q_{BX}"'] \arrow[drr, phantom, "(\text{mult. } \lambda)"] & & BTX \arrow[dddd, bend left, "Bq_X"] \\
TTBX \arrow[u, "\mu_{BX}"] \arrow[r, "T\lambda_X"] \arrow[d, "q_{TBX}"] \arrow[dr, phantom, "(\text{nat. } q)"] & TBTX \arrow[r, "\lambda_{TX}"] \arrow[d, "q_{BTX}"] \arrow[dr, phantom, "(7.7)"] & BTTX \arrow[u, "B\mu_X"'] \arrow[d, "Bq_{TX}"] \\
T^{\mathcal{E}}TBX \arrow[r, "T^{\mathcal{E}}\lambda_X"] \arrow[d, "T^{\mathcal{E}}q_{BX}"] \arrow[dr, phantom, "(7.7)"] & T^{\mathcal{E}}BTX \arrow[r, "\lambda^{\mathcal{E}}_{TX}"] \arrow[d, "T^{\mathcal{E}}Bq_X"] \arrow[dr, phantom, "(\text{nat. } \lambda^{\mathcal{E}})"] & BT^{\mathcal{E}}TX \arrow[d, "BT^{\mathcal{E}}q_X"] \\
T^{\mathcal{E}}T^{\mathcal{E}}BX \arrow[r, "T^{\mathcal{E}}\lambda^{\mathcal{E}}_X"] \arrow[d, "\mu^{\mathcal{E}}_{BX}"] & T^{\mathcal{E}}BT^{\mathcal{E}}X \arrow[r, "\lambda^{\mathcal{E}}_{T^{\mathcal{E}}X}"] & BT^{\mathcal{E}}T^{\mathcal{E}}X \arrow[d, "B\mu^{\mathcal{E}}_X"'] \\
T^{\mathcal{E}}BX \arrow[rr, "\lambda^{\mathcal{E}}_X"] & & BT^{\mathcal{E}}X
\end{tikzcd} \tag{7.9}$$

The small upper-left square commutes by naturality of q. The small lower-left square commutes by applying $T^{\mathcal{E}}$ to (7.7). The outer crescents commute since q is a monad morphism, and the outermost part does due to (7.7). Finally, we use that by naturality of q, $T^{\mathcal{E}} q_{BX} \circ q_{TBX} = q_{T^{\mathcal{E}}BX} \circ Tq_{BX}$, which by Theorem 7.1.5 is an epi, and hence can be right-cancelled to yield commutativity of the lower rectangle as desired. □

Remark 7.2.5. Every distributive law uniquely corresponds to a functor lifting on $\mathcal{T}$-algebras. The distributive law $\lambda^{\mathcal{E}}$ in Theorem 7.2.4 exists if and only if the lifting restricts to $\mathcal{T}^{\mathcal{E}}$-algebras. A similar statement for the case when B is a monad is made in [MM07, Corollary 3.4.2].

As a corollary we obtain the analogue of Theorem 7.2.4 for monads presented by operations and equations.

Corollary 7.2.6. *Suppose $\mathcal{K} = (K, \theta, \nu)$ is a monad that is presented by operations Σ and equations $\mathcal{E}$ with a monad isomorphism $i\colon T^{\mathcal{E}} \Rightarrow K$, and suppose we have a distributive law $\lambda\colon \Sigma^* B \Rightarrow B\Sigma^*$ of Σ^* over B that preserves $\mathcal{E}$. Then there exists a unique distributive law $\kappa\colon KB \Rightarrow BK$ of $\mathcal{K}$ over B such that $i \circ q\colon \lambda \Rightarrow \kappa$ is a morphism of distributive laws.*

Proof. By Theorem 7.2.4 we obtain a distributive law $\lambda^{\mathcal{E}}$ of $\mathcal{T}^{\mathcal{E}}$ over B. The distributive law $\kappa\colon KB \Rightarrow BK$ is defined as $\kappa = Bi \circ \lambda^{\mathcal{E}} \circ i^{-1}$. The proof proceeds by checking that κ indeed satisfies the defining axioms of a distributive law, which is an easy but tedious exercise. □

Theorem 7.2.4 states that if λ preserves the equations $\mathcal{E}$, then we can *present* $\lambda^{\mathcal{E}}$ *as "λ modulo equations"*. We illustrate this with an example.

Example 7.2.7 (Stream calculus)**.** Behavioural differential equations are used to define streams and stream operations (Section 3.1.1). We define the following system of behavioural differential equations:

$$\begin{array}{ll} (\sigma \times \tau)_0 = \sigma_0 \cdot \tau_0 & (\sigma \times \tau)' = (\sigma' \times [\tau_0]) + ((\sigma' \times (\mathtt{X} \times \tau')) + ([\sigma_0] \times \tau')) \\ \mathtt{X}_0 = 0 & \mathtt{X}' = [1] \end{array}$$

where the sum $+$ and the constants $[r] = (r, 0, 0, \ldots)$ for all $r \in \mathbb{R}$, are as defined in Section 3.1.1. The operation $\times$ is the convolution product, defined differently here than in Section 3.1.1; we explain this choice at the end of the example.

Since we are defining two binary operations ($+$ and $\times$), one constant stream $\mathtt{X}$ and $\mathbb{R}$ many streams $[r]$, the signature under consideration is $\Sigma X = X \times X + X \times X + 1 + \mathbb{R}$. The differential equations can be modelled as a natural transformation $\rho\colon \Sigma(\mathbb{R} \times \mathsf{Id}) \Rightarrow \mathbb{R} \times \Sigma^*$, where Σ^* is the free monad for Σ. On a component X, ρ is given by:

$$\begin{array}{rcl} \rho_X^{[r]} & = & (r, [0]) \\ \rho_X^{\mathtt{X}} & = & (0, [1]) \\ \rho_X^{+}((a, x), (b, y)) & = & (a + b, x + y) \\ \rho_X^{\times}((a, x), (b, y)) & = & (a \cdot b, (x \times [b]) + ((x \times (\mathtt{X} \times y)) + ([a] \times y))) \end{array}$$

This differs from Example 3.5.5, where we considered GSOS specifications, which are of a slightly different form, involving the copointed functor $(\mathbb{R} \times \mathsf{Id}) \times \mathsf{Id}$ on the left-hand side. Similar to what is described for GSOS specifications in Section 3.5.2, the above natural transformation ρ induces a distributive law $\rho^\dagger \colon \Sigma^*(\mathbb{R} \times \mathsf{Id}) \Rightarrow (\mathbb{R} \times \mathsf{Id})\Sigma^*$.

Let $\mathcal{E}$ be given by the following axioms where v, u, w are variables and $a, b \in \mathbb{R}$ (see Example 7.1.1 for an explanation of how this corresponds to a functor with two natural transformations):

$$\begin{array}{lll}
(v + u) + w = v + (u + w) & [0] + v = v & v + u = u + v \\
(v \times u) \times w = v \times (u \times w) & [1] \times v = v & v \times u = u \times v \\
v \times (u + w) = (v \times u) + (v \times w) & [0] \times v = [0] & \\
[a + b] = [a] + [b] & [a \cdot b] = [a] \times [b] &
\end{array}$$

$\mathcal{E}$ consists of the *commutative* semiring axioms together with axioms stating the inclusion of the underlying semiring of the reals. We would like to apply Theorem 7.2.4 to obtain a distributive law for the quotient monad arising from Σ^* and $\mathcal{E}$. To this end, we show that $\rho^\dagger$ preserves $\mathcal{E}$. Let $(a, x), (b, y), (c, z) \in \mathbb{R} \times X$ for some set X. First note that $(r_1, t_1)\ \mathsf{Rel}(\mathbb{R} \times \mathsf{Id})(\equiv_X)\ (r_2, t_2)$ iff $r_1 = r_2$ and $t_1 \equiv_X t_2$. It is straightforward to check preservation of the axioms that only concern addition, as well as of $[1] \times v = v$, $[0] \times v = [0]$ and $v \times u = u \times v$. We show that $[a \cdot b] = [a] \times [b]$ is preserved:

$$\begin{array}{lcl}
\rho^\dagger_X([a] \times [b]) & = & (a \cdot b, [0] \times [b] + [0] \times \mathtt{X} \times [0] + [a] \times [0]) \\
& \mathsf{Rel}(\mathbb{R} \times \mathsf{Id})(\equiv_X) & (a \cdot b, [0]) \\
& = & \rho^\dagger_X([a \cdot b])
\end{array}$$

We check that $\rho^\dagger$ preserves the distribution axiom:

$$\begin{array}{ll}
\multicolumn{2}{l}{\rho^\dagger_X((a, x) \times ((b, y) + (c, z)))} \\
= & (a \cdot (b + c), (x \times [b + c]) + (x \times X \times (y + z)) + [a] \times (y + z)) \\
\mathsf{Rel}(\mathbb{R} \times \mathsf{Id})(\equiv_X) & (a \cdot (b + c), (x \times [b + c]) + (x \times X \times y) + (x \times X \times z) + \\
& \qquad ([a] \times y) + ([a] \times z)) \\
\mathsf{Rel}(\mathbb{R} \times \mathsf{Id})(\equiv_X) & ((a \cdot c) + (b \cdot c), (x \times [b]) + (x \times X \times y) + ([a] \times y) + \\
& \qquad (x \times [c]) + (x \times X \times z) + ([a] \times z)) \\
= & \rho^\dagger_X(((a, x) \times (b, y)) + ((a, x) \times (c, z)))
\end{array}$$

Note that we used $[a + b] = [a] + [b]$. Similarly, preservation of $\times$-associativity can be verified, and it uses the axiom $[a \cdot b] = [a] \times [b]$. We have thus shown that $\rho^\dagger$ preserves $\mathcal{E}$, and by Theorem 7.2.4 we obtain a distributive law of the quotient monad over $\mathbb{R} \times \mathsf{Id}$.

The derivative of the convolution product is usually given differently than we defined it above. However, with the usual definition (Section 3.1.1), we did not manage to show that the commutativity of $\times$ is preserved although all other axioms remain preserved. However, the convolution product (interpreted in the final coalgebra) *is* commutative. This suggests that, even if a given set of equations holds in (the algebra induced by the distributive law on) the final coalgebra, these equations are not necessarily preserved (cf. Example 7.2.9 below).

In the above example, we did not have a specific monad in mind; we simply considered a free monad and a set of equations. In Example 7.2.11 below, we give an example for the idempotent semiring monad.

Remark 7.2.8. The concrete proof method for preservation of equations bears a close resemblance to *bisimulation up to congruence* as presented in Chapters 2,4 and 5, since one must show that for every pair in the (image of the) equations, its derivatives are related by the least *congruence* $\equiv_X$ instead of just the equivalence relation induced by the equations.

Example 7.2.9. In this example we show that it is not always possible to show that a given λ preserves a given equation that holds in the final coalgebra. Again, we consider stream systems, i.e., coalgebras for the functor $BX = \mathbb{R} \times X$. We define the constant stream of zeros by three different constants n_1, n_2 and n_3 by the following behavioural differential equations:

$$\begin{array}{rclcrcl} n_1(0) & = & 0 & \quad & n_1' & = & n_1 \\ n_2(0) & = & 0 & & n_2' & = & n_3 \\ n_3(0) & = & 0 & & n_3' & = & n_3 \end{array}$$

The corresponding signature functor is $\Sigma X = 1+1+1$, and the above specification gives rise to a distributive law $\lambda\colon \Sigma^* B \Rightarrow B\Sigma^*$. Now consider the equation $n_1 = n_2$; this clearly holds when interpreted in the final coalgebra. However, this equation is not preserved by λ. To see this, notice that $\lambda(n_1) = (0, n_1)$ and $\lambda(n_2) = (0, n_3)$, but $n_1 \not\equiv_X n_3$, so $\lambda(n_1)$ and $\lambda(n_2)$ are not related by $\mathsf{Rel}(B)(\equiv_X)$.

7.2.2 Distributive laws over copointed functors

We now show that the main result of this chapter also holds for distributive laws over copointed functors. This extends our method to deal with operations specified in the abstract GSOS format (Section 3.5.2).

Proposition 7.2.10. *Theorem 7.2.4 and Corollary 7.2.6 hold as well for any distributive law of a monad over a copointed functor.*

Proof. Let (B, ϵ) be a copointed functor and $\lambda\colon TB \Rightarrow BT$ a distributive law of $\mathcal{T}$ over (B, ϵ). Suppose λ preserves equations $\mathcal{E}$. Then by Theorem 7.2.4 there is a distributive law $\lambda^{\mathcal{E}}$ of $\mathcal{T}^{\mathcal{E}}$ over B such that $q\colon T \Rightarrow T^{\mathcal{E}}$ is a morphism of distributive laws. In order to show that $\lambda^{\mathcal{E}}$ is a distributive law of $\mathcal{T}^{\mathcal{E}}$ over (B, ϵ), we only need to prove that $\lambda^{\mathcal{E}}$ satisfies the additional axiom, i.e., that the right-hand crescent in the following diagram commutes:

$$\begin{tikzcd}
TBX \arrow[r, "q_{BX}"] \arrow[d, "\lambda_X"] \arrow[dd, bend right=60, "T\epsilon_X"'] & T^{\mathcal{E}}BX \arrow[d, "\lambda^{\mathcal{E}}_X"'] \arrow[dd, bend left=60, "T^{\mathcal{E}}\epsilon_X"] \\
BTX \arrow[r, "Bq_X"] \arrow[d, "\epsilon_{TX}"] & BT^{\mathcal{E}}X \arrow[d, "\epsilon_{T^{\mathcal{E}}X}"'] \\
TX \arrow[r, "q_X"'] & T^{\mathcal{E}}X
\end{tikzcd}$$

The outermost part commutes by naturality of q, the upper square commutes since q is a morphism of distributive laws, the lower square commutes commute by naturality of ϵ, and the left crescent commutes by the fact that λ is a distributive law of $\mathcal{T}$ over (B,ϵ). Consequently we have $\epsilon_{T^{\mathcal{E}}X} \circ \lambda^{\mathcal{E}}_X \circ q_{BX} = T^{\mathcal{E}}\epsilon_X \circ q_{BX}$, and since q_{BX} is an epi (Theorem 7.1.5) we obtain $\epsilon_{T^{\mathcal{E}}X} \circ \lambda^{\mathcal{E}}_X = T^{\mathcal{E}}\epsilon_X$ as desired. □

Example 7.2.11 (Context-free languages)**.** A context free grammar (in Greibach normal form) consists of a finite set A of terminal symbols, a (finite) set X of non-terminal symbols, and a map $\langle o,t\rangle\colon X \to 2 \times \mathcal{P}_\omega(X^*)^A$, i.e., it is a coalgebra for the behaviour functor $B = 2 \times \mathsf{Id}^A$ composed with the idempotent semiring monad $\mathcal{P}_\omega(\mathsf{Id}^*)$ from Example 7.1.10. Intuitively, $o(x) = 1$ means that the variable x can generate the empty word, whereas $w \in t(x)(a)$ if and only if x can generate aw (see [WBR13, Win14]).

It is a rather difficult task to describe concretely a distributive law of the monad $\mathcal{P}_\omega(\mathsf{Id}^*)$ over $B \times \mathsf{Id}$ defining the sum $+$ and sequential composition $\cdot$ of context-free grammars (and it is impossible to use B rather than $B \times \mathsf{Id}$, see [Win14]). Instead, we use Example 7.1.10, which presents the monad $\mathcal{P}_\omega(\mathsf{Id}^*)$ by the operations and axioms of idempotent semirings. We proceed by defining a distributive law of the free monad Σ^* generated by the signature functor $\Sigma X = 1+1+(X\times X)+(X\times X)$ (to be interpreted as the constants $0, 1$ and the binary operators $+$, $\cdot$) over the copointed functor $(B \times \mathsf{Id}, \pi_2)$, and show that it preserves the semiring axioms. This distributive law arises from the abstract GSOS specification $\rho\colon \Sigma(B \times \mathsf{Id}) \Rightarrow B\Sigma^*$ whose components are given by:

$$\begin{aligned}
\rho^0_X &= (0, a \mapsto 0)\\
\rho^1_X &= (1, a \mapsto 0)\\
\rho^+_X((x,o,f),(y,p,g)) &= (\mathsf{max}\{o,p\}, a \mapsto f(a)+g(a))\\
\rho^{\cdot}_X((x,o,f),(y,p,g)) &= \left(\mathsf{min}\{o,p\}, a \mapsto \begin{cases} f(a)\cdot y & \text{if } p=0\\ f(a)\cdot y + g(a) & \text{if } p=1\end{cases}\right)
\end{aligned}$$

We proceed to show that the induced distributive law $\rho^\dagger$ preserves the defining equations of idempotent semirings. We only treat the case of distributivity, i.e., $u \cdot (v + w) = u \cdot v + u \cdot w$. To this end, let X be arbitrary and suppose that

$(o, d, x), (p, e, y), (q, f, z) \in BX \times X$. Notice that either $o = 0$ or $o = 1$; we treat both cases separately:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\rho^\dagger((0, d, x) \cdot ((p, e, y) + (q, f, z)))} \\
\quad = & (0, a \mapsto d(a) \cdot (y + z), x \cdot (y + z)) \\
\quad \mathsf{Rel}(B)(\equiv_X) & (0, a \mapsto d(a) \cdot y + d(a) \cdot z, x \cdot y + x \cdot z) \\
\quad = & \rho^\dagger((0, d, x) \cdot (p, e, y) + (0, d, x) \cdot (q, f, z))
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\rho^\dagger((1, d, x) \cdot ((p, e, y) + (q, f, z)))} \\
\quad = & (p + q, a \mapsto d(a) \cdot (y + z) + (e(a) + f(a)), x \cdot (y + z)) \\
\quad \mathsf{Rel}(B)(\equiv_X) & (p + q, a \mapsto (d(a) \cdot y + d(a) \cdot z) + (e(a) + f(a)), x \cdot y + x \cdot z) \\
\quad \mathsf{Rel}(B)(\equiv_X) & (p + q, a \mapsto (d(a) \cdot y + e(a)) + (d(a) \cdot z + f(a)), x \cdot y + x \cdot z) \\
\quad = & \rho^\dagger((1, d, x) \cdot (p, e, y) + (1, d, x) \cdot (q, f, z))\,.
\end{array}
$$

In a similar way, one can show that $\rho^\dagger$ preserves the other idempotent semiring equations. Thus, from Proposition 7.2.10 and Corollary 7.2.6 we obtain a distributive law κ of $\mathcal{P}_\omega(\mathsf{Id}^*)$ over $B \times \mathsf{Id}$ such that $i \circ q \colon \rho^\dagger \Rightarrow \kappa$ is a morphism of distributive laws, i.e., κ is presented by $\rho^\dagger$ (which is in turn determined by ρ) and the equations of idempotent semirings.

7.2.3 Distributive laws over comonads

A further type of distributive law, which generalizes all of the above, is that of a distributive law of a monad over a comonad. These arise from GSOS laws as well as from *coGSOS* laws, which allow to model operational rules which involve lookahead in the premises. We refer to [Kli11] for technical details and an example of a coGSOS format on streams. In this subsection, we prove for future reference that when constructing the quotient distributive law as above for a distributive law over a comonad, the axioms are preserved, i.e., the quotient is again a distributive law over the comonad.

Proposition 7.2.12. *Theorem 7.2.4 and Corollary 7.2.6 hold as well for any distributive law of a monad over a comonad.*

Proof. Let (D, ϵ, δ) be a comonad and $\lambda \colon TD \Rightarrow DT$ a distributive law of the monad (T, η, μ) over the comonad (D, ϵ, δ). Suppose λ preserves equations $\mathcal{E}$. By Proposition 7.2.10 there is a distributive law $\lambda^{\mathcal{E}}$ of $\mathcal{T}^{\mathcal{E}}$ over the copointed functor (D, ϵ). To show that $\lambda^{\mathcal{E}}$ is a distributive law over the comonad (D, ϵ, δ), we need to check that the corresponding axiom holds.

$$
\begin{tikzcd}
TD \arrow[rr, "\lambda"] \arrow[d, "T\delta"] \arrow[ddd, bend right=60, "q_D"'] & & DT \arrow[d, "\delta_T"] \arrow[ddd, bend left=60, "Dq"] \\
TDD \arrow[r, "\lambda_D"] \arrow[d, "q_{DD}"] & DTD \arrow[r, "D\lambda"] \arrow[d, "Dq_D"] & DDT \arrow[d, "DDq"] \\
T^{\mathcal{E}}DD \arrow[r, "\lambda^{\mathcal{E}}_D"] & DT^{\mathcal{E}}D \arrow[r, "D\lambda^{\mathcal{E}}"] & DDT^{\mathcal{E}} \\
T^{\mathcal{E}}D \arrow[rr, "\lambda^{\mathcal{E}}"] \arrow[u, "T^{\mathcal{E}}\delta"] & & DT^{\mathcal{E}} \arrow[u, "\delta_{T^{\mathcal{E}}}"']
\end{tikzcd}
$$

The outermost part and the right-hand square both commute by the fact that q is a morphism of distributive laws. The outer crescents commute by naturality of q and δ. The upper rectangle commutes by the assumption that λ is a distributive law over the comonad. Checking that the lower rectangle commutes, which is what we need to prove, is now an easy diagram chase, using that q_D is epic (Theorem 7.1.5). $\square$

7.3 Quotients of bialgebras

We show how initial and final λ-bialgebras for a distributive law relate to initial and final bialgebras for a quotiented distributive law as constructed in the previous section. We study this in the general setting of morphisms of distributive laws, and to this end we assume:

- monads $\mathcal{T} = (T, \eta, \mu)$ and $\mathcal{K} = (K, \theta, \nu)$;
- distributive laws $\lambda \colon TB \Rightarrow BT$ and $\kappa \colon KB \Rightarrow BK$ (both of monad over functor);
- a morphism of distributive laws $\tau \colon T \Rightarrow K$ from λ to κ.

Morphisms of distributive laws are defined to be monad morphisms, and hence respect the algebraic structure. The next proposition shows that, as one might expect, they also respect the coalgebraic structure, and hence morphisms of distributive laws induce morphisms between bialgebras.

Proposition 7.3.1. *Let $\widehat{T} \colon TB\text{-coalg} \to B\text{-coalg}$ and $\widehat{K} \colon KT\text{-coalg} \to K\text{-coalg}$ be liftings induced by λ and κ as in Equation* (3.14) *of Section 3.5. For all $\delta \colon X \to BTX$, τ_X is a B-coalgebra morphism from $\widehat{T}(X, \delta)$ to $\widehat{K}(X, B\tau_X \circ \delta)$.*

Proof. The following diagram commutes:

$$\begin{tikzcd}
TX \arrow[r, "\tau_X"] \arrow[d, "T\delta"'] \arrow[dr, phantom, "(\text{nat. } \tau)"] & KX \arrow[d, "K\delta"] & \\
TBTX \arrow[r, "\tau_{BTX}"] \arrow[d, "\lambda_{TX}"'] \arrow[dr, phantom, "(\text{morph. of distr. laws})"] & KBTX \arrow[r, "KB\tau_X"] \arrow[d, "\kappa_{TX}"] \arrow[dr, phantom, "(\text{nat. } \kappa)"] & KBKX \arrow[d, "\kappa_{KX}"] \\
BTTX \arrow[r, "B\tau_{TX}"] \arrow[d, "B\mu_X"'] & BKTX \arrow[r, "BK\tau_X"] \arrow[d, phantom, "(\tau \text{ monad morphism})"] & BKKX \arrow[d, "B\nu_X"] \\
BTX \arrow[rr, "B\tau_X"'] & & BKX
\end{tikzcd}$$

Commutativity of the outside is the desired result. □

If τ arises from a set of preserved equations $\mathcal{E}$ as in Section 7.2 (with $\kappa = \lambda^{\mathcal{E}}$), then Proposition 7.3.1 states that, for any coalgebra $\delta\colon X \to BTX$, the coalgebra $\widehat{K}(X, B\tau_X \circ \delta)$ is a quotient of the coalgebra $\widehat{T}(X, \delta)$, and in particular, the congruence $\equiv_X$ is included in behavioural equivalence on $\widehat{T}(X, \delta)$.

Example 7.3.2. Recall from Example 7.2.11 that the abstract GSOS specification for context-free grammars induces a morphism $i \circ q\colon \Sigma^* \Rightarrow \mathcal{P}_\omega(X^*)$ of distributive laws, where Σ^* is the free monad for the signature $\Sigma X = X \times X + X \times X + 1 + 1$ representing a binary choice $+$, a binary composition $\cdot$, and constants 0 and 1. These distributive laws induce liftings $\widehat{\Sigma^*}$ and $\widehat{\mathcal{P}_\omega(\mathsf{Id}^*)}$.

By Proposition 7.3.1 we have the following commutative diagram for any coalgebra of the form $\delta\colon X \to 2 \times (\Sigma^* X)^A$:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "\delta"'] & \Sigma^* X \arrow[r, "(i\circ q)_X"] \arrow[d, "\widehat{\Sigma^*}(\delta)"] & \mathcal{P}_\omega(X^*) \arrow[r] \arrow[d, "\widehat{\mathcal{P}_\omega(\mathsf{Id}^*)}(Bi_X \circ Bq_X \circ \delta)"] & \mathcal{P}(A^*) \arrow[d, "\zeta"] \\
& 2 \times (\Sigma^* X)^A \arrow[r, "\mathsf{id} \times ((i\circ q)_X)^A"'] & 2 \times \mathcal{P}_\omega(X^*)^A \arrow[r] & 2 \times \mathcal{P}(A^*)^A
\end{tikzcd} \tag{7.10}$$

where ζ is the final coalgebra for $BX = 2 \times X^A$.

This gives the expected correspondence between two of the three different coalgebraic approaches to context-free languages introduced in [WBR13] (the third approach is about fixed-point expressions and is outside the scope of this chapter). These two approaches are:

1. A context-free grammar is defined as a coalgebra $X \to 2 \times (\mathcal{P}_\omega(X^*))^A$ and inductively extended to a coalgebra $\mathcal{P}_\omega(X^*) \to 2 \times (\mathcal{P}_\omega(X^*))^A$, and the language semantics arises by finality. This extension coincides with our lifting $\widehat{\mathcal{P}_\omega(\mathsf{Id}^*)}$.

2. A context-free grammar is defined more syntactically (viewed as a system of behavioural differential equations in [WBR13]) as a coalgebra $X \to 2 \times (\Sigma^* X)^A$, which is inductively extended to a coalgebra $\Sigma^* X \to 2 \times (\Sigma^* X)^A$ to obtain its language semantics. This extension coincides with our lifting $\widehat{\Sigma^*}$.

The situation in diagram (7.10) yields the correspondence between these two approaches.

Similarly, if B has a final coalgebra (Z, ζ), then the algebra on ζ induced by λ (Lemma 3.5.1) factors through the algebra on ζ induced by κ.

Proposition 7.3.3. *Let $\alpha\colon TZ \to Z$ and $\alpha'\colon KZ \to Z$ be the algebras induced by λ and κ respectively on the final B-coalgebra (Z, ζ). Then $\alpha = \alpha' \circ \tau_Z$.*

Proof. Consider the following diagram:

$$\begin{tikzcd}
TZ \arrow[r, "\tau_Z"] \arrow[d, "T\zeta"'] & KZ \arrow[r, "\alpha'"] \arrow[d, "K\zeta"] & Z \arrow[dd, "\zeta"] & TZ \arrow[l, "\alpha"'] \arrow[d, "T\zeta"] \\
TBZ \arrow[r, "\tau_{BZ}"] \arrow[d, "\lambda_Z"'] & KBZ \arrow[d, "\kappa_Z"] & & TBZ \arrow[d, "\lambda_Z"] \\
BTZ \arrow[r, "B\tau_Z"'] & BKZ \arrow[r, "B\kappa"'] & BZ & BTZ \arrow[l, "B\alpha"]
\end{tikzcd}$$

The upper left square commutes by naturality of τ, whereas the lower left square commutes since τ is a morphism of distributive laws. The two rectangles commute by definition of α and α'. Thus $\alpha' \circ \tau_Z$ and α are both coalgebra homomorphisms from $(TZ, \lambda_Z \circ T\zeta)$ to (Z, ζ) and consequently $\alpha' \circ \tau_Z = \alpha$ by finality. □

Example 7.3.4. Continuing Example 7.3.2, it follows from Proposition 7.3.3 that the algebra $\alpha\colon \Sigma^*(\mathcal{P}(A^*)) \to \mathcal{P}(A^*)$ induced by the distributive law for the free monad for Σ can be decomposed as $i \circ q \circ \alpha'$, where α' is the algebra on $\mathcal{P}(A^*)$ induced by the distributive law for $\mathcal{P}_\omega(\mathsf{Id}^*)$. It can be shown by induction that α is the algebra on languages given by union and concatenation product.

Now $\alpha'\colon \mathcal{P}_\omega(\mathcal{P}(A^*)^*) \to \mathcal{P}(A^*)$ can be given by selecting a representative term and applying α, and it follows that

$$\alpha'(\mathcal{L}) = \bigcup_{L_1 \cdots L_n \in \mathcal{L}} \{w_1 \cdots w_n \mid w_i \in L_i\}\,.$$

We thus retrieved this algebra α' induced by the distributive law for $\mathcal{P}_\omega(\mathsf{Id}^*)$ from the algebra $\alpha\colon \Sigma^*(\mathcal{P}(A^*)) \to \mathcal{P}(A^*)$ on terms.

7.4 Discussion and related work

We presented a preservation condition that is sufficient for the existence of a distributive law $\lambda^{\mathcal{E}}$ for a monad with equations, given a distributive law λ for the underlying monad. This condition consists of checking that the equations are preserved by λ. We demonstrated the method by constructing distributive laws for stream calculus over commutative semirings, and for context-free grammars which use the monad of idempotent semirings. The reader is invited to compare the complexity of checking that λ preserves the equations with describing and verifying the distributive law requirements directly.

Morphisms of distributive laws are used in [Wat02] as a general approach for studying translations between operational semantics. In the current chapter, we investigated in detail the case of quotients of distributive laws. Distributive laws for monad quotients and equations are also studied in [LPW04, MM07]. The setting and motivation of [MM07] is different as they study distributive laws of one monad over another with the aim to compose these monads. We study distributive laws of a monad over a plain functor, a copointed functor or a comonad. The approach in [LPW04] (in particular Theorem 31) differs from ours in that the desired distributive law is contingent on two given distributive laws and the existence of the coequalizer (in the category of monads) which encodes equations. We have given a more direct analysis which includes a concrete proof principle.

We have focused on adding equations which already hold in the final bialgebra, whereas in Chapter 6 we introduced an approach for adding equations to a distributive law via structural congruence. The results of these chapters can possibly be combined to give a more general account of equations and structural congruences for different monads.

In the case of GSOS on labelled transition systems, proving equations to hold at the level of a specification was considered in [ACI12], based on the notion of *rule-matching bisimulation*, a refinement of De Simone's *FH-bisimulation*. Rule-matching bisimulations are based on the syntactic notion of *ruloids*, while our technique is based on preservation of equations at the level of distributive laws. It is currently not clear what the precise relation between these two approaches is; one difference is that preserving equations naturally incorporates reasoning up to congruence. Further, we do not know how, and to what extent, the decidability result of [ACI12], which is based on identifying a finite set of ruloids, is reflected at the more abstract level of the current chapter.

Bibliography

[AC14] Davide Ancona and Andrea Corradi. Sound and complete subtyping between coinductive types for object-oriented languages. In Richard Jones, editor, *ECOOP 2014 - Object-Oriented Programming - 28th European Conference, Proceedings*, volume 8586 of *Lecture Notes in Computer Science*, pages 282–307. Springer, 2014. (Cited on page 9.)

[ACI12] Luca Aceto, Matteo Cimini, and Anna Ingólfsdóttir. Proving the validity of equations in GSOS languages using rule-matching bisimilarity. *Mathematical Structures in Computer Science*, 22(2):291–331, 2012. (Cited on page 160.)

[Acz88] Peter Aczel. *Non-well-founded Sets*. Center for the Study of Language and Information Publication Lecture Notes. Cambridge University Press, 1988. (Cited on page 11.)

[AFV01] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier Science, 2001. (Cited on pages 9 and 14.)

[AGJJ12] Robert Atkey, Neil Ghani, Bart Jacobs, and Patricia Johann. Fibrational induction meets effects. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FoSSaCS 2012, proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2012. (Cited on page 52.)

[AKV00] Jiří Adámek, Václav Koubek, and Jiří Velebil. A duality between infinitary varieties and algebraic theories. *Commentationes Mathematicae Universitatis Carolinae*, 41(3):529–542, 2000. (Cited on pages 86 and 143.)

[AM89] Peter Aczel and Nax Mendler. A final coalgebra theorem. In *Category Theory and Computer Science, 1989, proceedings*, volume 389 of *LNCS*, pages 357–365. Springer, 1989. (Cited on pages 11, 46, 79, and 84.)

[APTS13] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: programming infinite structures by observations. In Giacobazzi and Cousot [GC13], pages 27–38. (Cited on page 9.)

[Awo10] Steve Awodey. *Category Theory*. Oxford Logic Guides. OUP Oxford, 2010. (Cited on page 41.)

[Bar03] Falk Bartels. Generalised coinduction. *Mathematical Structures in Computer Science*, 13(2):321–348, 2003. (Cited on page 93.)

[Bar04] Falk Bartels. *On generalised coinduction and probabilistic specification formats*. PhD thesis, CWI, Amsterdam, April 2004. (Cited on pages 14, 15, 16, 41, 60, 62, 64, 65, 82, 117, and 121.)

[BBB+12] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Information and Computation*, 211:77–105, 2012. (Cited on pages 44, 46, 48, 63, 68, 84, and 87.)

[BH98] Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticae*, 33(4):309–338, 1998. (Cited on page 9.)

[BHKR13] Marcello Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot. Presenting distributive laws. In Reiko Heckel and Stefan Milius, editors, *CALCO*, volume 8089 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2013. (Cited on page 18.)

[BHKR15] Marcello Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot. Presenting distributive laws. *Logical Methods in Computer Science*, 11(3), 2015. (Cited on page 18.)

[BIM95] Bard Bloom, Sorin Istrail, and Albert Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995. (Cited on pages 14 and 64.)

[BK01] Peter Buchholz and Peter Kemper. Quantifying the dynamic behavior of process algebras. In Luca de Alfaro and Stephen Gilmore, editors, *PAPM-PROBMIV*, volume 2165 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2001. (Cited on pages 87 and 132.)

[BM02] Maria Grazia Buscemi and Ugo Montanari. A first order coalgebraic model of pi-calculus early observational equivalence. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR 2002 - Concurrency Theory, 13th International Conference, proceedings*, volume 2421 of *Lecture Notes in Computer Science*, pages 449–465. Springer, 2002. (Cited on page 139.)

[BP12] Thomas Braibant and Damien Pous. Deciding kleene algebras in coq. *Logical Methods in Computer Science*, 8(1), 2012. (Cited on page 38.)

[BP13] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Giacobazzi and Cousot [GC13], pages 457–468. (Cited on pages 9, 13, 38, 70, and 73.)

[BP15] Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Communications of the ACM*, 58(2):87–95, 2015. (Cited on page 13.)

[BPPR14] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In Thomas Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, proceedings*, page 20. ACM, 2014. (Cited on pages 17, 18, and 117.)

[BPT15] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Witnessing (co)datatypes. In Jan Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 359–382. Springer, 2015. (Cited on page 9.)

[Brz64] Janusz Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964. (Cited on pages 21, 22, and 30.)

[BW05] Michael Barr and Charles Wells. Toposes, theories, and triples. Reprints in Theory and Applications of Categories, No. 12, 2005. Available at `http://www.tac.mta.ca/tac/reprints/articles/12/tr12abs.html`. (Cited on pages 41, 85, and 143.)

[CHM02] Andrea Corradini, Reiko Heckel, and Ugo Montanari. Compositional SOS and beyond: a coalgebraic view of open systems. *Theoretical Computer Science*, 280(1-2):163–192, 2002. (Cited on page 139.)

[Con71] John Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971. (Cited on pages 21 and 22.)

[CS11] Thierry Coquand and Vincent Siles. A decision procedure for regular expression equivalence in type theory. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs - First International Conference, CPP 2011, proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2011. (Cited on pages 19 and 38.)

[DK09] Manfred Droste and Werner Kuich. Semirings and formal power series. In *Handbook of Weighted Automata*, pages 3–28. Springer, 2009. (Cited on page 122.)

[EHB13] Jörg Endrullis, Dimitri Hendriks, and Martin Bodin. Circular coinduction in Coq using bisimulation-up-to techniques. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2013. (Cited on pages 13 and 38.)

[FS10] Marcelo Fiore and Sam Staton. Positive structural operational semantics and monotone distributive laws. In *Coalgebraic Methods in Computer Science - 10th International Workshop, CMCS 2012, Short Contributions*, page 8, 2010. (Cited on pages 116 and 123.)

[FS12] Simon Foster and Georg Struth. Automated analysis of regular algebra. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 271–285. Springer, 2012. (Cited on page 25.)

[GC13] Roberto Giacobazzi and Radhia Cousot, editors. *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, proceedings*. ACM, 2013. (Cited on pages 161, 162, and 165.)

[GJF13] Neil Ghani, Patricia Johann, and Clément Fumex. Indexed induction and coinduction, fibrationally. *Logical Methods in Computer Science*, 9(3), 2013. (Cited on page 52.)

[GR62] Seymour Ginsburg and H. Gordon Rice. Two families of languages related to ALGOL. *Journal of the ACM*, 9(3):350–371, 1962. (Cited on page 26.)

[Gra05] Clemens Grabmayer. Using proofs by coinduction to find "traditional" proofs. In José Fiadeiro, Neil Harman, Markus Roggenbach, and Jan Rutten, editors, *Algebra and Coalgebra in Computer Science: First International Conference, CALCO 2005, proceedings*, volume 3629 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2005. (Cited on page 38.)

[GS00] H. Peter Gumm and Tobias Schröder. Coalgebraic structure from weak limit preserving functors. *Electronic Notes Theoretical Computer Science*, 33:111–131, 2000. (Cited on page 80.)

[GS01] H. Peter Gumm and Tobias Schröder. Monoid-labeled transition systems. *Electronic Notes Theoretical Computer Science*, 44(1):185–204, 2001. (Cited on pages 48, 68, and 84.)

[HCKJ13] Ichiro Hasuo, Kenta Cho, Toshiki Kataoka, and Bart Jacobs. Coinductive predicates and final sequences in a fibration. *Electronic Notes Theoretical Computer Science*, 298:197–214, 2013. (Cited on pages 12, 41, 51, 52, and 56.)

[HJ97] Ulrich Hensel and Bart Jacobs. Proof principles for datatypes with iterated recursion. In Eugenio Moggi and Giuseppe Rosolini, editors, *Category Theory and Computer Science, 7th International Conference, CTCS 1997, Proceedings*, volume 1290 of *Lecture Notes in Computer Science*, pages 220–241. Springer, 1997. (Cited on page 9.)

[HJ98] Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998. (Cited on pages 12, 41, 50, 52, 56, and 104.)

[HJ04] Jesse Hughes and Bart Jacobs. Simulations in coalgebra. *Theoretical Computer Science*, 327(1-2):71–108, 2004. (Cited on pages 92, 113, 114, and 115.)

[HK71] John Hopcroft and Richard Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University, December 1971. (Cited on page 70.)

[HK11] Helle Hvid Hansen and Bartek Klin. Pointwise extensions of gsos-defined operations. *Mathematical Structures in Computer Science*, 21(2):321–361, 2011. (Cited on page 141.)

[HKP09] Helle Hvid Hansen, Clemens Kupke, and Eric Pacuit. Neighbourhood structures: Bisimilarity and basic model theory. *Logical Methods in Computer Science*, 5(2), 2009. (Cited on page 68.)

[HKR14] Helle Hvid Hansen, Clemens Kupke, and Jan Rutten. Stream differential equations: Specification formats and solution methods. Technical Report No. FM-1404, CWI, 2014. (Cited on pages 11, 14, 26, 27, 32, 38, 45, 47, and 65.)

[HMSW11] Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011. (Cited on page 36.)

[HN11] Fritz Henglein and Lasse Nielsen. Regular expression containment: coinductive axiomatization and computational interpretation. In Thomas Ball and Mooly Sagiv, editors, *38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, proceedings*, pages 385–398. ACM, 2011. (Cited on page 38.)

[HNDV13] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In Giacobazzi and Cousot [GC13], pages 193–206. (Cited on page 9.)

[HR15] Thomas Henzinger and Jean-François Raskin. The equivalence problem for finite automata: technical perspective. *Communications of the ACM*, 58(2):86, 2015. (Cited on page 38.)

[HU79] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. (Cited on page 19.)

[Jac99] Bart Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999. (Cited on pages 52, 53, 54, 98, and 99.)

[Jac06a] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, volume 4060 of *LNCS*, pages 375–404. Springer, 2006. (Cited on page 9.)

[Jac06b] Bart Jacobs. Distributive laws for the coinductive solution of recursive equations. *Information and Computation*, 204(4):561–587, 2006. (Cited on pages 14, 15, 62, 139, and 141.)

[Jac12] Bart Jacobs. Introduction to coalgebra. Towards mathematics of states and observations, 2012. Draft. (Cited on pages 12, 41, 50, 51, 103, and 149.)

[JNRS11] Bart Jacobs, Milad Niqui, Jan Rutten, and Alexandra Silva. Preface. *Theoretical Computer Science*, 412(38):4967–4968, 2011. (Cited on page 12.)

[Joh75] Peter Johnstone. Adjoint lifting theorems for categories of algebras. *Bulletin of the London Mathematical Society*, 7:294–297, 1975. (Cited on page 62.)

[JR12] Bart Jacobs and Jan Rutten. An introduction to (co)algebras and (co)induction. In *Advanced Topics in Bisimulation and Coinduction* [SR12], pages 38–99. (Cited on pages 11, 41, and 43.)

[JSS12] Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. In Dirk Pattinson and Lutz Schröder, editors, *Coalgebraic Methods in Computer Science - 11th International Workshop, CMCS 2012, Revised Selected Papers*, volume 7399 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2012. (Cited on pages 14, 46, 60, 62, and 63.)

[Kel80] Max Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22:1–84, 1980. (Cited on page 145.)

[KKV12] Clemens Kupke, Alexander Kurz, and Yde Venema. Completeness for the coalgebraic cover modality. *Logical Methods in Computer Science*, 8(3), 2012. (Cited on page 51.)

[Kli04] Bartek Klin. Adding recursive constructs to bialgebraic semantics. *Journal of Logic and Algebraic Programming*, 60-61:259–286, 2004. (Cited on pages 16 and 139.)

[Kli07] Bartek Klin. Bialgebraic methods in structural operational semantics: Invited talk. *Electronic Notes Theoretical Computer Science*, 175(1):33–43, 2007. (Cited on pages 15 and 121.)

[Kli09] Bartek Klin. Structural operational semantics for weighted transition systems. In Jens Palsberg, editor, *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, volume 5700 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2009. (Cited on pages 14, 48, 68, and 84.)

[Kli11] Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412(38):5043–5069, 2011. (Cited on pages 14, 41, 60, 61, 63, 64, 65, 132, 139, and 156.)

[KN12] Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, 49(1):95–106, 2012. (Cited on pages 19 and 38.)

[KN14] Bartek Klin and Beata Nachyla. Distributive laws and decidable properties of SOS specifications. In Johannes Borgström and Silvia Crafa, editors, *Combined 21st International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics, EXPRESS/SOS 2014, proceedings*, volume 160 of *EPTCS*, pages 79–93, 2014. (Cited on page 139.)

[KNR11] Clemens Kupke, Milad Niqui, and Jan Rutten. Stream differential equations: concrete formats for coinductive definitions. Technical Report No. RR-11-10, Oxford University, 2011. (Cited on pages 27 and 32.)

[Koz90] Dexter Kozen. On Kleene algebras and closed semirings. In Branislav Rovan, editor, *MFCS*, volume 452 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 1990. (Cited on page 26.)

[KS14] Dexter Kozen and Alexandra Silva. Practical coinduction. *To appear in Mathematical Structures in Computer Science*, 2014. (Cited on page 9.)

[Lan98] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1998. (Cited on pages 41 and 98.)

[Len98] Marina Lenisa. *Themes in Final Semantics*. PhD thesis, Università di Pisa-Udine, 1998. (Cited on page 41.)

[Len99] Marina Lenisa. From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. *Electronic Notes Theoretical Computer Science*, 19:2–22, 1999. (Cited on pages 15, 16, and 117.)

[LGCR09] Dorel Lucanu, Eugen-Ioan Goriac, Georgiana Caltais, and Grigore Rosu. CIRC: A behavioral verification tool based on circular coinduction. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, proceedings*, volume 5728 of *Lecture Notes in Computer Science*, pages 433–442. Springer, 2009. (Cited on pages 9, 19, and 38.)

[LLYL14] Lingyun Luo, Xinxin Liu, Xiaohua Yang, and Zhiming Liu. Up-to technique for product functorâŃĘ. *Journal of Computational Information Systems*, 10(22):9597–9607, 2014. (Cited on page 94.)

[LPW00] Marina Lenisa, John Power, and Hiroshi Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electronic Notes Theoretical Computer Science*, 33:230–260, 2000. (Cited on pages 15, 93, and 117.)

[LPW04] Marina Lenisa, John Power, and Hiroshi Watanabe. Category theory for operational semantics. *Theoretical Computer Science*, 327(1-2):135–154, 2004. (Cited on pages 16, 64, 139, and 160.)

[Luo06] Lingyun Luo. An effective coalgebraic bisimulation proof method. *Electronic Notes Theoretical Computer Science*, 164(1):105–119, 2006. (Cited on pages 16 and 117.)

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. (Cited on pages 10 and 47.)

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983. (Cited on pages 13, 15, 39, 67, and 71.)

[Mil89] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. (Cited on pages 9 and 15.)

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992. (Cited on page 120.)

[Mis15] Michael Mislove. Semantics column. *SIGLOG News*, 2(2), 2015. (Cited on page 12.)

[MM07] Ernie Manes and Philip Mulry. Monad compositions I: General constructions and recursive distributive laws. *Theory and Applications of Categories*, 18(7):172–208, 2007. (Cited on pages 16, 152, and 160.)

[MMS13] Stefan Milius, Lawrence S. Moss, and Daniel Schwencke. Abstract gsos rules and a modular treatment of recursive definitions. *Logical Methods in Computer Science*, 9(3:28):52 pp., 2013. (Cited on pages 14 and 139.)

[MPdS12] Nelma Moreira, David Pereira, and Simão de Sousa. Deciding regular expressions (in-)equivalence in Coq. In Wolfram Kahl and Timothy Griffin, editors, *Relational and Algebraic Methods in Computer Science - 13th International Conference, RAMiCS 2012, proceedings*, volume 7560 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2012. (Cited on page 38.)

[MR05] Mohammad Reza Mousavi and Michel Reniers. Congruence for structural congruences. In Sassone [Sas05], pages 47–62. (Cited on pages 15, 16, 120, 133, 134, 135, and 138.)

[NR09] Milad Niqui and Jan Rutten. Coinductive predicates as final coalgebras. In *6th Workshop on Fixed Points in Computer Science, FICS 2009, proceedings*, pages 79–85, 2009. (Cited on page 51.)

[NR11] Milad Niqui and Jan Rutten. A proof of moessner's theorem by coinduction. *Higher-Order and Symbolic Computation*, 24(3):191–206, 2011. (Cited on pages 10 and 13.)

[NT14] Tobias Nipkow and Dmitriy Traytel. Unified decision procedures for regular expression equivalence. *Archive of Formal Proofs*, 2014, 2014. (Cited on page 38.)

[Okh13] Alexander Okhotin. Conjunctive and boolean grammars: The true general case of the context-free grammars. *Computer Science Review*, 9:27–59, 2013. (Cited on page 30.)

[Par81] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer, 1981. (Cited on pages 9, 10, 20, and 47.)

[Plo01] Gordon Plotkin. Bialgebraic semantics and recursion (extended abstract). *Electronic Notes Theoretical Computer Science*, 44(1):285–288, 2001. (Cited on pages 16 and 139.)

[Pou07] Damien Pous. Complete lattices and up-to techniques. In Zhong Shao, editor, *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, proceedings*, volume 4807 of *Lecture Notes in Computer Science*, pages 351–366. Springer, 2007. (Cited on pages 13, 15, 67, and 75.)

[PS12] Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*

[SR12], pages 233–289. (Cited on pages 13, 15, 27, 67, 68, 72, 75, 81, 82, 83, 119, 120, and 139.)

[PW02] John Power and Hiroshi Watanabe. Combining a monad and a comonad. *Theoretical Computer Science*, 280(1-2):137–162, 2002. (Cited on page 148.)

[RB14] Jurriaan Rot and Marcello Bonsangue. Combining bialgebraic semantics and equations. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FoSSaCS 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2014. (Cited on pages 17, 18, and 122.)

[RB15] Jurriaan Rot and Marcello Bonsangue. Structural congruence for bialgebraic semantics. *Submitted*, 2015. (Cited on pages 17 and 83.)

[RBB$^+$15] Jurriaan Rot, Filippo Bonchi, Marcello Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced coalgebraic bisimulation. *To appear in Mathematical Structures in Computer Science*, 2015. (Cited on pages 17, 18, and 80.)

[RBR13a] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Coalgebraic bisimulation-up-to. In Peter van Emde Boas, Frans Groen, Giuseppe Italiano, Jerzy Nawrocki, and Harald Sack, editors, *39th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2013, proceedings*, volume 7741 of *Lecture Notes in Computer Science*, pages 369–381. Springer, 2013. (Cited on pages 17 and 18.)

[RBR13b] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Coinductive proof techniques for language equivalence. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, proceedings*, volume 7810 of *Lecture Notes in Computer Science*, pages 480–492. Springer, 2013. (Cited on pages 16 and 18.)

[RBR15] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten. Proving language inclusion and equivalence by coinduction. *To appear in Information and Computation*, 2015. (Cited on pages 16 and 18.)

[RT93] Jan Rutten and Daniele Turi. Initial algebra and final coalgebra semantics for concurrency. In Jaco de Bakker, Willem de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 530–582. Springer, 1993. (Cited on page 14.)

[Rut98a] Jan Rutten. Automata and coinduction (an exercise in coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR 1998: Concurrency Theory, 9th International Conference, proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998. (Cited on pages 9, 13, 19, 21, 22, 26, 37, and 39.)

[Rut98b] Jan Rutten. Relators and metric bisimulations. *Electronic Notes Theoretical Computer Science*, 11:252–258, 1998. (Cited on page 50.)

[Rut00] Jan Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. (Cited on pages 11, 41, 43, 46, 47, 78, 79, 80, 84, and 138.)

[Rut03] Jan Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1-3):1–53, 2003. (Cited on pages 9, 10, 11, 19, 26, 28, 45, 47, and 75.)

[Rut05] Jan Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005. (Cited on page 13.)

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, October 1998. (Cited on pages 13, 15, 67, and 117.)

[San12a] Davide Sangiorgi. *An introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012. (Cited on pages 9, 12, 48, 49, 127, and 132.)

[San12b] Davide Sangiorgi. Origins of bisimulation and coinduction. In *Advanced Topics in Bisimulation and Coinduction* [SR12], pages 1–37. (Cited on page 20.)

[Sas05] Vladimiro Sassone, editor. *Foundations of Software Science and Computational Structures, 8th International Conference, FoSSaCS 2005, proceedings*, volume 3441 of *Lecture Notes in Computer Science*. Springer, 2005. (Cited on pages 169 and 172.)

[SBBR10] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Generalizing the powerset construction, coalgebraically. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, Proceedings*, volume 8 of *LIPIcs*, pages 272–283. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. (Cited on pages 13 and 14.)

[SBBR13] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Generalizing determinization from automata to coalgebras. *Logical*

Methods in Computer Science, 9(1), 2013. (Cited on pages 15, 46, 62, and 141.)

[Sch05] Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. In Sassone [Sas05], pages 440–454. (Cited on page 98.)

[Sil15] Alexandra Silva. A short introduction to the coalgebraic method. *SIGLOG News*, 2(2), 2015. (Cited on page 12.)

[SR12] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012. (Cited on pages 9, 166, 170, and 171.)

[Sta11] Sam Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1), 2011. (Cited on page 50.)

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001. (Cited on page 119.)

[TP97] Daniele Turi and Gordon Plotkin. Towards a mathematical operational semantics. In *12th Annual IEEE Symposium on Logic in Computer Science, 1997, proceedings*, pages 280–291. IEEE Computer Society, 1997. (Cited on pages 9, 14, 26, 41, 60, 62, 63, 65, and 82.)

[Trn80] Vera Trnková. General theory of relational automata. *Fundamenta Informaticae*, 3(2):189–234, 1980. (Cited on page 51.)

[Tur96] Daniele Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996. (Cited on page 41.)

[Wat02] Hiroshi Watanabe. Well-behaved translations between structural operational semantics. *Electronic Notes Theoretical Computer Science*, 65(1):337–357, 2002. (Cited on pages 16, 148, and 160.)

[WBR13] Joost Winter, Marcello Bonsangue, and Jan Rutten. Coalgebraic characterizations of context-free languages. *Logical Methods in Computer Science*, 9(3), 2013. (Cited on pages 15, 16, 142, 155, 158, and 159.)

[Win14] Joost Winter. *Coalgebraic Characterizations of Automata-theoretic Classes*. PhD thesis, Radboud Universiteit Nijmegen, 2014. (Cited on page 155.)

[Win15] Joost Winter. A completeness result for finite λ-bisimulations. In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2015. (Cited on pages 13 and 73.)

[ZLL+10] Xiaocong Zhou, Yongji Li, Wenjun Li, Hai-yan Qiao, and Zhongmei Shu. Bisimulation proof methods in a path-based specification language for polynomial coalgebras. In Kazunori Ueda, editor, *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, proceedings*, volume 6461 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2010. (Cited on pages 16 and 117.)

Index

Curriculum vitae

- Born in Amsterdam **1987**
- High school **1999 – 2005**
  Atheneum College Hageveld, Heemstede
- BSc Computer Science **2007 – 2010**
  Leiden University
- MSc Computer Science (cum laude) **2010 – 2011**
  Leiden University
- PhD student **2012 – 2015**
  Leiden University
- Post-doctoral researcher **2015 – ...**
  Laboratoire de l’Informatique du Parallélisme (LIP), ENS Lyon

Acknowledgements

In the last years I had the pleasure of learning and researching in an inspiring, exciting and very friendly environment. I am grateful to many people for being a part of this; in particular Henning Basold, Stijn de Gouw, Jan van Rijn and Jonathan Vis. I want to thank Marcello Bonsangue, Frank de Boer and Jan Rutten for getting (and keeping) me excited about research and for their kindness. This thesis has been improved based on detailed comments on previous versions, by Bart Jacobs, Helle Hansen and Henning Basold. In the last years I have worked together with many people, which, for me, has always been one of the best parts of doing research. I am convinced this is the case because I have had the chance of working with excellent people. I want to single out Bartek Klin, who hosted me during an inspiring research period in Warsaw.

I am grateful to my family—in the first place for their support and care. To Job, for sharing his brilliant thoughts and listening carefully. Finally I thank Hanna, for supporting me in every possible way, and reminding me of what is valuable in life.

Samenvatting

Coinductie, de duale van inductie, is een fundamenteel principe voor het definiëren van oneindige objecten, en het bewijzen van eigenschappen van zulke objecten. Het belangrijkste voorbeeld van coinductie in de informatica is *bisimulatie*, een algemene karakterisatie van equivalentie tussen systemen met oneindig of circulair gedrag, met een concrete bewijsmethode. Coinductieve technieken verschaffen nuttige bewijsprincipes voor verschillende onderzoeksgebieden zoals de theorie van concurrency, de studie van oneindige datastructuren en de automatentheorie.

De brede toepasbaarheid en toenemende interesse in coinductieve technieken zijn gebaseerd op de theorie van *coalgebra's*. Dit is een wiskunde theorie waarin we eigenschappen van toestandsgebaseerde modellen van berekening kunnen begrijpen en bewijzen op een hoog abstractieniveau, en deze eigenschappen vervolgens toepassen op concrete systemen. De theorie van coalgebra's geeft een structureel en algemeen perspectief op bisimulatie en coinductie, met een canonieke karakterisatie van equivalentie en bijbehorende bewijsprincipes.

In dit proefschrift ontwikkelen we technieken die coinductief redeneren vereenvoudigen en verbeteren. We gebruiken hiervoor de theorie van coalgebra's, om algemeen toepasbare methoden te verkrijgen. In het eerste deel van het proefschrift introduceren we verbeteringen van coinductieve bewijsprincipes, en in het tweede gedeelte van coinductieve definitieprincipes.

We introduceren een coalgebraische theorie van verbeterde bewijstechnieken voor bisimilariteit, in Hoofdstuk 4. Onze theorie generaliseert de zogeheten *up-to-technieken*, die geintroduceerd zijn door Milner en Sangorgi om het rederen over processen te vereenvoudigen, van processen naar een breed scala aan toestandsgebaseerde systemen, zoals (niet)deterministische automaten, systemen die oneindige rijtjes representeren en transitiesystemen met kwantitatieve informatie. In Hoofdstuk 2 passen we deze technieken toe om te redeneren over formele talen. In Hoofdstuk 5 worden onze bewijsprincipes verder gegeneraliseerd, op basis van een algemeen perspectief op coinductieve predicaten, zoals geïntroduceerd door Hermida en Jacobs. Met deze generalisatie verkrijgen we verbeterde bewijsprincipes voor willekeurige coinductieve predicaten, wat we toepassen om nieuwe methoden te verkijgen voor het redeneren over simulatie van transitiesystemen, taalinclusie van automaten met kwantitatieve informatie, en divergentie van processen.

Coinductieve definitietechnieken zijn geschikt voor het definiëren en bestuderen van de semantiek van talen. Turi en Plotkin hebben getoond dat men een compositionele semantiek kan verkrijgen door de interactie tussen syntax (gemodelleerd door algebra's) en observaties (gemodelleerd door coalgebra's) te specificeren door middel van een zogeheten distributieve wet. In Hoofdstuk 6 laten we zien hoe zulke distributieve wetten geïntegreerd kunnen worden met recursieve vergelijkingen, om zo het specificeren van talen te vereenvoudigen. Het belangrijkste resultaat uit dit hoofdstuk is dat de interpretatie van een specificatie, die recursieve gelijkheden van een bepaalde vorm kan bevatten, compositioneel is, en dat de bewijsprincipes uit eerdere hoofdstukken gebruikt kunnen worden.

Distributieve wetten kunnen nuttig zijn om coinductief gedefinieerde talen te bestuderen, maar ze zijn soms moeilijk te beschrijven. In Hoofdstuk 7 laten we zien hoe distributieve wetten gepresenteerd kunnen worden als quotient van andere distributieve wetten, die op hun beurt makkelijk te presenteren zijn met gebruik van bestaande technieken. We passen onze techniek toe om eenvoudig distributieve wetten af te leiden voor de semantiek van operaties op oneinidige rijtjes en contextvrije grammatica's.

Titles in the IPA Dissertation Series since 2009

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction*. Faculty of Mathematics and Natural Sciences, UL. 2015-18