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.