View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by CWI's Institutional Repository

Coalgebraic Characterizations of Automata-Theoretic Classes

Proefschrift
ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. mr. S.C.J.J. Kortmann,
volgens besluit van het college van decanen
in het openbaar te verdedigen op dinsdag 1 juli 2014
om 14.30 uur precies

door
Joost Winter
geboren op 20 maart 1980
te Amsterdam

Promotor:
Prof. dr. J.J.M.M. Rutten

Copromotor:
Dr. M.M. Bonsangue (Universiteit Leiden)

Manuscriptcommissie:
Prof. dr. H. Geuvers
Prof. dr. J.W. Klop (Vrije Universiteit Amsterdam)
Prof. dr. J.P. Allouche (Institut de mathématiques de Jussieu)

Copyright © 2014 Joost Winter

Printed and bound by Ipskamp, Enschede

IPA Dissertation Series 2014-07

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

The research culminating in this dissertation was conducted as a part of the NWO project CoRE (Coinductive Calculi for Regular Expressions).

in memoriam

Dik T. Winter

1945—2009

Acknowledgements

First of all, I would like to thank Krzysztof Apt, without whom this dissertation would not have been possible, as he was the one who encouraged me to apply for this position at the CWI, for which I was eventually selected.

I would like to thank Jan and Marcello, my promotor and co-promotor, for encouragement, and for their significant role in my learning of the basics (and beyond) of coalgebras, coinduction, and category theory. Next, I would like to thank Milad Niqui, who has been an additional supervisor for this project in the first half year.

Thanks are due to all my co-authors—besides the already named Jan en Marcello—who have made possible a number of papers and short contributions: Helle Hansen, Clemens Kupke, Baltasar Trancón y Widemann, and Jurriaan Rot. Additionally, Alexandra Silva and Henning Basold also have helped a lot over the course of various discussions and the sharing, and explaining, of ideas.

For additional discussion, comments, and/or the sharing of their insights and knowledge, I would furthermore like to thank a number of people. This includes Christophe Reutenauer, Arto Salomaa, and Jeffrey Shallit, all of whom quickly replied to a question I had about the notions of algebraicity and constructive algebraicity, pointing me to the paper by Michel Fliess. Furthermore I would like to thank Jan van Eijck, for sharing his knowledge of doing stream calculus in Haskell, and Jean-Paul Allouche for his knowledge about the k-automatic and k-regular sequences. I would also like to thank Jonas Teuwen and Amar van Leeuwaarde for helping me out with some of the algebra, and the intricacies of Haskell, respectively.

Others who have been helpful to me, in some way or another, by sharing their insights, include Pierre Lescanne, Wieb Bosma, Stefan Milius, Filippo Bonchi, Jacques Sakarovitch, and Bram Westerbaan. More generally, I would like to thank the organizers and participants of the various conferences (CALCO in Winchester and Warsaw, and CMCS in Tallinn), seminars (the two Streams seminars in Leiden and the workshop on coalgebraic logics in Dagstuhl), and the summer school in Marktoberdorf: at all these places combined, I certainly learned a lot and additionally had some great times.

I would like to thank my colleagues in the Formal Methods group (or, before its renaming, the SEN3 group): Michiel Helvensteijn (for the numerous games of go and the hiking holidays in Austria), Farhad Arbab (amongst other things for introducing me to the music of Kaykhosro Pournazeri and Triskilian), Frank de Boer, Immo Grabe (for the movie nights), Lacramioara Astefanoaei (for introducing me to the movies of Béla Tarr), Stijn de Gouw, Sung-Shik Jongmans, Stephanie Kemper, Natallia Kokash, Mahdi Jaghoori, Behnaz Changizi (for inviting me to go to the Deep Purple concert in Zwolle), Dominic Luecke, Matteo Mio, Erik de Vink, Francesco Santini, David Costa, Georgiana Caltais, Christian Krause, Kasper Dokter, Nikolaos Bezirgiannis, Sean Halle, and Vlad Serbanescu. Additionally, thanks are due to Susanne van Dam and Maarten Dijkema for support in technical and bureaucratic matters, to Atze van der Ploeg, to Michaël Smeding, to Florian Speelman for joining the go playing, and to Bart de Keijzer for some evenings filled with good beer!

The IPA spring days have helped me to broaden my knowledge and understanding beyond my own research topic, and beyond that, have also been major source of fun times! For some of us, the fun times of IPA were so much fun that it led to the organizing of IPAZoP (*IPA zonder praatjes*) events! I would like to thank the organizers of the IPA spring days, Tim Willemse, Meivan Cheng, and all the participants I got to know there, including but not limited to Cynthia Kop, Zé Pedro Magelhães, Mark Timmer-van der Stam, Marijn Schraagen, and Frank Takes.

I owe a lot of thanks to Annetje, Sara, David, and all other relatives for their support throughout the period, as well as to all of the friends whom helped or supported me in whichever way during the last four years!

And finally:

This dissertation is dedicated to my father, Dik Winter (1945—2009), who worked at the CWI for 40 years as a scientific programmer and system administrator, and who gave to me his love for mathematics and computer science.

Joost Winter
Amsterdam, May 2014

Contents

1

Introduction

This dissertation presents a *coalgebraic* treatment of a number of automata-theoretic classes, in the setting of formal power series in noncommuting variables over some semiring (or, sometimes, a commutative semiring), of which both formal languages and streams are instances.

Automata theory is the study of systems involving some kind of notion of a *state space*, and transitions on this state space labelled by a collection of input symbols or an *alphabet*, and has been classically linked to the study of formal languages. A general introduction to the theory of automata and formal languages can be found in e.g. [HMU06]. One particular concern of automata theory is the classes of formal languages that can be characterized by certain types of automata (or other formal systems, such as grammars). A well-known characterization of these classes is given by the Chomsky hierarchy [Cho59] consisting of 1) the regular languages, 2) the context-free languages, 3) the context-sensitive languages, and 4) the recursively enumerable languages, with each class being contained in the next class.

Automata theory, as the study of formal languages, has been generalized in the French and Finnish schools to a study of formal power series, over arbitrary semirings. The notion of a weighted automaton (with weights in a semiring) plays an important role in this work, pioneered by Schützenberger and others in e.g. [Sch61a], [Sch61b], and [CS63]. General and comprehensive introductions to the theory of weighted automata can be found in, for example, [BR11], [Eil76], [SS78], [DKV09] and [Sak09].

During the last two decades, a *coalgebraic* view on automata and state-based systems thus emerged. Within theoretical computer science, coalgebra studies state-based transition systems on a high level of abstraction, and, closely related to it, *coinduction* which can be seen as a definition scheme *dual* to induction. Coalgebra and coinduction have so far found applications in, for example, the framework of Structural Operational Semantics [TP97], functional programming, formalized mathematics, and dynamical systems.

This introductory chapter starts with a brief survey of the surrounding background and context. The next section motivates the research and summarizes the results found in this dissertation. Section 1.3 presents an outline of the various chapters and appendices, and Section 1.4 gives an overview of the various publications on which the larger part of this dissertation is based.

1.1 Coalgebra and automata: background

In [Brz64], and later in [Con71], the notion of a *derivative* was introduced for regular expressions, closely parallelling classical calculus, with the notable exception of the derivative of the product of two regular expressions s and t, which is given by the rule:

$$(st)_a = s_a t + o(s) t_a$$

which we call the *Brzozowski product rule*. Here $o(s)$ can be seen as an *output value* of s, giving the regular expression 1 if s matches with the empty word, and the regular expression 0 otherwise. Taking the derivative to an alphabet symbol a here corresponds to making an a-transition in an automaton consisting of all regular expressions, where a is taken from an *alphabet* A, generally considered to be finite.

In [Rut98], a presentation of finite automata and regular expressions as *coalgebras* is presented, making use of Brzozowski's product rule. In the language of category theory, a coalgebra for an endofunctor F (on some category) consists of an object X in this category, together with a morphism $\delta : X \to FX$, and can be seen as the *dual* to an algebra for this endofunctor F. This abstract point of view has been developed further in the framework of *universal coalgebra* in [Rut00]. Various classes of state based systems other than finite automata, such as Moore and Mealy machines, and labelled transition systems, can be modelled as coalgebras.

Both finite automata and regular expressions are regarded as coalgebras for the functor $\mathbb{B} \times -^A$ (where $\mathbb{B}$ is the Boolean semiring identified with its carrier $\{0, 1\}$). Here, the calculus of derivatives can be seen as coinductively *defining* a coalgebra, or automaton, stucture on the set of all regular expressions. Additionally, in [Rut98], a coalgebraic presentation of Kleene's theorem (stating that a formal language is accepted by a finite automaton if and only if it matches with some regular expression) is given.

It is also possible to use this calculus of derivatives to reason about arbitrary languages, which together again form an (infinite) automaton: given a formal language $L \in \mathcal{P}(A^*)$ and an alphabet symbol $a \in A$, its a-derivative L_a can be given by

$$L_a = \{w \mid aw \in L\}.$$

This automaton is a *final* object in the category of $\mathbb{B} \times -^A$-coalgebras, i.e. for every $\mathbb{B} \times -^A$-coalgebra, there is a unique morphism function to $\mathcal{P}(A^*)$ with the above automaton (or coalgebra) structure.

The coalgebraic approach to automata has been extended to *weighted automata* in for example [Rut08] and [BBB$^+$12]. In a weighted automaton, every transition is labelled by a weight (over an assumed underlying semiring), and weighted automata can be determinized to *linear automata*[1]. The coalgebra structure of the linear automaton can be constructed as the unique linear mapping extending the coalgebra structure of the weighted automaton. This construction can be seen both as a generalization of the classical powerset construction extending each nondeterministic automaton into a deterministic one, and as an instance of a more general categorical construction presented in [SBBR10].

In the setting of weighted automata, the role of *formal languages* is taken over by *formal power series* in noncommuting variables, which can be regarded as mappings σ assigning to each word $w \in A^*$ a value $[\sigma \Downarrow w]$. Likewise, language concatenation now is replaced by the *convolution product*, which can be defined either as

$$[\sigma\tau \Downarrow w] = \sum_{uv=w} [\sigma \Downarrow u][\tau \Downarrow v]$$

or, equivalently, by Brzozowski's product rule together with the equation $1_a = 0$.

The class of power series that can be characterized using a finite weighted automaton is called *recognizable*, and corresponds to the class of *rational* power series, i.e. power series characterizable using a *rational expression* (generalizing regular expressions for languages).

In [Rut03a], (Brzozowski) derivatives of formal languages and power series are used to specify (systems of) *behavioural differential equations*, where languages, streams, power series, families thereof, or operations on these classes are characterized by means of systematically specifying the output as well as the derivative of whatever is being defined. Roughly, behavioural differential equations can be seen as differential equations with respect to Brzozowski's product rule, and have languages or formal power series as their solutions. Brzozowski's product rule itself can also be regarded as a behavioural differential equation, itself defining the *convolution product* on power series, which instantiates to language concatenation in the case of formal languages.

[1]in [BBB$^+$12] called linear weighted automata

In [Rut03b], the coalgebraic approach to weighted automata, in the specific setting of streams, is applied to combinatorial problems, giving rise to the technique of *coinductive counting*, where combinatorial problems are presented using systems of behavioural differential equations, specifying streams corresponding to the combinatorial problem. The coinductive stream calculus, and many of these counting problems, can furthermore easily and conveniently be implementated in the functional programming language Haskell: this connection has been explored in for example [McI01] and [DvE04].

Extending the framework of universal coalgebra, the theory of λ-bialgebras and distributive laws has been developed as an abstract method of looking at transition systems (or coalgebras) with an algebraic structure, in [Bar04], [Jac06], and [Kli11]. More precisely, a *bialgebra* is a structure that is simultaneously an algebra (say, for a functor T) and a coalgebra (say, for a functor F), and a λ-bialgebra additionally provides a distributive law in the form of a natural transformation $\lambda : TF \Rightarrow FT$. This framework has been used, for example, to give a coalgebraic (or bialgebraic) interpretation to rules in the abstract GSOS format, based on the SOS format from [TP97]. This framework of λ-bialgebras also forms the basis for the *generalized powerset construction* presented in [SBBR10], which gives a categorical generalization of the classical construction of deterministic automata from nondeterministic automata.

1.2 Motivation and main contributions

1.2.1 Motivation

The overarching aim of this dissertation is to extend the coalgebraic approach to automata theory to a number of additional automata-theoretic classes (of formal languages, streams, or, more generally, formal power series), and to give coalgebraic characterizations of these classes.

An important instance of such a class consists of the context-free languages. Giving a coalgebraic presentation of the context-free languages along the lines of the material in [Rut98] and [Sil10] was one of the main initial motivations for the research presented in this dissertation. This work on the context-free languages enables us to see the first two levels of the Chomsky hierarchy (the regular languages and context-free languages) in a coalgebraic context.

A secondary aim has been a search for coalgebraic presentations (or, maybe rather: presentations aligning well with the coalgebraic approach) of important existing results in automata theory. Sometimes, these presentations allow us to better connect the (traditional) algebraic approach to automata theory with the (more recent) coalgebraic and bialgebraic approaches, and allow us to see existing (concrete) results in a more general, category-theoretical, light.

1.2.2 Main contributions

The main contributions of this dissertation are roughly the following:

1. A closer integration between, on one side, the algebraic approach to automata theory (on the level of generality of semirings, as this is the maximal level of generality on which most of the theory has been developed) pioneered by Schützenberger and others, and, on the other side, the coalgebraic approach developed in e.g. [Rut98], [Rut02], [Rut03a], [SBBR10], and [BBB$^+$12].

2. Extension of the coalgebraic approach to deterministic and weighted automata to systems that represent the context-free languages and their algebraic generalizations, the *constructively algebraic* power series (over a commutative semiring).

3. A new construction of the Greibach normal form, which relies on a generalized version of a lemma using which the Kleene-Schützenberger-Eilenberg theorem (stating the equivalence between recognizable and rational power series) can also be proven. Unlike traditional constructions of the Greibach normal form, this construction does not depend on reduction to any intermediate forms, such as the Chomsky normal form.

4. A coalgebraic characterization of the k-regular sequences, extending (a variant of) an earlier characterization of the k-automatic sequences presented in [KR12]. These characterizations are both based on an isomorphism of final coalgebras, between a final coalgebra consisting of streams on one side, and a final coalgebra consisting of formal power series on the other side.

5. Semantic characterizations of the constructively algebraic power series and k-regular sequences, which connect well to their coalgebraic characterization, and to similar, existing, characterizations of simple and recognizable power series and k-automatic sequences.

6. An investigation of various ways in which the coalgebraic approach to context-free languages and their generalizations can be connected to the world of λ-bialgebras and distributive laws.

7. A brief survey of how the various classes presented in this dissertation (simple, recognizable, constructively algebraic, k-automatic, and k-regular) can all easily be implemented in Haskell. In addition, a simple tool, QStream, is presented including helper functions for looking up streams in the Online Encyclopedia of integer Sequences.

1.2.3 Main characterizations

Throughout this dissertation, various different *automata-theoretic classes* (such as the regular languages, the context-free languages, and their generalizations to other semirings) are characterized by means of a format of behavioural differential equations, as well as a semantic characterization. As a basic example, the regular languages can be characterized by finite systems of behavioural differential equations where each derivative is specified using simply an element of some system, and a language is regular if and only if there is a finite set Σ of languages, such that for each language in it, and each alphabet symbol, the derivative with respect to this symbol is in Σ again.

We now present a summary of the various automata-theoretic classes, the corresponding formats for behavioural differential equations, and the corresponding semantic characterizations. Each of these semantic characterizations relies on the existence of a finite set Σ, for which all derivatives satisfy some kind of closure property, which is different for each of the classes.

The various types of systems of behavioural differential equations can be gathered in the following table:

class	type of system	example
simple	simple	$o(x) = 1, x' = x$
recognizable	linear	$o(x) = 1, x' = 2x$
constructively algebraic	polynomial	$o(x) = 1, x' = x^2$
k-automatic	simple **zip**	$o(x) = 1, x' = \mathbf{zip}(x, y)$
		$o(y) = 0, y' = \mathbf{zip}(y, x)$
k-regular	linear **zip**	$o(x) = 1, x' = \mathbf{zip}(x, 2x)$

Similarly, the next table combines each of the semantic characterizations, highlighting the structural similarity between these results: these characterizations can be found in the dissertation as Proposition 2.6, Proposition 2.16, Proposition 3.6, Proposition 3.16, Corollary 5.4, and Corollary 5.10.

The table can be read as follows: each row of the table states that a formal power series $\sigma \in M$ satisfies property $P(\sigma)$ if and only if there is a finite $\Sigma \subseteq M$

with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$, τ satisfies property $Q(\tau)$. Or equivalently, every row of the table satisfies the following formula of first order logic:

$$\forall \sigma \in M.(P(\sigma) \iff \exists \Sigma \in \mathcal{P}_\omega(M).(\sigma \in \Sigma \wedge \forall \tau \in \Sigma.Q(\tau)))$$

with the appropriate instantiations of M, P, and Q.

M	$P(\sigma)$	$Q(\tau)$
$S\langle\langle A\rangle\rangle$	σ is simple	$\forall a \in A$, $\tau_a \in \Sigma$
$S\langle\langle A\rangle\rangle$	σ is recognizable	$\forall a \in A$, τ_a is linear in Σ
$S\langle\langle A\rangle\rangle$	σ is constructively algebraic	$\forall a \in A$, τ_a is polynomial in Σ
$S\langle\langle A\rangle\rangle$	σ is constructively algebraic	$\forall a \in A$, τ_a is rational in Σ
$S^{\mathbb{N}}$	σ is k-automatic	$\forall i < k$, $\mathbf{unzip}_{i,k}(\tau') \in \Sigma$
$S^{\mathbb{N}}$	σ is k-regular	$\forall i < k$, $\mathbf{unzip}_{i,k}(\tau')$ is linear in Σ

Of these characterizations, the ones for constructively algebraic power series and k-regular sequences can be seen as new contributions in this dissertation.

1.3 Overview of the material

Chapter 2 gives a survey of the existing coalgebraic presentations of deterministic automata (or Moore machines) and weighted automata. Deterministic, weighted, and linear automata are introduced, and existing results are presented on the level of generality of arbitrary semirings.

This chapter also gives a (perhaps to some extent new) proof of a classical result due to Kleene, Schützenberger, and Eilenberg, stating that a power series is recognizable (by a weighted automaton) if and only if it is rational (that is, definable by a rational expression). This result is a generalization of Kleene's theorem, stating that a language is definable by a finite automaton if and only if it can be described by a regular expression. One important advantage of our presentation is that, in the next chapter, we can re-use the main lemma in order to present a construction of the Greibach normal form.

Finally, this chapter presents the notion of *bisimulations up to linear combinations*, as a proof technique for behavioural equivalence between linear automata. This notion essentially corresponds to that of *linear weighted simulations* presented in [BBB+12], and provides us with a highly useful proof technique for some of the main results in the remainder in this dissertation.

Chapter 3 extends the presentation from Chapter 2 to systems of behavioural differential equations in which each derivative is given as a polynomial over the underlying set of variables (or nonterminals). Such systems again can be given an interpretation coalgebraically, using similar techniques to those used in Chapter 2, and characterize precisely the context-free languages as well as their usual generalization to commutative semirings, the constructively algebraic power series.

Furthermore, in Section 3.3, we re-use Lemma 2.22 to present a new construction of the Greibach normal form, which does not depend on intermediate normal forms such as the Chomsky normal form. Finally, in Section 3.4, we employ a result due to Chomsky and Schützenberger in order to give coinductive characterizations of streams representing combinatorial problems of counting functions of context-free grammars.

Chapter 4 extends the coalgebraic framework for automata from the two previous chapters further, by giving characterizations of pushdown automata, the Hadamard product, and finally the **zip**-operator, which will play a prominent role in Chapter 5. Two characterizations of pushdown automata are given: first, pushdown automata with empty stack acceptance, and second, pushdown automata which are in an accepting configuration when both the stack is empty, and the current state is an accepting one, which we call refined empty stack acceptance. One of the results in this chapter is a bisimulation-based proof of a classical result of Schützenberger, stating that the Hadamard product of a constructively algebraic power series and a rational power series again is constructively algebraic. Unlike the traditional proof of this result, our proof relies on a direct transformation into pushdown automata of pairs of systems of behavioural differential equations.

Chapter 5 gives a coalgebraic treatment of the k-automatic and k-regular sequences, based on the **zip** and **unzip**-operators presented in the previous chapter. This is done by presenting an alternate (but necessarily isomorphic) final S-automaton for an alphabet A_k of size k, consisting of the set of all sequences (or streams, or power series) over S, which, on reading the ith alphabet symbol, makes a transition consisting of composing the operator $\mathbf{unzip}_{i,k}$ with the **tail** operator. This isomorphism can be given by making use of the bijective base k numeration system, which yields a bijection between words over A_k and natural numbers. Both the k-automatic and k-regular sequences can be represented coalgebraically by making use of this isomorphism, and are in correspondence with simple and recognizable power series.

It is next shown that this coalgebraic characterization provides a characterization of a family of sequences definable by *divide-and-conquer recurrences* as k-regular sequences. The chapter concludes with a discussion of Christol's theorem and connects this result to a result by Fliess, establishing the equivalence between *algebraic* and *constructively algebraic* power series in a single variable (or, equivalently, streams).

Chapter 6 presents two alternative coinductive characterizations of the constructively algebraic power series. The first of these characterizations is through term algebras, in which terms over the signature of semirings take the place of polynomials. This yields a notion of syntactic systems of behavioural differential equations, similar to weighted automata and polynomial systems, and we establish that every S-automaton generated by a syntactic system is mapped by a homomorphism onto a S-automaton generated by a polynomial system: in other words, we can see polynomial systems as quotients of syntactic systems.

We next present a characterization through μ-expressions (here seen as *unique* fixed point expressions), and give a transformation back and forth from μ-expressions to term algebras. The method used can be regarded as a reworking of the (generic) correspondence between coalgebras and μ-expressions presented in [Sil10], albeit in a different setting not considered there.

Chapter 7 connects the constructions from Chapters 2 and 3 with the general categorical framework of *(λ)bialgebras* and distributive laws (see for example [Bar04], [Jac06], and [Kli11]). After presenting some relevant ideas and results of the general framework, and recalling how weighted automata can directly be obtained from this framework, we turn to a treatment of constructively algebraic power series. Here, a direct fit is less clear and some subtleties arise, which can be resolved in various ways, for example by considering a distributive law over a co-pointed functor. Finally, we give a definition for what we call *Brzozowski bialgebras*, which are both automata and algebras over a commutative semiring, and which satisfy the Brzozowski product rule. Brzozowski bialgebras again form a category, which has a number of useful properties in common with categories of λ-bialgebras.

Chapter 8 gives an implementation of the coinductive stream calculus (which arises as an instance of the coinductive calculus of power series) in the functional programming language Haskell. Using Haskell's *lazy evaluation*, it is directly possible to give coinductive definitions, and by extending Haskell's numerical type `Num` to streams, rational and algebraic streams can be defined directly. In order to specify k-automatic and k-regular streams in Haskell, we need one more function `xzip`, which again is defined coinductively, and using which the operations $\mathbf{zip}_k$ for any k can be given. Finally, a set of *point-free* definitions of the coinductive calculus is given, bringing us again closer to the categorical perspective.

Appendices A and B recall the underlying definitions from algebra and category theory, together with some important examples and results. These appendices, to a certain extent, also serve as preliminary material for this dissertation. In Chapters 2 until 6, however, the dependence on category is very light, and does not—at least, not *explicitly*—depend on the notions of monads, natural transformations, or Eilenberg-Moore algebras, which are only considered as such in Chapter 7.

Appendix C concludes with collection of coinductively defined streams, with a few exceptions all inside one of the main classes considered in this dissertation.

1.4 Origins of the material

For the most part, the new contributions in this dissertation have their origins in the following papers:

1. [WBR11]: Joost Winter, Marcello M. Bonsangue, and Jan Rutten. *Context-free languages, coalgebraically.* (2011)

2. [BRW12]: Marcello M. Bonsangue, Jan Rutten, and Joost Winter. *Defining context-free power series coalgebraically.* (2012)

3. [WBR13]: Joost Winter, Marcello M. Bonsangue, and Jan Rutten. *Coalgebraic characterizations of context-free languages.* (2013)

4. [WBR14]: Joost Winter, Marcello M. Bonsangue, and Jan Rutten. *Context-free coalgebras.* (Accepted for publication)

5. [Win13]: Joost Winter. *QStream: A Suite of Streams* (2013)

6. [HKRW14]: Helle H. Hansen, Clemens Kupke, Jan Rutten, and Joost Winter. *A final coalgebra for k-regular sequences.* (Accepted for publication)

The papers [WBR13] and [WBR14] are journal papers extending the conference papers [WBR11] and [BRW12], and have been used as source material for this dissertation only indirectly.

The new material in Chapter 2 is mostly taken from [WBR14], with the exception of the (unpublished) presentation of rational series and the Kleene-Schützenberger-Eilenberg theorem. Chapter 3 incorporates, and integrates, results from [WBR13] as well as [WBR14]. The material in Chapter 4 is unpublished. Chapter 5 for the most part is based on [HKRW14]. Finally, the material in Chapters 6 and 7 originates from [WBR13]. Finally, Chapter 8 is based on [Win13].

2

Regular languages and rational power series

In this chapter, we introduce the basics of the coalgebraic approach to automata theory, including a variety of results which have been presented before in e.g. [Rut98], [Rut03a], [BBB$^+$12] and [SBBR13].

We start by having a look at automata with output in an arbitrary set S without assuming any additional structure on S (often called *Moore automata* in the literature) from a coalgebraic perspective. At this level, using instantiations of the general framework of universal coalgebra [Rut00], we can already talk about homomorphisms, bisimulations, and the existence of a final automaton (or, in more abstract terms, a final coalgebra).

We bring algebra into the picture in Section 2.2 by considering final automata for the case when S is the carrier of a *semiring*, and show that the final automaton can be seen as both a semiring and an S-module. These semiring and S-module structures are compatible with the interpretation of $S\langle\langle A\rangle\rangle$ as *formal power series* in noncommuting variables. Specific instances of such final automata include the set of all *formal languages* over any alphabet, and the set of all *streams* over any semiring, both of which have been studied extensively in a coalgebraic context.

Next, in Section 2.3, we introduce *S-linear automata* (for a semiring S), of which the final automata from the previous section are instances, as S-automata which additionally have the structure of a S-module. Closely related to S-linear automata are *S-weighted* automata, which can be *determinized* into S-linear automata. This determinization generalizes the classical determinization of nondeterministic automata, and, in a broader categorical context, is itself an instance of the *generalized powerset construction* presented in [SBBR13].

In Section 2.4, we introduce the notion of S-rational power series and present a classic result by Kleene, Schützenberger and Eilenberg ([Sch61a], [Eil76]) stating that a power series is S-rational if and only if it is recognized by some finite S-weighted automaton. Finally, in Section 2.5, we introduce the notion of *bisimulation up to linear combinations*, an instance of the categorical presentation in [RBR13] of *bisimulation up to*, as a useful technique to prove equality of languages and formal power series.

2.1 Automata as coalgebras

Fixing a finite alphabet A, and given an arbitrary set S, an S*-automaton* is defined as a triple

$$(Q, o, \delta)$$

where

1. Q is a set of states;
2. $o : Q \to S$ is a function assigning an *output value* from S to each state; and
3. $\delta : Q \to Q^A$ assigns a mapping from alphabet symbols to states to each state. We call δ the *transition function* or the *derivative* of the automaton.

We call an S-automaton (Q, o, δ) *finite* whenever Q is finite, and often simply refer to such an automaton as Q, when no confusion about the meaning of the output and transition funcitons concerned is likely to arise.

We will usually write q_a or in some cases, when we want to be more explicit (for example, in the frequent case when we are comparing different automata), the curried notation $\delta(q, a)$ instead of $\delta(q)(a)$, and call q_a the a*-derivative* of q.

Contrary to more traditional presentations of automata, but following most coalgebraic work on automata theory, our automata do not any have designated initial states. However, it will soon become clear that nothing is lost in doing so, as we can always assume any state from the automaton as 'initial' if so desired. At the same time, by not regarding initial states as an intrinsic part of the automaton, we gain the advantage (as we will see soon) of the existence of unique automaton morphisms from arbitrary automata into a designated automaton satisfying the property of being *final*.

Example 2.1. Consider the following automaton over the alphabet $\{a, b\}$ with outputs in $\{0, 1\}$, represented pictorially by

$$\begin{tikzcd}
x \downarrow 1 \arrow[loop above, "a"] \arrow[r, bend left, "b"] & y \downarrow 0 \arrow[loop above, "a"] \arrow[l, bend left, "b"]
\end{tikzcd}$$

or equivalently by the following *system of behavioural differential equations*:

$$\begin{array}{lll} o(x) = 1 & x_a = x & x_b = y \\ o(y) = 0 & y_a = y & y_b = x \end{array}$$

Note that both representations contain exactly the same information: we will henceforth identify any automaton with the corresponding system of behavioural differential equations.

We can extend the transition function δ to a transition function δ^* over words $w \in A^*$ inductively by setting

$$\delta^*(q, 1) = q \qquad \text{and} \qquad \delta^*(q, aw) = \delta^*(\delta(q, a), w). \tag{2.1}$$

(Throughout this dissertation, the empty word is denoted by 1.)

Since $\delta^*(q, a) = \delta^*(\delta(q, a), 1) = \delta(q, a)$ for all $q \in Q$ and $a \in A$, the functions δ and δ^* are compatible, and we can rewrite (2.1) using the earlier shorthand notation as

$$q_1 = q \qquad \text{and} \qquad q_{aw} = (q_a)_w$$

which hold for all $q \in Q$, $a \in A$, and $w \in A^*$. The second of these equations can be generalized using a simple lemma:

Lemma 2.2. *Given a S-automaton (Q, o, δ) and any $q \in Q$, the equation $q_{vw} = (q_v)_w$ holds for all $v, w \in A^*$.*

Proof. Induction on the length of v. If $|v| = 0$, then $v = 1$ and note $q_{1w} = q_w = (q_1)_w$. If $|v| > 0$ then $v = au$ for some $a \in A$ and $u \in A^*$, and use the inductive hypothesis that $q_{zw} = (q_z)_w$ holds for all $z \in A^*$ with $|z| < |v|$. In particular, we have $q_{uw} = (q_u)_w$. Now observe

$$q_{vw} = q_{(au)w} = q_{a(uw)} = (q_a)_{uw} = ((q_a)_u)_w = (q_{au})_w = (q_v)_w$$

and the proof is complete. □

A *morphism* between two S-automata (Q, o, δ) and (R, p, γ) is a mapping $h : Q \to R$ such that for all $q \in Q$ and $a \in A$, we have $o(q) = p(h(q))$ and $h(q_a) = h(q)_a$. Or equivalently, iff it makes the diagram

$$\begin{tikzcd} Q \arrow[r, "h"] \arrow[d, "{(o,\delta)}"'] & R \arrow[d, "{(p,\gamma)}"] \\ S \times Q^A \arrow[r, "1_S \times h^A"] & S \times R^A \end{tikzcd}$$

commute. Thus, S-automata and their morphisms form a category, with the identity function and function composition as identity morphism and composition of morphisms. This category is, by the above diagram, identical to the category of coalgebras for the **Set**-endofunctor $S \times -^A$.

Lemma 2.3. *Given S-automata (Q, o, δ) and (R, p, δ), for any morphism $h : Q \to R$, any $q \in Q$ and any $w \in A^*$, $h(q_w) = h(q)_w$.*

Proof. Induction on the length of w. Base case: $h(p_1) = h(p) = h(p)_1$. Inductive case: assume $h(p_v) = h(p)_v$ for all $p \in Q$, then also $h(p_{av}) = h((p_a)_v) = h(p_a)_v = h(p)_{av}$. □

The category of S-automata has a final object, which can be given by $(S\langle\langle A\rangle\rangle, O, \Delta)$, where $S\langle\langle A\rangle\rangle$ is defined as the function space

$$\{f \mid f : A^* \to S\}$$

from A^* to S. We will subsequently use the notation

$$[\sigma \Downarrow w]$$

for the evaluation of $\sigma \in S\langle\langle A\rangle\rangle$ at the word w. We will by convention let elements from $S\langle\langle A\rangle\rangle$ be denoted by the Greek letters σ, τ, etc.

For now, S can be any set, but once we start adding more structure to our automata, we will assume S to be a semiring and also consider the elements of $S\langle\langle A\rangle\rangle$ as *formal power series*. In the absence of any algebraic structure on $S\langle\langle A\rangle\rangle$, elements of $S\langle\langle A\rangle\rangle$ can maybe best be thought of as *S-language partitions*, that is, partitions of A^* into S.

We define the automaton structure (O, Δ) on $S\langle\langle A\rangle\rangle$, for any $\sigma \in S\langle\langle A\rangle\rangle$, $a \in A$ and $w \in A^*$, by

$$O(\sigma) = \sigma(1) \qquad \text{and} \qquad [\sigma_a \Downarrow w] = [\sigma \Downarrow aw]$$

and it's easy to see that this, indeed, unambiguously specifies such an automaton structure.

The equality $[\sigma_a \Downarrow w] = [\sigma \Downarrow aw]$ can again be generalized inductively:

Lemma 2.4. *For all $v, w \in A^*$, we have $[\sigma_v \Downarrow w] = [\sigma \Downarrow vw]$.*

Proof. First show by induction on the length of v that for all $\sigma \in S\langle\langle A\rangle\rangle$ and $w \in A^*$,

$$[\sigma \Downarrow v] = [\sigma_v \Downarrow 1]$$

and then use Lemma 2.2 to conclude $[\sigma_v \Downarrow w] = [\sigma \Downarrow vw]$ for arbitrary w. □

The following proposition establishes the fact that $S\langle\langle A\rangle\rangle$ is a final automaton or, in the more general terminology of universal coalgebra, a final coalgebra for the functor $S \times -^A$:

Proposition 2.5. *For any S-automaton (Q, o, δ) over an alphabet A, there exists a unique morphism from (Q, o, δ) to the automaton $(S\langle\langle A\rangle\rangle, O, \Delta)$.*

Proof. Consider the mapping $[\![-]\!] : Q \to S\langle\langle A\rangle\rangle$ defined by

$$[[\![q]\!] \Downarrow w] = o(q_w).$$

To see that this mapping is a morphism, we have to show that $o(q) = O([\![q]\!])$ and that $[\![q_a]\!] = [\![q]\!]_a$ for all $q \in Q$ and $a \in A$. But this is the case, since for any $w \in A^*$, we have

$$O([\![q]\!]) = [[\![q]\!] \Downarrow 1] = o(q_1) = o(q)$$

and

$$[[\![q_a]\!] \Downarrow w] = o((q_a)_w) = o(q_{aw}) = [[\![q]\!] \Downarrow aw] = [[\![q]\!]_a \Downarrow w].$$

For unicity, assume that h is a morphism from Q to $S\langle\langle A\rangle\rangle$. But this gives

$$[h(q) \Downarrow w] = o(h(q)_w) = o(h(q_w)) = o(q_w) = [[\![q]\!] \Downarrow w]$$

for arbitrary $q \in Q$ and $w \in A^*$, so we must have $h = [\![-]\!]$. □

The statement of this proposition can be summarized in the following commuting diagram

$$\begin{tikzcd}
Q \arrow[r, dotted, "{[\![-]\!]}"] \arrow[d, "{(o,\delta)}"'] & S\langle\langle A\rangle\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times Q^A \arrow[r, dotted, "{1_S \times [\![-]\!]^A}"] & S \times S\langle\langle A\rangle\rangle^A
\end{tikzcd}$$

with the dotted arrows indicating that $[\![-]\!]$ is the *only* function making this diagram commute.

We call any $\sigma \in S\langle\langle A\rangle\rangle$ *S-simple* if and only if there is a finite S-automaton (Q, o, δ) and a $q \in Q$ such that $[\![q]\!] = \sigma$. Equivalently, simple power series can be characterized as follows:

Proposition 2.6. *Given any set S and any $\sigma \in S\langle\langle A\rangle\rangle$, σ is S-simple if and only if there is a finite set $\Sigma \subseteq S\langle\langle A\rangle\rangle$ with $\sigma \in \Sigma$, such that for all $\tau \in \Sigma$ and all $a \in A$, $\tau_a \in \Sigma$.*

Proof. If σ is S-simple, there is a finite automaton (Q, o, δ) with a q s.t. $[\![q]\!] = \sigma$. Now notice $\{[\![q]\!] \mid q \in Q\}$ is a subset of $S\langle\langle A\rangle\rangle$ with the stated property. Conversely, if Σ has the stated property, then (Σ, O, Δ) is a finite automaton with, for all $\sigma \in \Sigma$, $[\![\sigma]\!] = \sigma$, so σ is S-simple. □

We now turn to bisimulations, which can be seen as a relational generalization of morphisms. The notion of *bisimulation* can, in general, be seen as a technique to establish behavioural equivalence between different systems—in this case S-automata. Given S-automata (P, o_P, δ_P) and (Q, o_Q, δ_Q), we say a relation $R \subseteq P \times Q$ is a *bisimulation between P and Q* if and only if, whenever $(p, q) \in R$, we have

1. $o_P(p) = o_Q(q)$, and

2. for all $a \in A$, $(p_a, q_a) \in R$.

This notion of bisimulation can be seen as an instance of a more general concept of bisimulation between F-coalgebras, and the results that now follow are instances of more general results, which can be found in [Rut00].

Proposition 2.7. *If $R \subseteq P \times Q$ is a bisimulation between (P, o_P, δ_P) and (Q, o_Q, δ_Q), and $(p, q) \in R$, then $[\![p]\!] = [\![q]\!]$.*

Proof. We can define an automaton structure on R by defining, for $(p, q) \in R$, $o_R((p, q)) = o_P(p)(= o_Q(q))$, and $(p, q)_a = (p_a, q_a)$. Now observe that the projection functions $\pi_1 : R \to P$ and $\pi_2 : R \to Q$ are morphisms as $o_P(\pi_1(p, q)) = o_P(p) = o_R((p, q))$, and $\pi_1((p, q)_a) = \pi_1(p_a, q_a) = p_a = (\pi_1(p, q))_a$, and similarly for π_2.

We now obtain morphisms $[\![-]\!] \circ \pi_1$ and $[\![-]\!] \circ \pi_2$ from R into the final coalgebra, but by Proposition 2.5, it follows that these morphisms must be identical, and hence, given $(p, q) \in R$,

$$[\![p]\!] = [\![\pi_1(p, q)]\!] = [\![\pi_2(p, q)]\!] = [\![q]\!]$$

completing the proof. □

We thus obtain the following categorical characterization: a relation R is a bisimulation between S-automata (P, o_P, δ_P) and (Q, o_Q, δ_Q) if and only if the diagram

$$\begin{tikzcd}
P \arrow[d, "{(o_P, \delta_P)}"] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "{(o_R, \delta_R)}"] & Q \arrow[d, "{(o_Q, \delta_Q)}"] \\
S \times P^A & S \times R^A \arrow[l, "1_S \times {\pi_1}^A"'] \arrow[r, "1_S \times {\pi_2}^A"] & S \times Q^A
\end{tikzcd}$$

commutes, with o_R and δ_R as in the above proposition.

Furthermore, given S-automata (P, o_P, δ_P) and (Q, o_Q, δ_Q), we say elements $p \in P$ and $q \in Q$ are *bisimilar* and write $p \sim_{P,Q} q$ if and only if there is some bisimulation R between (P, o_P, δ_P) and (Q, o_Q, δ_Q) such that $(p, q) \in R$.

Lemma 2.8. *Given S-automata (P, o_P, δ_P) and $Q, o_Q, \delta_Q)$, as well as $p \in P$ and $q \in Q$, we have $p \sim_{P,Q} q$ if and only if $[\![p]\!] = [\![q]\!]$.*

Proof. If $p \sim_{P,Q} q$, there is some bisimulation R between P and Q, such that $(p, q) \in R$. It follows from Proposition 2.7 that $[\![p]\!] = [\![q]\!]$. Conversely, given $p \in P$ and $q \in Q$ such that $[\![p]\!] = [\![q]\!]$, consider the relation

$$R = \{(r, s) \mid r \in P, s \in Q, [\![r]\!] = [\![s]\!]\}.$$

If $[\![r]\!] = [\![s]\!]$, then $o(r) = O([\![r]\!]) = O([\![s]\!]) = o(s)$ and for any $a \in A$, $[\![r_a]\!] = [\![r]\!]_a = [\![s]\!]_a [\![s_a]\!]$, so R is a bisimulation and, as $(p, q) \in R$, we get $p \sim_{P,Q} q$. □

We will only need this technique in Chapter 6, but this is a good point to introduce the notion of *bisimulation up to bisimilarity*, an instance of the general framework of *coalgebraic bisimulation up to* (see e.g. [Pou13], [RBR13]).

Given S-automata (P, o_P, δ_P) and (Q, o_Q, δ_Q), a relation $R \subseteq P \times Q$ is called a *bisimulation up to bisimilarity* if and only if, whenever $(p, q) \in R$, we have

1. $o_P(p) = o_Q(q)$, and
2. for all $a \in A$, there are elements $s \in Q$, $t \in R$, such that $s \sim p_a$, $t \sim q_a$, and $(p_a, q_a) \in R$.

Proposition 2.9. *If R is a bisimulation up to bisimilarity between (P, o_P, δ_P) and (Q, o_Q, δ_Q), and $(p, q) \in R$, then $[\![p]\!] = [\![q]\!]$.*

Proof. Extend R to a relation

$$S := \{(p,q) \mid \exists r, s : p \sim r, r\ R\ s, s \sim q\}.$$

We now verify that S is a bisimulation. If $(p,q) \in S$, then by $p \sim r$, $r\ R\ s, s \sim q$ we get $o_P(p) = o_P(r) = o_Q(s) = o_Q(q)$. Given any $a \in A$, moreover, because R is a bisimulation up to bisimilarity and $r\ R\ s$, there are $t \in P$, $u \in Q$, such that $t \sim r_a$, $u \sim s_a$, and $t\ R\ u$. From $p \sim r$ and $s \sim q$ we now get $p_a \sim r_a$ and $s_a \sim q_a$ and transitivity now gives us $p_a \sim t$ and $q_a \sim u$. As $t\ R\ u$, it now follows that $p_a\ S\ q_a$. Hence, S is a bisimulation and $[\![p]\!] = [\![q]\!]$. $\square$

2.2 Languages, power series, and streams

At this point, we add algebra into the purely colgebraic presentation of the previous section, by considering the case where the output set S has the structure of a semiring. In this case, we have canonical semiring and semimodule structures on the set $S\langle\langle A\rangle\rangle$; these structures will be presented in this section. Most of the results in this section can be found in [BR11], [Rut03a], and [Rut05].

Whenever this is the case, we can say a few things about the possible structure of $S\langle\langle A\rangle\rangle$. Specifically, we will see that $S\langle\langle A\rangle\rangle$ can be given

- a semiring structure;
- a left S-module structure; and
- a right S-module structure.

2.2.1 Formal languages

We first consider the case of $\mathbb{B}$, the Boolean semiring with carrier $\{0,1\}$, and with multiplication and addition representing the operations min (or $\wedge$) and max (or $\vee$). The $\mathbb{B}$-simple languages are, by definition, precisely the *regular* languages.

Via the bijection $f : \mathcal{P}(A^*) \rightarrowtail \mathbb{B}\langle\langle A\rangle\rangle$ defined by

$$[f(L) \Downarrow w] = \textbf{if } w \in L \textbf{ then } 1 \textbf{ else } 0$$

the final automaton now becomes

$$\mathcal{P}(A^*) \cong \mathbb{B}\langle\langle A\rangle\rangle$$

with the operations $O : \mathcal{P}(A^*) \to \mathbb{B}$ and $\Delta : (\mathcal{P}(A^*) \times A) \to \mathcal{P}(A^*)$ translating over this bijection as

$$\begin{array}{rcl} O(L) & = & \textbf{if } 1 \in L \textbf{ then } 1 \textbf{ else } 0 \\ L_a & = & \{w \mid aw \in L\} \end{array}$$

for all $L \in \mathcal{P}(A^*)$. Given any $\mathbb{B}$-automaton Q, we moreover call a state $q \in Q$ *accepting* whenever $o(q) = 1$.

Example 2.10. Consider the $\mathbb{B}$-automaton from Example 2.1, which was given by the system:

$$\begin{array}{lll} o(x) = 1 & x_a = x & x_b = y \\ o(y) = 0 & y_a = y & y_b = x \end{array}$$

The final homomorphism $[\![-]\!]$ maps x to the language (over the alphabet $\{a, b\}$ of all words containing an even number of bs, and maps y to the language of all words containing an odd number of bs.

We now define a multiplication operator representing language concatenation

$$LM = \{vw \mid v \in L, w \in M\}$$

and it is easily verified that

$$(\mathcal{P}(A^*), \cup, \cdot, \varnothing, \{1\})$$

is again a semiring.

We moreover define a mapping $i : \mathbb{B} \to \mathcal{P}(A^*)$ by

$$i(0) = \varnothing, \qquad i(1) = \{1\}, \qquad \text{and} \qquad j(w) = \{w\}.$$

which is directly seen to be a semiring morphism between $\mathbb{B}$ and $\mathcal{P}(A^*)$. An element from $S\langle\langle A \rangle\rangle$ that is in the image of i is called a *scalar* in S.

Moreover (as we will soon prove in a more general setting), the following *derivative rules*, involving both the semiring structure and the $\mathbb{B}$-automaton structure of $\mathcal{P}(A^*)$ hold:

$$\begin{array}{rclrcl} O(\varnothing) & = & 0 & \varnothing_a & = & \varnothing \\ O(\{1\}) & = & 1 & \{1\}_a & = & \varnothing \\ O(\{b\}) & = & 0 & \{b\}_a & = & \textbf{if } b = a \textbf{ then } \{1\} \textbf{ else } \varnothing \\ O(L \cup M) & = & O(L) \vee O(M) & (L \cup M)_a & = & \sigma_a \cup \tau_a \\ O(LM) & = & O(L) \wedge O(M) & (LM)_a & = & L_a M \cup i(O(L)) M_a \end{array}$$

This list of rules corresponds to, and generalizes, the derivative rules given by Brzozowski for regular expressions in [Brz64].

2.2.2 Formal power series

This structural presentation fully generalizes from formal languages to *formal power series in noncommuting variables*, which have been extensively studied in automata theory. Originally conceived of as an abstraction of the Taylor series of analytic functions, for automata-theoretic purposes formal power series can maybe best be seen as S-weighted languages, assinging to each word over the alphabet a weight in S.

Given any semiring S, we can define a semiring structure on $S\langle\langle A\rangle\rangle$ by giving definitions for the constants 0 and 1 and the binary operations $+$ and $\cdot$ pointwise for all $w \in A^*$:

$$\begin{array}{rcl} [0 \Downarrow w] & = & 0 \\ [1 \Downarrow w] & = & \textbf{if } w = 1 \textbf{ then } 1 \textbf{ else } 0 \\ [\sigma + \tau \Downarrow w] & = & [\sigma \Downarrow w] + [\tau \Downarrow w] \\ [\sigma\tau \Downarrow w] & = & \sum\limits_{uv=w} [\sigma \Downarrow u][\tau \Downarrow v]. \end{array}$$

It now follows that $(S\langle\langle A\rangle\rangle, +, \cdot, 0, 1)$ is again a semiring. The product that we defined on $S\langle\langle A\rangle\rangle$ is known as the *Cauchy product* or the *convolution product.*

Furthermore, we define a left scalar product $\cdot_l$ and a right scalar product $\cdot_r$ again pointwise using

$$\begin{array}{rcl} [\sigma + \tau \Downarrow w] & = & [\sigma \Downarrow w] + [\tau \Downarrow w] \\ [k \cdot_l \sigma \Downarrow w] & = & k[\sigma \Downarrow w] \\ [\sigma \cdot_r k \Downarrow w] & = & [\sigma \Downarrow w]k. \end{array}$$

for all $k \in S$, and it now follows that the structures

$$(S\langle\langle A\rangle\rangle, \cdot_l, +, 0) \qquad \text{and} \qquad (S\langle\langle A\rangle\rangle, \cdot_r, +, 0)$$

are left and right S-modules, respectively (as usual coinciding whenever S is commutative).

Again, we can define an injective function $i_S : S \to S\langle\langle A\rangle\rangle$ pointwise using the equation

$$[i_S(s) \Downarrow w] = \textbf{if } w = 1 \textbf{ then } s \textbf{ else } 0.$$

and once again i_S is a semiring morphism from S to $S\langle\langle A\rangle\rangle$. Similarly, we can define a function $j : A^* \to S\langle\langle A\rangle\rangle$ by

$$[j(v) \Downarrow w] = \textbf{if } v = w \textbf{ then } 1 \textbf{ else } 0$$

and j is a monoid morphism from A^* to $S\langle\langle A\rangle\rangle$.

In what follows, we will usually omit the notation for i_S and j, and simply write s and v for $i_S(s)$ and $j(v)$: this can be done without causing any problems because i_S and j are semiring and monoid morphisms, respectively.

Combining the semiring structure on $S\langle\langle A\rangle\rangle$ with the earlier automaton structure, it becomes clear that Brzozowski's derivative rules again are valid:

Proposition 2.11. *For all $s \in S$, $b \in A$, $\sigma, \tau \in S\langle\langle A\rangle\rangle$, we have:*

$$\begin{array}{rclrcl} O(s) & = & s & s_a & = & 0 \\ O(b) & = & 0 & b_a & = & \textbf{if } b = a \textbf{ then } 1 \textbf{ else } 0 \\ O(\sigma + \tau) & = & O(\sigma) + O(\tau) & (\sigma + \tau)_a & = & \sigma_a + \tau_a \\ O(\sigma\tau) & = & O(\sigma)O(\tau) & (\sigma\tau)_a & = & \sigma_a\tau + O(\sigma)\tau_a \end{array}$$

Proof. The only case that is not trivial is the case of the product: here, first we have

$$O(\sigma\tau) = [\sigma\tau \Downarrow 1] = [\sigma \Downarrow 1][\tau \Downarrow 1] = O(\sigma)O(\tau)$$

and then, for any $w \in A^*$,

$$\begin{array}{rcl} [(\sigma\tau)_a \Downarrow w] & = & [\sigma\tau \Downarrow aw] \\ & = & \displaystyle\sum_{tu=aw} [\sigma \Downarrow t][\tau \Downarrow u] \\ & = & \displaystyle\sum_{vz=w} [\sigma \Downarrow av][\tau \Downarrow z] + [\sigma \Downarrow 1][\tau \Downarrow aw] \\ & = & \displaystyle\sum_{vz=w} [\sigma_a \Downarrow v][\tau \Downarrow z] + [\sigma \Downarrow 1][\tau_a \Downarrow w] \\ & = & [\sigma_a\tau \Downarrow w] + O(\sigma)[\tau_a \Downarrow w] \\ & = & [\sigma_a\tau + O(\sigma)\tau_a \Downarrow w]. \end{array}$$

□

Notation aside, the crucial difference between Brzozowzki's calculus of derivatives and the familiar calculus of function derivatives lies in the *product rule*

$$(\sigma\tau)_a = \sigma_a\tau + O(\sigma)\tau_a$$

which divers from the ordinary product rule

$$(fg)_x = f_x g + f g_x$$

by an additional application of the output function O on σ.

We will now turn to an elementary but useful result, also called the *fundamental theorem* of the coinductive calculus (see e.g. [Rut03a]):

Proposition 2.12. *For any $\sigma \in S\langle\langle A\rangle\rangle$, we have:*

$$\sigma = O(\sigma) + \sum_{a\in A} a\sigma_a$$

Proof. Consider the following relation $R \subseteq S\langle\langle A\rangle\rangle \times S\langle\langle A\rangle\rangle$:

$$R = \left\{ \left(\sigma, O(\sigma) + \sum_{a\in A} a\sigma_a\right) \middle| \sigma \in S\langle\langle A\rangle\rangle \right\} \cup \{(\sigma,\sigma) \mid \sigma \in S\langle\langle A\rangle\rangle\}.$$

It is easy to see that R is a bisimulation on $S\langle\langle A\rangle\rangle$. □

This result can again be represented diagrammatically, yielding a bijection between $S\langle\langle A\rangle\rangle$ and $S \times S\langle\langle A\rangle\rangle^A$:

$$\begin{tikzcd} S\langle\langle A\rangle\rangle \arrow[r, shift left, "(O,\Delta)"] & S \times S\langle\langle A\rangle\rangle^A \arrow[l, shift left, "\lambda s.\pi_1(s) + \sum_{a\in A} a\pi_2(s)(a)"] \end{tikzcd}$$

2.2.3 Streams

Frequently, we will be concerned with the case where the alphabet A consists of a single symbol, which we will represent using the symbol $\mathcal{X}$. This case corresponds to so-called *streams* over S, which can be seen as infinite sequences over S together with operators **head** and **tail**, again also called *output* and *derivative*. Extensive studies of the coinductive stream calculus can be found in for example [Rut03a] and [Rut05].

We regard streams as infinite sequences $\sigma \in S^{\mathbb{N}}$ together with operations **head** and **tail** defined by

$$\mathbf{head}(\sigma) = \sigma(0) \qquad \text{and} \qquad [\mathbf{tail}(\sigma)](n) = \sigma(n+1).$$

Again, we call the tail of a stream its *derivative* and will commonly use the notation σ' for $\mathbf{tail}(\sigma)$. Likewise, we call the derivative $\sigma_{\mathcal{X}^n}$ the nth derivative of σ and denote it as $\sigma^{(n)}$.

The set $S\langle\langle\{\mathcal{X}\}\rangle\rangle$ then is in bijective correspondence with the set $S^{\mathbb{N}}$ of streams over S, by identifying $[\sigma \Downarrow \mathcal{X}^n]$ with $\sigma(n)$ between the two representations.

This creates an isomorphism of S-automata

$$(S\langle\langle\{X\}\rangle\rangle, O, \Delta) \quad \cong \quad (S^{\mathbb{N}}, \mathbf{head}, \mathbf{tail})$$

and now $S^{\mathbb{N}}$ directly inherits the semiring and semimodule structure of $S\langle\langle\{X\}\rangle\rangle$. For example, the definition of the product now instantiates as

$$(\sigma\tau)(n) = \sum_{i=0}^{n} \sigma(i)\tau(n-i)$$

and the corresponding product rule as

$$(\sigma\tau)' = \sigma'\tau + \mathbf{head}(\sigma)\tau'.$$

Moreover, the fundamental theorem instantiates for streams as

$$\sigma = \sigma(0) + \mathcal{X}\sigma'.$$

When dealing with streams over arbitrary sets S, we have no semiring or module structure on $S^{\mathbb{N}}$, and thus terms like $\mathcal{X}\sigma$ are not defined. For these cases, we introduce as an alternative a *cons* operator ::, defined coinductively by

$$O(s :: \sigma) = s \qquad \text{and} \qquad (s :: \sigma)' = \sigma$$

as an inverse of the output and derivative operators. Or equivalently, by the single equation $\sigma = O(\sigma) :: \sigma'$.

When S is a semiring, it now directly follows that

$$s + \mathcal{X}\sigma = s :: \sigma$$

for all $s \in S$ and $\sigma \in S^{\mathbb{N}}$ using the fundamental theorem and the equality $(\mathcal{X}\sigma)' = \sigma$.

When the underlying semiring S is a field, an stream $\sigma \in S^{\mathbb{N}}$ is invertible if and only if $\mathbf{head}(\sigma)$ is invertible, and satisfies the behavioural differential equations

$$\mathbf{head}(\sigma^{-1}) = \mathbf{head}(\sigma)^{-1} \qquad \text{and}(\sigma^{-1})' = -\mathbf{head}(\sigma)^{-1} \cdot \sigma' \cdot \sigma^{-1}.$$

We note that there is a connection between the operations on streams defined this way, and the world of *generating functions* (which play a very minor role in this dissertation), considering functions that generate certain power series as their Taylor expansion. An introduction to generating functions can be found in [Wil06]: we note that the operations of addition, multiplication and inverse on streams coincide with those of generating functions. For example, the power series expansion of the generating function

$$A(X) = \frac{X}{(1-X)(1-2X)}$$

is given by the following stream:

$$\frac{\mathcal{X}}{(1-\mathcal{X})(1-2\mathcal{X})}$$

This gives a justification for the symbol $\mathcal{X}$, which can be regarded as a 'fossilized variable', that is, a variable regarded as a constant (or a singleton alphabet symbol) in the coalgebraic approach.

We can now summarize the relationship between the general case of formal power series, and the two specific instances of formal languages and streams as follows:

	Formal languages	Formal power series	Streams
Alphabet	A	A	$\{\mathcal{X}\}$
Outputs	$\mathbb{B}$	S	S
Solution space	$\mathcal{P}(A^*) \cong \mathbb{B}\langle\langle A\rangle\rangle$	$S\langle\langle A\rangle\rangle$	$S\langle\langle\{\mathcal{X}\}\rangle\rangle \cong S^{\mathbb{N}}$
Derivative of x	x_a $(a \in A)$	x_a $(a \in A)$	x'

Many of the examples in this dissertation consist of systems of behavioural differential equations defining streams over natural numbers or integers. For these examples, we give a number of initial values of the streams, together with the number of the associated entry in the Online Encyclopedia of Integer Sequences, which can be found at:

`http://oeis.org`

Example 2.13. Consider the automaton

$$\begin{tikzcd} x \downarrow 0 \arrow[r, bend left] & y \downarrow 1 \arrow[l, bend left] \end{tikzcd}$$

over a single alphabet symbol ($\mathcal{X}$, not shown in the drawing of the automaton), and with outputs in $\mathbb{N}$. This automaton corresponds to the system of behavioural differential equations

$$\begin{array}{ll} o(x) = 0 & x' = y \\ o(y) = 1 & y' = x \end{array}$$

and it is easily seen that the final homomorphism maps x onto the stream perpetually repeating the pattern $0, 1$:

$$0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, \ldots \quad \text{(A000035)}$$

2.3 Weighted and linear automata

We will now give a coalgebraic presentation of a part of the theory of *weighted automata* over a semiring S, which constitutes a foundation for most of the material in the remainder of this dissertation. Languages and power series characterizable by finite weighted automata are called *S-recognizable.* In our presentation, weighted automata can be regarded as linear systems of behavioural differential equations. These systems, moreover, have unique solutions, which can be obtained by transforming any S-weighted automaton into a corresponding S-linear automaton.

Our presentation contrasts with (but can be linked to) the traditional treatment of weighted automata by means of transition matrices: introductions to this traditional approach can be found in for example [Eil76], [BR11], [Sak09], [DKV09]. Instead of matrices, we formulate our presentation using *formal linear combinations* together with general mappings of S-modules and the S-linear mappings between them.

The observation that rational power series occur as the solutions of (classical) systems of differential equations was first made in [FR83]. More recently, rational power series have been considered in a coalgebraic setting in [Rut08] and [BBB$^+$12]. The determinization method presented in this section can be traced back to [Bar04], and has more recently been treated extensively in a more general coalgebraic context (to which we will return in Chapter 7) in [SBBR10].

2.3.1 Linear automata

We start by considering S-automata (Q, o, δ) where the output set S is the carrier of a semiring, and the state space Q is the carrier of a S-module. Assuming a semiring $(S, +, \cdot, 0, 1)$ (which we will usually just refer to as S), we call an S-automaton (Q, o, δ) a *S-linear automaton* whenever:

1. Q is the carrier of a S-module; and
2. $o : Q \to S$ is a S-linear mapping; and
3. for each $a \in A$, the derivative to A, $(-)_a : Q \to Q$ is a S-linear mapping.

Hence, we can regard S-linear automata as S-automata with outputs in a semiring S, that are S-modules and satisfy additional linearity conditions. S-linear automata and S-linear mappings that are also S-automata morphisms again form a category.

The automaton $(S\langle\langle A\rangle\rangle, O, \Delta)$ together with its S-module structures presented in the previous section is easily seen to be a S-linear automaton. Soon, after weighted automata have been introduced, we will see that S-linear automata also occur as the *determinized* or *linearized* extensions of weighted automata.

We can now establish directly that the final mapping $[\![-]\!]$ from any linear automaton (Q, o, δ) is a S-linear mapping:

Proposition 2.14. *If (Q, o, δ) is a S-linear automaton, $[\![-]\!]$ is a S-linear mapping.*

Proof. By Proposition 2.5, we have $[\![q]\!](w) = o(q_w)$ for all $q \in Q$. Because for all w, the operation $q \mapsto \delta^*(q, w)$ is S-linear, and o itself is S-linear, we have

$$\begin{array}{rl} & [[\![k_0x_0 + k_1x_1]\!] \Downarrow w] \\ = & o((k_0x_0 + k_1x_1)_w) \\ = & k_0o((x_0)_w) + k_1o((x_1)_w) \\ = & k_0[[\![x_0]\!] \Downarrow w] + k_1[[\![x_1]\!] \Downarrow w] \\ = & [k_0[\![x_0]\!] + k_1[\![x_1]\!] \Downarrow w] \end{array}$$

for all $w \in A^*$, and hence

$$[\![k_0x_0 + k_1x_1]\!] = k_0[\![x_0]\!] + k_1[\![x_1]\!]$$

proving that $[\![-]\!]$ is S-linear. □

From this proposition, it follows directly that $S\langle\langle A\rangle\rangle$ is a final object in the category of S-linear automata and S-linear automata morphisms.

2.3.2 Formal linear combinations

Given a semiring $(S, \cdot_S, +_S, 1_S, 0_S)$, any set Y, and a function $f : Y \to S$, let the *support* of the function f

$$\mathbf{supp}(f) := \{y \in Y \mid f(y) \neq 0\}$$

denote the set of elements in Y that are mapped onto a nonzero element of S.

Furthermore, let

$$\mathrm{Lin}_S(X) := \{f : X \to S \mid \mathbf{supp}(f) \text{ is finite}\}$$

denote the set of functions from X to S with finite support, which we regard as representing formal finite S-linear combinations over X.

Given any $s \in \mathrm{Lin}_S(X)$ and any $x \in X$, we again use the notation

$$[s \Downarrow x]$$

to denote the evaluation of s (seen as a function from X to S) at x.

Again, $\mathrm{Lin}_S(X)$ can be assigned a S-module structure $(\mathrm{Lin}_S(X), +, \cdot, 0)$, *overloading* the symbols 0, $+$, and $\cdot$, and specified by:

$$\begin{array}{rcl}
[0 \Downarrow x] & = & 0 \\
[t + u \Downarrow x] & = & [t \Downarrow x] + [u \Downarrow x] \\
[s \cdot t \Downarrow x] & = & s[t \Downarrow x]
\end{array}$$

We also define an injection $\eta^1_X : X \to \mathrm{Lin}_S(X)$ by

$$[\eta^1_X(x) \Downarrow y] = (\mathbf{if}\ x = y\ \mathbf{then}\ 1\ \mathbf{else}\ 0)$$

allowing us to represent elements of $\mathrm{Lin}_S(X)$ using summation notation, as the equality

$$s = \sum_{x \in X} [s \Downarrow x]\eta^1_X(x)$$

holds for all $s \in \mathrm{Lin}_S(X)$ w.r.t. the S-module structure on $\mathrm{Lin}_S(X)$. Note that this infinite sum is always defined, because by definition there are only finitely many $x \in X$ with $s \Downarrow x \neq 0$.

We can extend any function $f : X \to Y$ to a function

$$\mathrm{Lin}_S(f) : \mathrm{Lin}_S(X) \to \mathrm{Lin}_S(Y)$$

by defining, for arbitrary $s \in \mathrm{Lin}_S(X)$

$$[\mathrm{Lin}_S(f)(s) \Downarrow y] = \sum_{x \in f^{-1}(y)} [s \Downarrow x].$$

It is moreover easy to check that $\mathrm{Lin}_S(-)$ is a functor.

Given any S-module M, any set X, and a function $f : X \to M$, there moreover is always a unique linear mapping $\hat{f} : \mathrm{Lin}_S(X) \to M$ making the diagram

$$\begin{tikzcd} X \arrow[r, hook, "\eta^1_X"] \arrow[d, "f"'] & \mathrm{Lin}_S(X) \arrow[dl, dotted, "\hat{f}"] \\ M & \end{tikzcd} \tag{2.2}$$

commute. Moreover, $\hat{f}$ can be specified by

$$\hat{f}(t) = \sum_{x \in X} [t \Downarrow x] x.$$

In the special instance when S is the Boolean semiring, $\mathrm{Lin}_S(X)$ is equivalent to $\mathcal{P}_\omega(X)$, the set of finite subsets of X; and, given a $f : X \to Y$, the function $\mathrm{Lin}_S(f)$ corresponds to $\mathcal{P}_\omega(f)$, defined by $\mathcal{P}_\omega(f)(Z) = \{f(z) \mid z \in Z\}$ for finite $Z \subseteq X$.

2.3.3 Weighted automata

Fixing a finite alphabet A, a weighted automaton with weights in a semiring S, or a *S-weighted automaton*, consists of a triple

$$(X, o, \delta)$$

where

1. X is a set, to be regarded as a set of *variables* or *nonterminals*;
2. $o : X \to S$ is, as in the case of automata, an output function
3. $\delta : X \to \mathrm{Lin}_S(X)^A$ is the transition function, describing each of the possible derivatives of the set X as an S-weighted linear combination of elements of X.

Similarly to how S-automata can be seen as coalgebras for the functor $S \times -^A$, S-weighted automata can be seen as coalgebras for the functor $S \times \mathrm{Lin}_S(-)^A$. In other words, S-weighted automata have a different type from S-automata.

However, we can transform any weighted automaton (X, o, δ) into a linear automaton $(\mathrm{Lin}_S(X), \hat{o}, \hat{\delta})$. To ensure compatibility between o and δ on one side, and $\hat{o}$ and $\hat{\delta}$ on the other side, we must have $\hat{o}(\eta^1(x)) = o(x)$ for all $x \in X$, and $\hat{\delta}(\eta^1(x))(a) = \delta(x)(a)$ for all $x \in X$ and $a \in A$. Moreover, $\hat{o}$ and $\hat{\delta}$ have to be linear mappings as well in order for $(\mathrm{Lin}_S(X), \hat{o}, \hat{\delta})$ to be a linear S-automaton. Because $\mathrm{Lin}_S(X)$ is the free S-module over X, however, we know that there must be a unique mapping satisfying these properties: given any $s \in \mathrm{Lin}_S(X)$, we have

$$\hat{o}(s) = \hat{o}\left(\sum_{x \in X} [s \Downarrow x] x\right) = \sum_{x \in X} [s \Downarrow x] o(x)$$

and for all $\sigma \in \mathrm{Lin}_S(X)$ and $a \in A$, we have

$$\sigma_a = \left(\sum_{x \in X} [s \Downarrow x] x\right)_a = \sum_{x \in X} [s \Downarrow x] x_a.$$

This construction, which can be seen as an instance of a more general categorical construction originally presented in [SBBR10], can be summarized in the following diagram, together with the final homomorphism from the linear automaton $\mathrm{Lin}_S(X)$ into the final automaton:

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^1_X"] \arrow[d, "(o,\delta)"'] & \mathrm{Lin}_S(X) \arrow[r, dotted, "\llbracket - \rrbracket"] \arrow[dl, "(\hat{o},\hat{\delta})"'] & S\langle\langle A \rangle\rangle \arrow[d, "(O,\Delta)"] \\
S \times \mathrm{Lin}_S(X)^A \arrow[rr, dotted, "1_S \times \llbracket - \rrbracket^A"] & & S \times S\langle\langle A \rangle\rangle^A
\end{tikzcd} \tag{2.3}$$

We will call a formal power series σ *S-recognizable* whenever there is a finite weighted automaton X, and an $x \in X$, such that $\llbracket \eta^1(x) \rrbracket = \sigma$.

We end this section with a result giving another characterization of the S-recognizable power series. Given a $\Sigma \subseteq S\langle\langle A \rangle\rangle$, we call a power series $\sigma \in S\langle\langle A \rangle\rangle$ *S-linear in* Σ whenever there is a finite sequence $\tau_1, \ldots, \tau_n$ of elements from Σ, and a corresponding sequence $s_1, \ldots, s_n$ of elements from S, such that

$$\sigma = \sum_{i=1}^{n} s_i \tau_i.$$

An equivalent characterization in terms of formal linear combinations can be given as follows: σ is linear in Σ if and only if there is some $t \in \mathrm{Lin}_S(\Sigma)$ with $\alpha_{S\langle\langle A \rangle\rangle}(t) = \sigma$.

Lemma 2.15. *Given any S-linear automaton (Q, p, γ) and any S-weighted automaton (X, o, δ), if a function $f : X \to Q$ makes the diagram*

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "{(o,\delta)}"'] & Q \arrow[d, "{(p,\gamma)}"] \\
S \times \mathrm{Lin}_S(X)^A \arrow[r, "1_S \times (\hat{f})^A"] & S \times Q^A
\end{tikzcd}$$

commute, then the unique linear mapping $\hat{f} : \mathrm{Lin}_S(X) \to Q$ extending f (i.e. with the property $\hat{f} \circ \eta^1_X = f$) makes the diagram

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^1_X"] \arrow[d, "{(o,\delta)}"'] & \mathrm{Lin}_S(X) \arrow[r, "\hat{f}"] \arrow[dl, "{(\hat{o},\hat{\delta})}"'] & Q \arrow[d, "{(p,\gamma)}"] \\
S \times \mathrm{Lin}_S(X)^A \arrow[rr, "1_S \times \hat{f}^A"] & & S \times Q^A
\end{tikzcd}$$

commute. Moreover, when (Q, p, γ) is a final S-linear automaton, we have $\hat{f} = [\![-]\!]$.

Proof. Because $f = \hat{f} \circ \eta^1_X$, the commutativity of the diagram obtained by removing $(\hat{o}, \hat{\delta})$ from the second diagram is immediate. Moreover, the triangle on the left commutes as a direct result of the definition of $(\hat{o}, \hat{\delta})$.

Now observe that $(1_S \times \hat{f}^A) \circ (\hat{o}, \hat{\delta})$ and $(p, \gamma) \circ \hat{f}$ are both S-linear mappings from $\mathrm{Lin}_S(X)$ extending $(p, \gamma) \circ f$, and now, because for any function $g : X \to Q$ there is a unique linear mapping $\hat{g}$ extending g, it follows that $(1_S \times \hat{f}^A) \circ (\hat{o}, \hat{\delta}) = (p, \gamma) \circ \hat{f}$, and thus the second diagram again commutes.

If (Q, p, γ), moreover, is a final S-linear automaton, $\hat{f}$ necessarily coincides with the unique linear mapping into it. □

Proposition 2.16. *Given a semiring S and any $\sigma \in S\langle\langle A\rangle\rangle$, σ is S-recognizable if and only if there is a finite Σ with $\sigma \in \Sigma$, such that for all $\tau \in \Sigma$ and $a \in A$, τ_a is linear in Σ.*

Proof. If σ is S-recognizable, then let (X, o, δ) be a weighted automaton witnessing this and note that the set

$$\Sigma := \{[\![\eta^1(x)]\!] \mid x \in X\}$$

satisfies the required property as a result of diagram (2.3).

Conversely, assume there is a finite Σ with $\sigma \in \Sigma$, such that for all $\tau \in \Sigma$ and $a \in A$, τ_a is linear in Σ. This is equivalent (letting $\iota : \Sigma \to S\langle\langle A\rangle\rangle$ denote the inclusion of Σ into $S\langle\langle A\rangle\rangle$) to the condition that there exists a mapping (o, δ) making the diagram

$$\begin{tikzcd}
\Sigma \arrow[r, "\iota"] \arrow[d, "(o,\delta)"'] & S\langle\langle A\rangle\rangle \arrow[d, "(p,\gamma)"] \\
S \times \mathrm{Lin}_S(\Sigma)^A \arrow[r, "1_S \times \hat{\iota}^A"] & S \times S\langle\langle A\rangle\rangle^A
\end{tikzcd}$$

commute and now by Lemma 2.15 it follows that $\sigma = [\![\eta^1_\Sigma(\sigma)]\!]$, establishing that σ is S-recognizable. □

We call a subset $\Sigma \subseteq S\langle\langle A\rangle\rangle$ *S-linearly stable* whenever, for each $\sigma \in \Sigma$ and each $a \in A$, σ_a is S-linear in Σ. This allows us to rephrase the preceding proposition as follows: any $\sigma \in S\langle\langle A\rangle\rangle$ is S-recognizable if and only if there is a finite S-linearly stable $\Sigma \in S\langle\langle A\rangle\rangle$ with $\sigma \in \Sigma$.

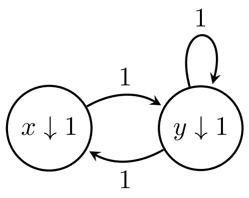
This result is essentially equivalent to [BR11, Proposition 1.5.1], which states that a formal power series $\sigma \in S\langle\langle A\rangle\rangle$ is S-recognizable if and only if there is a stable S-submodule M of $S\langle\langle A\rangle\rangle$ containing σ that is finitely generated (i.e. that includes a finite Σ such that each $\tau \in M$ is linear in Σ).

The definition of *stable* from [BR11] is exactly equivalent to the statement that there is a finite Σ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$ and each $w \in A^*$, $\Delta^*(\tau, w)$ is linear in Σ. This in turn is equivalent to the condition that for each $a \in A$, $\Delta(\tau, a)$ is linear in Σ, as a direct consequence of the fact that linear combinations of linear combinations can again be regarded as linear combinations.

Example 2.17. Consider the following system over the semiring $\mathbb{N}$, with a set $X = \{x, y\}$ of two variables, and a single alphabet symbol $\mathcal{X}$:

$$\begin{array}{rr} o(x) = 1 & x' = y \\ o(y) = 1 & y' = x + y \end{array}$$

This system corresponds to the weighted automaton[1]



x is mapped by the final homomorphism to the stream of Fibonacci numbers

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots \quad \text{(A000045)},$$

easily establishing that this sequence is a $\mathbb{N}$-recognizable stream.

2.4 Rational expressions and power series

In this section, we give a presentation of a well-known generalization of Kleene's theorem to arbitrary semirings, often known as the Kleene-Schützenberger theorem and also often called (again) *fundamental theorem.* This result was, in fact, first presented in the most general case of arbitrary semirings by Eilenberg in [Eil76].

Our proof of this result can be seen as a variation, or an alternate presentation, of more traditional proofs of this result, which can be found, for example, in [BR11]. This presentation is similar in flavour to the presentation of Kleene's theorem in [Rut98], and its proof can be regarded as a direct generalization of the proof technique used there:

[1]The 1s labelling the arrows here are the weights of the transitions, rather than alphabet symbols.

- Starting from rational series, we inductively show that for each rational power series, it is possible to construct a S-linearly stable subset of $S\langle\langle A\rangle\rangle$ containing this power series. This can be seen as a 'semantic' analogue of the explicit construction of new automata from old automata. This direction of the proof already appeared using this technique as Theorem 3 in [Rut99].
- From automata to expressions, we combine Arden's rule with induction to prove that systems of equations satisfying certain properties are guaranteed to have unique rational solutions, using a reduction from systems in $k+1$ variables to systems in k variables.

In [Sil10], a similar result was proved coalgebraically in a more general setting, establishing the equivalence between expressions and finite systems for coalgebras for a large class of functors. This class includes weighted automata over arbitrary semirings. However, the language of expressions canonically obtained there is slightly different from ours, with a unique fixed point operator μ taking over the role of the star operator.

2.4.1 The star operator and Arden's rule

We call a power series $\sigma \in S\langle\langle A\rangle\rangle$ *proper* whenever $O(\sigma) = 0$. If σ is proper, we want to define the star σ^* as the evaluation of the infinite sum $\sigma^* := \sum_{i=0}^{\infty} \sigma^i$. However, we first need to make precise what we mean when we talk about such an infinite sum. In order to do this, we take a more direct route by concretely defining a suitable notion of limit, rather than deriving this notion of limit from the discrete topology on S and the product topology on $S\langle\langle A\rangle\rangle$, as is done in e.g. [BR11].

Given a $\mathbb{N}$-indexed family $\{\sigma_i \mid i \in \mathbb{N}\}$ of formal power series, we say that the limit

$$\lim_{i\to\infty} \sigma_i$$

exists iff for all $w \in A^*$ there is a $k \in \mathbb{N}$ such that $[\sigma_j \Downarrow w] = [\sigma_k \Downarrow w]$ for all $j \geq k$. In other words: this limit exists iff for all $n \in \mathbb{N}$, the infinite sequence $[\sigma_0 \Downarrow w], [\sigma_1 \Downarrow w], \ldots$ is eventually constant. Given a particular $n \in \mathbb{N}$ and the associated k, we then write:

$$[\lim_{i\to\infty} \sigma_i \Downarrow w] = [\sigma_k \Downarrow w]$$

Now note that, for any proper power series σ, we have

$$[\sigma^n \Downarrow w] = 0$$

for all w with $|w| < n$, and thus the limit

$$\sigma^* := \lim_{n\to\infty} \sum_{i\in\mathbb{N}, i\leq n} \sigma_i$$

exists, and will be denoted using the notation σ^*.

Proposition 2.18 (Arden's rule)**.** *Given a $\sigma \in S\langle\langle A\rangle\rangle$ and a proper $\tau \in S\langle\langle A\rangle\rangle$, the unique solution to*

$$x = \sigma + \tau x \qquad \textit{is} \qquad x = \tau^*\sigma \tag{2.4}$$

and the unique solution to

$$x = \sigma + x\tau \qquad \textit{is} \qquad x = \sigma\tau^*. \tag{2.5}$$

Proof. See e.g. [BR11, Lemma 4.1]. □

Remark 2.19. When S is the Boolean semiring $\mathbb{B}$, or more generally an idempotent semiring, the star operator can be defined without restriction (as is usually done in the case of languages), and it is possible to present Arden's rule without the reference to properness. In this dissertation, we will restrict ourselves to the case of arbitrary semirings together with the properness condition.

From Arden's rule, by substituting 1 for σ in (2.4), we can directly derive the equality

$$\tau^* = 1 + \tau(\tau^*)$$

for all proper $\tau \in S\langle\langle A\rangle\rangle$. Applying the output and derivative operators on both sides of this expression, we obtain the equalities

$$o(\tau^*) = 1 \qquad \text{and} \qquad (\tau^*)_a = \tau_a(\tau^*) \tag{2.6}$$

which again hold for all proper $\tau \in S\langle\langle A\rangle\rangle$. When the underlying semiring S is a field, we moreover have

$$\sigma^* = (1-\sigma)^{-1}$$

whenever $o(\sigma) = 0$, and

$$\sigma^{-1} = (1-\sigma)^*$$

whenever $o(\sigma) = 1$.

2.4.2 Rational power series and expressions

We now define the set of S-*rational* power series in a fixed set $\Sigma \subseteq S\langle\langle A\rangle\rangle$. Let $S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$ be the smallest subset of $S\langle\langle A\rangle\rangle$ such that:

1. $\Sigma \subseteq S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$;
2. $A \subseteq S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$ and $S \subseteq S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$;
3. $S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$ is closed under the operators $+$ and $\cdot$; and
4. if $\sigma \in S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$ and σ is proper, then $\sigma^* \in S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$.

Henceforth, we call any $\tau \in S\langle\langle A\rangle\rangle$ S-*rational in* Σ whenever $\tau \in S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$. In the case where $\Sigma = \varnothing$, we simply call τ S-*rational.* When S is a field, by the duality between the star and inverse operators, this notion can easily be seen to correspond to the 'classical' notion in terms of division.

This gives rise to an induction principle: by showing that a property holds for all $\sigma \in \Sigma$, $a \in A$, and $s \in S$, and that whenever it holds for σ and τ, it holds for $\sigma + \tau$, $\sigma\tau$ and (if defined) σ^*, one proves that this property holds for all $\sigma \in S_{\mathrm{rat}(\Sigma)}\langle\langle A\rangle\rangle$.

We now turn to an alternate characterization of the S-rational power series in terms of rational expressions, which can be regarded as 'semi-syntactic'. This characterization is in fact not needed for the results in this chapter, but will be useful for one of the characterizations of constructively algebraic power series (which generalize the context-free languages) in Chapter 3.

Given a semiring S and a fixed set Σ of power series in $S\langle\langle A\rangle\rangle$, let us simultaneously define the class of $\mathfrak{Rat}_S(\Sigma)$ rational expressions, as well as the valuation function $\mathbf{val} : \mathfrak{Rat}_S(\Sigma) \to S\langle\langle A\rangle\rangle$ as follows:

1. For each $a \in A$, there is an expression $a \in \mathfrak{Rat}_S(\Sigma)$, with $\mathbf{val}(a) = a$;
2. for each $s \in S$, there is an expression $s \in \mathfrak{Rat}_S(\Sigma)$, with $\mathbf{val}(k) = k$;
3. for each $\sigma \in \Sigma$, there is an expression $[\sigma] \in \mathfrak{Rat}_S(\Sigma)$, with $\mathbf{val}([\sigma]) = \sigma$;
4. if there are expressions $t, u \in \mathfrak{Rat}_S(\Sigma)$, there is an expression $(t \oplus u) \in \mathfrak{Rat}_S(\Sigma)$ and $(t \otimes u) \in \mathfrak{Rat}_S(\Sigma)$, with $\mathbf{val}(t \oplus u) = \mathbf{val}(t) + \mathbf{val}(u)$ and $\mathbf{val}(t \otimes u) = \mathbf{val}(t)\mathbf{val}(u)$;
5. and if there is an expression $t \in \mathfrak{Rat}_S(\Sigma)$ and $O(\mathbf{val}(s)) = 0$, then there is an expression $t^{\bar{*}} \in \mathfrak{Rat}_S(\Sigma)$, with $\mathbf{val}(t^{\bar{*}}) = \mathbf{val}(t)^*$.

Alternately, we could describe $\mathfrak{Rat}_S(\Sigma)$ using a Backus Naur form-like notation

$$\mathfrak{Rat}_S(\Sigma) \ni t \ ::= \ s \in S \,|\, a \in A \,|\, [\sigma],\, \sigma \in \Sigma \,|\, (t \oplus t) \,|\, (t \otimes t) \,|\, t^{*},\ \mathbf{val}(t) = 0$$

but note that this description again requires **val** to be inductively defined at the same time. In fact, this definition can be seen as an instance of *induction-recursion*: a more general introduction to this topic, addressing its foundational aspects, can be found in [GH11].

This directly gives us the following lemma, which can be proven directly using elementary induction techniques:

Lemma 2.20. *Given any $\sigma \in S\langle\!\langle A\rangle\!\rangle$ and $\Sigma \subseteq S\langle\!\langle A\rangle\!\rangle$, σ is S-rational in Σ if and only if there is a $s \in \mathfrak{Rat}_S(\Sigma)$ such that* $\mathbf{val}(s) = \sigma$.

We will now turn to the special case where Σ is the empty set: in this case, we simply write $\mathfrak{Rat}_S$ instead of $\mathfrak{Rat}_S(\varnothing)$. We can define an (infinite) S-automaton $(\mathfrak{Rat}_S, o, \delta)$ with $\mathfrak{Rat}_S$ as carrier set, using the following system of behavioural differential equations:

t	$o(t)$	t_a
$s \in S$	s	0
$b,\ b \in A$	0	**if** $b = a$ **then** u **else** 0
$(u \oplus v)$	$\hat{o}(u) + \hat{o}(u)$	$(u_a \oplus v_a)$
$(u \otimes v)$	$\hat{o}(u) \wedge \hat{o}(v)$	$((u_a \otimes v) \oplus (\hat{o}(u) \otimes v_a))$
(u^{*})	1	$(u_a \otimes u^{*})$

(2.7)

and it is now easy to check that **val**, as defined before, makes the following diagram commute:

$$\begin{tikzcd}
\mathfrak{Rat}_S \arrow[r, "\mathbf{val}"] \arrow[d, "{(o,\delta)}"'] & S\langle\!\langle A\rangle\!\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times (\mathfrak{Rat}_S)^A \arrow[r, "1_S \times \mathbf{val}^A"] & S \times S\langle\!\langle A\rangle\!\rangle^A
\end{tikzcd}$$

Hence, it follows that the function **val** coincides with the unique morphism from $(\mathfrak{Rat}_S, o, \delta)$ into the final automaton $(S\langle\!\langle A\rangle\!\rangle, O, \Delta)$.

In the case where $\Sigma \neq \varnothing$, things are slightly more complicated: we will return to these matters in the next chapter.

2.4.3 The Kleene-Schützenberger-Eilenberg theorem

We now turn to the generalization of Kleene's theorem to formal power series by Schützenberger and Eilenberg, stating the equivalence between S-rational and S-recognizable power series.

Proposition 2.21. *If σ is S-rational, then σ is S-recognizable.*

Proof. Use the induction principle for rational power series. For the base cases, it is clear that $\{a\}$ and $\{s\}$ are linearly stable. For the inductive cases, assume that σ and τ are S-recognizable. Then there must be some linearly stable Σ with $\sigma, \tau \in \Sigma$.

It now is easily verified that

$$\Sigma \cup \{\upsilon\tau \,|\, \upsilon \in \Sigma\} \cup \{\sigma + \tau\}$$

is S-linearly stable again (i.e. all its derivatives are again linear combinations of elements from this set), so $\sigma\tau$ and $\sigma + \tau$ are again S-recognizable.

Finally, if $o(\sigma) = 0$, then

$$\{\sigma^*\} \cup \{\tau\sigma^* \,|\, \tau \in \Sigma\}$$

is S-linearly stable again, and hence σ^* is S-recognizable.

It now follows from the induction principle that any σ which is S-rational, is also S-recognizable. □

For the converse, we will first present a lemma at a more general level than needed to prove the equivalence between S-rational and S-recognizable series. In the next chapter, we will show that we can use this lemma as a basis for a construction of the Greibach normal form. For the results in the present chapter, however, we only need the instance of the following proposition of S-rational series over the empty set, rather than S-rational series over an arbitrary X.

Lemma 2.22. *Given a $k \in \mathbb{N}$, if for all $i, j \leq k$, r_{ij} is a proper S-rational series over X, and each p_i is a S-rational series over X, the system of equations given by*

$$x_i = p_i + \sum_{j=0}^{k} r_{ij} x_j$$

for $i \in 0 \ldots k$ has a unique solution, and each x_i is S-rational over X.

Proof. Natural induction on k.

If $k = 0$, then we simply have one equation

$$x_0 = p_0 + r_{00}x_0$$

and Arden's rule (2.4) now gives

$$x_0 = (r_{00})^* p_0,$$

as the unique solution and x_0 is S-rational over X by the closure properties of rational series.

If $k = n + 1$, assume the inductive hypothesis that the proposition holds for systems with n variables. We note

$$x_k = p_k + \sum_{j=0}^{n} r_{kj}x_j + r_{kk}x_k$$

and hence by Arden's rule

$$x_k = (r_{kk})^* \left(p_k + \sum_{j=0}^{n} r_{kj}x_j \right). \tag{2.8}$$

Now, for $i \le n$,

$$\begin{aligned} x_i &= p_i + \sum_{j=0}^{n} r_{ij}x_j + r_{ik}x_k \\ &= p_i + \sum_{j=0}^{n} r_{ij}x_j + r_{ik}(r_{kk})^* \left(p_k + \sum_{j=0}^{n} r_{kj}x_j \right) \\ &= p_i + r_{ik}(r_{kk})^* p_k + \sum_{j=0}^{n} (r_{ij} + r_{ik}(r_{kk})^* r_{kj})x_j. \end{aligned}$$

Setting

$$q_i := p_i + r_{ik}(r_{kk})^* p_k \qquad \text{and} \qquad s_{ij} := r_{ij} + r_{ik}(r_{kk})^* r_{kj},$$

we now obtain for all $i \le n$

$$x_i = q_i + \sum_{j=0}^{n} s_{ij}x_j,$$

a similar system in one variable less, as it's easy to see that each q_i is S-rational over X and each s_{ij} is S-rational over X and proper. By the inductive hypothesis, this system has a unique solution in $x_0, \ldots, x_n$, and filling in this solution in (2.8), which is again unique by Arden's rule, it becomes clear that x_k is S-rational over X too. $\square$

We can now state the converse:

Theorem 2.23. *If $\sigma \in S\langle\langle A\rangle\rangle$ is S-recognizable, then it is rational.*

Proof. If σ is S-recognizable, then it is the unique solution to a system of equations for $x_0, \ldots, x_k$ of the form

$$x_i = o_i + \sum_{a \in A} a \sum_{j=0}^{k} k_{aij} x_j$$

with all o_i and k_{aij} scalars in S.

Now observe

$$x_i = o_i + \sum_{j=0}^{k} \left(\sum_{a \in A} a k_{aij} \right) x_j$$

and, as o_i is clearly S-rational and $\sum_{a \in A} a k_{aij}$ is clearly S-rational and proper for all k, i, and j, it follows by Lemma 2.22 that σ is S-rational. $\square$

2.5 Bisimulation up to linearity

In the case of linear automata, we can establish equality under the final homomorphism using the notion of a bisimulation up to linearity. The general idea here is that, given two linear automata X and Y, whenever two elements (s, t) are related by a bisimulation up to linearity $R \subseteq X \times Y$, their outputs are identical, and moreover, the derivatives $s_a = t_a$ are linear combinations of projections of related pairs again. The idea of bisimulations up to linearity is closely related to the more abstract, categorically defined, notion of coalgebraic bisimulation up-to: for more background on this topic, we refer to [RBR13].

In order to make this notion precise, we will first need to define the relation ΣR, which can, in a way, be regarded as a linear extension of R. We can formally specify ΣR as

$$\Sigma R := \{(\alpha_X \circ \mathrm{Lin}_S(\pi_1), \alpha_Y \circ \mathrm{Lin}_S(\pi_2))(r) \mid r \in \mathrm{Lin}_S(R)\};$$

$\mathrm{Lin}_S(R)$ here is simply the set of formal S-linear combinations of elements of R; however, because X and Y themselves have a S-module structure, we can canonically transform elements of $\mathrm{Lin}_S(R)$ into elements of $X \times Y$: this is done by the function

$$(\alpha_X \circ \mathrm{Lin}_S(\pi_1), \alpha_Y \circ \mathrm{Lin}_S(\pi_2)).$$

Here mappings $\alpha_X : \mathrm{Lin}_S(X) \to X$ and $\alpha_Y : \mathrm{Lin}_S(Y) \to Y$ are as defined in Section A.3; and $\mathrm{Lin}_S(\pi_1)$ and $\mathrm{Lin}_S(\pi_2)$ are the liftings of the projection morphisms π_1 and π_2 over the functor $\mathrm{Lin}_S(-)$, going from $\mathrm{Lin}_S(R)$ to $\mathrm{Lin}_S(X)$ and $\mathrm{Lin}_S(Y)$, respectively. As a result, ΣR is again a subset of $X \times Y$.

Given linear automata X and Y, we will now call a relation $R \subseteq X \times Y$ a *bisimulation up to linearity* whenever the following conditions are satisfied:

1. for all $(x, y) \in R$, $o(x) \in o(y)$, and
2. for all $(x, y) \in R$ and $a \in A$, $x_a \; \Sigma R \; y_a$.

We will next show that a relation R is a bisimulation up to linearity whenever ΣR is a bisimulation. In order to show this, hovever, we first to prove a few auxiliary results.

Lemma 2.24. *Given any $(p, q) \in X \times Y$, we have $(p, q) \in \Sigma R$ if and only if there exists an index set I, and three mappings assigning elements $i \in I$ to coefficients $s_i \in S$ and elements $p_i \in X$ and $q_i \in Y$, such that*

$$p = \sum_{i \in I} s_i p_i \qquad \textit{and} \qquad q = \sum_{i \in I} s_i q_i,$$

and for each $i \in I$, $(p_i, q_i) \in R$.

Proof. Assume $(p, q) \in \Sigma R$. Then, by the definition of ΣR, there has to be a $r \in \mathrm{Lin}_S(R)$, such that

$$p = \alpha_X \circ \mathrm{Lin}_S(\pi_1)(r) \qquad \text{and} \qquad q = \alpha_Y \circ \mathrm{Lin}_S(\pi_2)(r).$$

However, because of the definition of $\mathrm{Lin}_S(-)$, it is clear that there must be some index set I and mappings assigning elements $i \in I$ to coefficients $s_i \in S$ and elements $r_i \in R$, such that

$$r = \sum_{i \in I} s_i r_i.$$

Writing each r_i as (p_i, q_i), we now obtain

$$p = \alpha_X \circ \mathrm{Lin}_S(\pi_1)\left(\sum_{i \in I} s_i(p_i, q_i)\right) = \alpha_X\left(\sum_{i \in I} s_i p_i\right) = \sum_{i \in I} s_i p_i$$

and similarly for q. The vanishing of the α here can be understood as representing a move from elements of $\mathrm{Lin}_S(X)$, represented as formal sums, to the corresponding real sums in X. For the other direction, we can simply define

$$r = \sum_{i \in I} s_i(p_i, q_i),$$

observe that $r \in \mathrm{Lin}_S(R)$, and verify that $p = \alpha_X \circ \mathrm{Lin}_S(\pi_1)(r)$ and $q = \alpha_Y \circ \mathrm{Lin}_S(\pi_2)(r)$. $\square$

From this lemma we directly obtain the result $R \subseteq \Sigma R$:

Corollary 2.25. *For all $R \subseteq X \times Y$, $R \subseteq \Sigma R$.*

Proof. Given a $(p, q) \in R$, consider the singleton index set $I = \{1\}$, together with the mappings $s_1 = 1$, $p_1 = p$, and $q_1 = q$, and apply Lemma 2.24 to obtain $(p, q) \in \Sigma R$. $\square$

We now are equipped with the prerequisites needed to establish the desired equivalence:

Proposition 2.26. *A relation $R \subseteq X \times Y$ is a bisimulation up to linearity if and only if ΣR is a bisimulation.*

Proof. First assume that ΣR is a bisimulation. If $(p, q) \in R$, then $(p, q) \in \Sigma R$, and hence we obtain both $o(p) = o(q)$ and $p_a \, \Sigma R \, q_a$ directly.

For the other direction, assume that $R \subseteq X \times Y$ is a bisimulation up to linearity. Now, take any $(p, q) \in \Sigma R$. By Lemma 2.24, we have an index set I and mappings assigning s_i, p_i and q_i to each $i \in I$, such that

$$p = \sum_{i \in I} s_i p_i \qquad \text{and} \qquad q = \sum_{i \in I} s_i q_i.$$

and for each $i \in I$, $(p_i, q_i) \in R$. We now have

$$o(p) = o\left(\sum_{i \in I} s_i p_i\right) = \sum_{i \in I} s_i o(p_i) = \sum_{i \in I} s_i o(q_i) = o\left(\sum_{i \in I} s_i q_i\right) = o(q)$$

and

$$p_a = \left(\sum_{i \in I} s_i p_i \right)_a = \sum_{i \in I} s_i (p_i)_a \quad \Sigma R \quad \sum_{i \in I} s_i (q_i)_a = \left(\sum_{i \in I} s_i q_i \right)_a = q_a,$$

and the proof is complete. □

We will conclude this section with an elementary but noteworthy result: just like in the case of ordinary bisimulation, elements related by a bisimulation up to linearity have the same semantics in the final automaton.

Proposition 2.27. *If $R \subseteq X \times Y$ is a bisimulation up to linearity, and $(p, q) \in R$, then $[\![p]\!] = [\![q]\!]$.*

Proof. Use the fact that $R \subseteq \Sigma R$, the fact that ΣR is a bisimulation, and Proposition 2.7. □

3

Context-free languages and algebraic power series

In this chapter, we will extend the approach from the previous chapter in order to encapsulate context-free languages as well as (constructively) algebraic power series, the usual generalization of context-free languages. The material in this chapter builds on both the coalgebraic approach to rational power series, as laid out in the previous chapter, and the traditional algebraic theory of context-free languages and (constructively) algebraic power series, parts of which are presented in e.g. [CS63], [PS09], and [SS78].

We will start in Section 3.1 by introducing the notion of a polynomial system of behavioural differential equations, and give a canonical method of extending any such system into a weighted automaton. This (infinite) weighted automaton then can, using the techniques from the previous chapter, be determinized into a linear automaton. We introduce the notion of a *constructively algebraic* power series, as a power series that can be characterized using a finite polynomial system of behavioural differential equations, and give a semantic characterization similar to that of Proposition 2.16.

In Section 3.2, we restrict ourselves to the situation where the underlying semiring is $\mathbb{B}$. After introducing context-free grammars and (leftmost) derivations in such grammars, we show a correspondence between languages derivable from a word $s \in X^*$, and the final coalgebra semantics $[\![s]\!]$ of the linear automaton generated from such a grammar.

In Section 3.3, we return to the more general point of view, and consider the more classical view of *systems of equations*, as can be found in e.g. [PS09]. We give a method of transforming any system that satisfies the coniditon of being *proper* into a polynomial system of behavioural differntial equations. Polynomial systems of behavioural differential equations directly correspond to systems of equations in Greibach normal form: thus, we can regard this transformation as a construction of the Greibach normal form.

Finally, in Section 3.4, we turn to a classical result due to Chomsky and Schützenberger, stating that $\mathbb{N}$-algebraic power series are precisely the counting functions of unambiguous context-free languages, and present it using the notion of bisimulation up to linearity. We furthermore show a few examples of counting problems that can be presented coalgebraically as a direct result of this theorem.

3.1 Polynomials and polynomial systems

This section will start off with a formal treatment of polynomials, which will be the building blocks for the remainder of the material in this chapter, in a way similar to the way formal linear combinations were used in Chapter 2. Next, we introduce polynomial systems of behavioural differential equations together with their solutions, which will constitute a model for the context-free languages as well as their generalizations to arbitrary commutative semirings.

3.1.1 Polynomials

We start off with a treatment of polynomials as formal objects, in direct analogy with the treatment of formal linear combinations in Chapter 2. Some, but not all, of the observations in this subsection can also be found in e.g. Chapter 1 of [BR11]. From a mathematical point of view, the polynomials that we will use can be regarded as polynomials in a set of *noncommuting variables*, with coefficients in a (commutative) semiring S. Wherever the commutativity of S is relevant, this will be indicated.

We define the set $S\langle X\rangle$ of polyniomials with coefficients in S and variables in X as equal to

$$\mathrm{Lin}_S(X^*),$$

the set of S-linear combinations of words over X. We can now regard $S\langle X\rangle$ as a functor again, obtained by composing the functors $\mathrm{Lin}_S(-)$ and $-^*$.

This definition also immediately implies the following characterization:

$$S\langle X\rangle := \{f : X^* \to S \mid \mathbf{supp}(f) \text{ is finite}\}$$

As a result of this characterization, it is also possible to regard $S\langle X\rangle$ as the subset of those elements in $S\langle\langle X\rangle\rangle$ with finite support. If we do this, it is easy to see that $S\langle X\rangle$ is closed under the semiring operations $\cdot$ and $+$ (in other words, if σ and τ have finite support: then so do $\sigma \cdot \tau$ and $\sigma + \tau$) as well as under the scalar product. As a result, $S\langle X\rangle$ can, in all cases, be seen as a subsemiring and a sub-S-module of $S\langle\langle X\rangle\rangle$.

Given any set X, define $\eta^{\mathrm{p}}_X : X \to S\langle X\rangle$ pointwise on w by

$$[\eta^{\mathrm{p}}_X(x) \Downarrow w] = \textbf{if } w = 1 \textbf{ then } x \textbf{ else } 0 \tag{3.1}$$

for all $x \in X$. This definition again corresponds to the definition

$$\eta^{\mathrm{p}} = \eta^{\mathrm{l}}_{X^*} \circ \eta^{\mathrm{w}}_X.$$

Regarding polynomials as linear combinations of words, we can define functions f on polyniomials by 1) specifying the value of f for the empty word, 2) inductively specifying the value $f(xw)$ in terms of $f(w)$ for any $x \in X$ and $w \in X^*$, and 3) specifying the value of f for all linear combinations.

For example, define the function $\alpha^{\mathrm{p}} : S\langle S\langle\!\langle X\rangle\!\rangle\rangle \to S\langle\!\langle X\rangle\!\rangle$ is defined by:

$$\begin{aligned} \alpha^{\mathrm{p}}(1) &= 1 \\ \alpha^{\mathrm{p}}(xw) &= x \cdot \alpha^{\mathrm{p}}(w) \\ \alpha^{\mathrm{p}}\left(\sum_{i=1}^{n} s_i w_i\right) &= \sum_{i=1}^{n} s_i \alpha^{\mathrm{p}}(w_i) \end{aligned}$$

in terms of the semiring structure of $S\langle\!\langle A\rangle\!\rangle$. This function can be seen as evaluating, or interpreting, polynomials of power series (regarded as formal objects) as (or to) the corresponding power series.

When S is commutative, the function α in fact has a *unique mapping property* (which will be considered in a more general setting in Chapter 7), which can be stated as follows: given a commutative semiring S, and a function f from an arbitrary set X to $S\langle\!\langle A\rangle\!\rangle$, there is a unique semiring morphism $f^\sharp$ making the diagram

$$\begin{tikzcd} X \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "f"'] & S\langle X\rangle \arrow[dl, dotted, "f^\sharp"] \\ S\langle\!\langle A\rangle\!\rangle & \end{tikzcd} \tag{3.2}$$

commute. More specifically, $f^\sharp$ can be specified by the equation

$$f^\sharp = \alpha^{\mathrm{p}} \circ S\langle f\rangle. \tag{3.3}$$

In fact, the extension $^\sharp$ can be obtained as a composition of the extiensions $\hat{}$ and $^-$,

$$f^\sharp = \hat{\bar{f}},$$

or in other words, $f^\sharp$ can be seen as the unique S-linear mapping extending the unique monoid morphism extending f. This unique mapping property, however, fails when S is not commutative:

Given any set $\Sigma \subseteq S\langle\langle A\rangle\rangle$, we call a formal power series σ *polynomial in* Σ, whenever there is a $s \in S\langle\Sigma\rangle$, such that $\alpha^{\mathrm{p}}(s) = \sigma$.

Remark 3.1. This unique mapping property fails when S is not commutative. As a counterexample, consider any semiring S that is not commutative, i.e. that has elements $s, t \in S$ such that $st \neq ts$.

Now take the set $X = \{x, y\}$ together with the mapping $f : X \to S\langle\langle A\rangle\rangle$ defined by $f(x) = i(s)$ and $f(y) = i(1)$. Note that in $S\langle X\rangle$ there are elements x and ty; the multiplication inherited from $S\langle\langle A\rangle\rangle$ now gives $(x)(ty) = txy$. In order for the mapping $f^\sharp$ to be a semiring morphism, it has to satisfy

$$f^\sharp(x)f^\sharp(ty) = f^\sharp((x)(ty)),$$

however note that

$$f^\sharp(x)f^\sharp(ty) = i(s)i(t) \neq i(t)i(s) = i(ts) = f^\sharp(txy) = f^\sharp((x)(ty))$$

so, in this case, there is *no* possiblility of a semiring morphism $f^\sharp$ extending f. In order to establish the existence of a unique extension, we need to rely on the property that, for an arbitrary $\sigma \in S\langle\langle A\rangle\rangle$ and an arbitrary $s \in S$, $i(s)\sigma = \sigma i(s)$, which holds precisely in the case when S is commutative. When S is not commutative, we do still have a semiring structure on $S\langle X\rangle$, but the property established in diagram (3.2) fails.

3.1.2 Polynomial systems and their coalgebraic semantics

We now turn to a format for *polynomial systems of behavioural differential equations*, where each derivative is given as a polynomial over the set of variables. With finite systems in this format format, which is based on work presented in [WBR13] and [WBR14], we can describe classes of formal power series, which we call *constructively algebraic*. Later in this chapter, we will see a number of alternate characterizations of these classes.

Fixing a finite alphabet A, a polynomial system of behavioural differential equations over a semiring S consists of a triple

$$(X, o, \delta)$$

where

1. X is a set, to be regarded as a set of *variables* or *nonterminals*;
2. $o : X \to S$ is, as in the case of automata, an output function; and
3. $\delta : X \to S\langle X\rangle^A$ is the transition function, describing each of the possible derivatives of the set X as a polynomial.

We can, also, regard these systems of equations as coalgebras of the functor $S \times S\langle -\rangle^A$, or diagrammatically:

$$X \overset{(o,\delta)}{\to} S \times S\langle X\rangle^A$$

Example 3.2. As an example, consider the polynomial system of behavioural differential equations over the alphabet $\{a, b\}$ and the set of variables $\{x, y\}$ specified by:

$$\begin{array}{lll} o(x) = 1 & x_a = xy & x_b = 0 \\ o(y) = 0 & y_a = 0 & y_b = 1 \end{array}$$

We will see soon that x can be interpreted as the language $\{a^n b^n \mid n \in \mathbb{N}\}$, while y can be interpreted as the singleton language $\{b\}$.

In order to be able to give meaning to these systems of behavioural differential equations, we first require a method of transforming such a system into an automaton (or, equivalently, a coalgebra for the functor $S \times -^A$). We will do this using an extension of the determinization method presented in Section 2.3.

To start, we extend (o, δ) into a pair of mappings $(\bar{o}, \bar{\delta})$

$$(\bar{o}, \bar{\delta}) : X^* \to S \times S\langle X\rangle^A$$

specifying output values and derivatives of *words* over X, by means of the inductive definition

$$\begin{array}{rr} \bar{o}(1) = 1 & 1_a = 0 \\ \bar{o}(xw) = o(x)\bar{o}(w) & (xw)_a = x_a w + o(x) w_a \end{array}$$

for all $x \in X$, $w \in X^*$, and $a \in A$.

We can see this inductive definition as an instance of a *product rule*, relating to Brzozowski derivatives in a similar manner as the familiar product rule relates to ordinary function derivatives. We can now prove that this product rule can easily be extended from products of an alphabet symbol and a word, to products of arbitrary words:

Proposition 3.3. *Given a polynomial system of behavioural differential equations (X, o, δ) over an arbitrary semiring S, for all $v, w \in X^*$, the equations*

$$\bar{o}(vw) = \bar{o}(v)\bar{o}(w) \qquad \textit{and} \qquad (vw)_a = v_a w + \bar{o}(v) w_a$$

hold w.r.t. the extension $(\bar{o}, \bar{\delta})$.

Proof. Induction on the length of v.

If $v = 1$, then

$$\bar{o}(vw) = \bar{o}(1w) = \bar{o}(w) = \bar{o}(1)\bar{o}(w)$$

and

$$(vw)_a = (1w)_a = w_a = 0w + 1w_a = v_a w + \bar{o}(v) w_a.$$

If $v = xu$ for $x \in X$ and $u \in X^*$, use the inductive hypothesis that

$$\bar{o}(uw) = \bar{o}(u)\bar{o}(w) \qquad \text{and} \qquad (uw)_a = u_a w + \bar{o}(u) w_a$$

and now observe

$$\bar{o}(vw) = \bar{o}(xuw) = o(x)\bar{o}(uw) = o(x)\bar{o}(u)\bar{o}(w) = \bar{o}(xu)\bar{o}(w) = \bar{o}(v)\bar{o}(w)$$

and

$$\begin{aligned}
(vw)_a &= (xuw)_a \\
&= x_a(uw) + o(x)(uw)_a \\
&= x_a uw + o(x)(u_a w + \bar{o}(u) w_a) \\
&= x_a uw + o(x) u_a w + o(x)\bar{o}(u) w_a \\
&= (xu)_a w + \bar{o}(xu) w_a \\
&= v_a w + \bar{o}(v) w_a,
\end{aligned}$$

completing the proof. □

Now, because $S\langle X\rangle = \mathrm{Lin}_S(X^*)$, the inductive extension presented above simply gives a linear system

$$(\bar{o}, \bar{\delta}) : X^* \to \mathrm{Lin}_S(X^*)^A.$$

As a result, we can at this stage simply apply the determinization method from Section 2.3, obtaining a (deterministic) S-linear automaton $(S\langle X\rangle, \hat{o}, \hat{\delta})$. This automaton again satisfies (a more general version of) Brzozowski's product rule, this time defined on polynomials. Note that the following proposition again requires S to be a commutative semiring.

Proposition 3.4. *Given a polynomial system of behavioural differential equations (X, o, δ) over a commutative semiring S, for all polynomials $s, t \in S\langle X\rangle$ and any $a \in A$, the equations*

$$\hat{o}(st) = \hat{o}(s)\hat{o}(t) \qquad \textit{and} \qquad (st)_a = s_a t + \hat{o}(s)t_a$$

hold w.r.t. the extension $(S\langle X\rangle, \hat{o}, \hat{\delta})$.

Proof. First recall the identities

$$s = \sum_{w\in A^*} [s \Downarrow w]w \qquad \text{and} \qquad t = \sum_{w\in A^*} [t \Downarrow w]w.$$

Now observe

$$\begin{aligned}
\hat{o}(st) &= \hat{o}\left(\sum_{v\in X^*}[s \Downarrow v]v \sum_{w\in X^*}[t \Downarrow w]w\right)\\
&= \hat{o}\left(\sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w]vw\right)\\
&= \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w]\bar{o}(vw)\\
&= \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w]\bar{o}(v)\bar{o}(w)\\
&= \sum_{v\in X^*}[s \Downarrow v]\bar{o}(v) \sum_{w\in X^*}[t \Downarrow w]\bar{o}(w)\\
&= \hat{o}(s)\hat{o}(t)
\end{aligned}$$

and

$$\begin{aligned}
(st)_a &= \left(\sum_{v\in X^*}[s \Downarrow v]v \sum_{w\in X^*}[t \Downarrow w]w\right)_a\\
&= \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w](vw)_a\\
&= \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w](v_a w + o(v)w_a)\\
&= \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w]v_a w + \sum_{v,w\in X^*}[s \Downarrow v][t \Downarrow w]o(v)w_a
\end{aligned}$$

$$\begin{aligned}
&= \sum_{v\in X^*}[s \Downarrow v]v_a \sum_{w\in X^*}[t \Downarrow w]w + \sum_{v\in X^*}[s \Downarrow v]\bar{o}(v)\sum_{w\in X^*}[t \Downarrow w]w_a \\
&= \sum_{v\in X^*}[s_a \Downarrow v]v \sum_{w\in X^*}[t \Downarrow w]w + \sum_{v\in X^*}[s \Downarrow v]\bar{o}(v)\sum_{w\in X^*}[t_a \Downarrow w]w \\
&= s_a t + \hat{o}(s)t_a,
\end{aligned}$$

and the proof is complete. □

Now we can combine any system of equations of the form $(X, o, \delta) : X \to S \times S\langle X\rangle^A$, its extension $(S\langle X\rangle, \hat{o}, \hat{\delta})$, and the unique mapping $[\![-]\!]$ into the final S-automaton, in the following diagram:

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "{(o,\delta)}"'] & S\langle X\rangle \arrow[r, dotted, "{[\![-]\!]}"] \arrow[dl, "{(\hat{o},\hat{\delta})}"'] & S\langle\!\langle A\rangle\!\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times S\langle X\rangle^A \arrow[rr, dotted, "1_S \times {[\![-]\!]}^A"] & & S \times S\langle\!\langle A\rangle\!\rangle^A
\end{tikzcd}$$

We will henceforth, given a polynomial system of behavioural differential equations, call the composition of η^{p}_X (as defined in (3.1)) and the final homomorphism $[\![-]\!]$ the *solution* to this system.

We will call a formal power series σ over a commutative semiring S and an alphabet A *constructively algebraic* whenever there is a finite polynomial system of behavioural differential equations (X, o, δ), and an $x \in X$, such that $[\![\eta^{\mathrm{p}}_X(x)]\!] = \sigma$.

The term 'constructively algebraic' is a translation of the French *algèbrique constrictif* used in [Fli74]; in most of the automatic theory on algebraic power series, e.g. [CS63], [SS78], and [PS09], this class is simply called 'algebraic'.

Lemma 3.5. *Given any polynomial system (X, o, δ) with coefficients in S, if a function $f : X \to Q$ makes the diagram*

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "{(o,\delta)}"'] & S\langle\!\langle A\rangle\!\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times S\langle\Sigma\rangle^A \arrow[r, "1_S \times (f^\sharp)^A"] & S \times S\langle\!\langle A\rangle\!\rangle^A
\end{tikzcd}$$

commute, then the unique semiring morphism $f^\sharp : S\langle X\rangle \to S\langle\langle A\rangle\rangle$ *extending* f *makes the diagram*

$$\begin{tikzcd} X \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "(o,\delta)"'] & S\langle X\rangle \arrow[r, "f^\sharp"] \arrow[dl, "(\hat{o},\hat{\delta})"'] & S\langle\langle A\rangle\rangle \arrow[d, "(O,\Delta)"] \\ S\times S\langle X\rangle^A \arrow[rr, "1_S\times (f^\sharp)^A"] & & S\times S\langle\langle A\rangle\rangle^A \end{tikzcd}$$

commute. Moreover, we have $f^\sharp = [\![-]\!]$.

Proof. Similar to Lemma 2.15, this proof relies on ensuring that $(O,\Delta)\circ f^\sharp = (1\times (f^\sharp)^A)\circ(\hat{o},\hat{\delta})$. To establish this, we first prove by induction on the length of w that, for all $w \in X^*$,

$$O(\bar{f}(w)) = \bar{o}(w) \qquad \text{and} \qquad \bar{f}(w)_a = f^\sharp(w_a),$$

in order to establish that the diagram

$$\begin{tikzcd} X \arrow[r, hook, "\eta^{\mathrm{w}}_X"] \arrow[d, "(o,\delta)"'] & X^* \arrow[r, "\bar{f}"] \arrow[dl, "(\bar{o},\bar{\delta})"'] & S\langle\langle A\rangle\rangle \arrow[d, "(O,\Delta)"] \\ S\times S\langle X\rangle^A \arrow[rr, "1_S\times (f^\sharp)^A"] & & S\times S\langle\langle A\rangle\rangle^A \end{tikzcd} \tag{3.4}$$

commutes.

If $|w| = 0$, then $w = 1$, and observe

$$O(\bar{f}(1)) = O(1) = 1 = o(1) = \bar{o}(1)$$

and

$$\bar{f}(1)_a = 1_a = 0 = f^\sharp(0) = f^\sharp(1_a).$$

If $|w| > 0$, then $w = xv$ for some $x \in X, v \in X^*$, and use the inductive hypothesis that the equalities hold for v. Now observe

$$O(\bar{f}(xw)) = O(f(x))O(\bar{f}(w)) = o(x)\bar{o}(w) = \bar{o}(xw)$$

and

$$\bar{f}(xw)_a = (f(x)\bar{f}(w))_a$$

$$\begin{aligned}
&= f(x)_a\bar{f}(w) + O(f(x))\bar{f}(w)_a\\
&= f^\sharp(x_a)f^\sharp(w) + o(x)f^\sharp(w_a)\\
&= f^\sharp(x_a w + o(x)w_a)\\
&= f^\sharp((xw)_a),
\end{aligned}$$

so the equalities again hold for av and the inductive proof is complete.

So (3.4) indeed commutes, and using the equalities $S\langle X\rangle = \mathrm{Lin}_S(X^*)$, $f^\sharp = \hat{f}^*$, and $\eta^{\mathrm{p}}_X = \eta^{\mathrm{l}}_{X^*} \circ \eta^{\mathrm{w}}_X$, we can now appeal to Lemma 2.15 to show that the diagram

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^{\mathrm{w}}_X"] \arrow[d, "(o,\delta)"'] & X^* \arrow[r, hook, "\eta^{\mathrm{l}}_{X^*}"] \arrow[dl, "(\bar{o},\bar{\delta})"'] & S\langle A\rangle \arrow[r, "f^\sharp"] \arrow[dll, "(\hat{o},\hat{\delta})"'] & S\langle\langle A\rangle\rangle \arrow[d, "(O,\Delta)"] \\
S\times S\langle X\rangle^A \arrow[rrr, "1_S\times (f^\sharp)^A"] & & & S\times S\langle\langle A\rangle\rangle^A
\end{tikzcd}$$

commutes to complete the proof. From the last diagram, $f^\sharp = [\![-]\!]$ is also immediate. □

Proposition 3.6. *Given a commutative semiring S, a formal power series $\sigma \in S\langle\langle A\rangle\rangle$ is constructively S-algebraic if and only if there is a finite set $\Sigma \subseteq S\langle\langle A\rangle\rangle$ with $\sigma\in\Sigma$, such that for each $\tau\in\Sigma$ and $a\in A$, τ_a is S-polynomial in Σ.*

Proof. Follows from the preceding lemma in the same manner as Proposition 2.16 follows from Lemma 2.15. □

Moreover, from the preceding lemma it directly follows that $[\![-]\!]$ is a semiring morphism:

Corollary 3.7. *Given a polynomial system of behavioural differential equations (X, o, δ) over a commutative semiring S, $[\![-]\!]$ is a semiring morphism w.r.t. the extension $(S\langle X\rangle, \hat{o}, \hat{\delta})$.*

We now turn to the observation that the constructively S-algebraic power series are closed under sum and product:

Corollary 3.8. *If σ and τ are constructively S-algebraic, then so are $\sigma+\tau$ and $\sigma\tau$.*

Proof. If σ and τ are constructively algebraic, then there are finite $\Sigma_0, \Sigma_1 \subseteq S\langle\langle A\rangle\rangle$ with $\sigma \in \Sigma_0$ and $\tau \in \Sigma_1$ such that for each $a \in A$, $\tau_0 \in \Sigma_0$, and $\tau_1 \in \Sigma_1$, $(\tau_0)_a$ and $(\tau_1)_a$ are polynomial in Σ_0 and Σ_1, respectively.

It now follows that for each $\tau \in \Sigma_2 := \Sigma_0 \cup \Sigma_1 \cup \{\sigma\tau, \sigma + \tau\}$ and each $a \in A$, τ_a is polynomial in Σ_2 again, so $\sigma\tau$ and $\sigma + \tau$ are again constructively S-algebraic. □

Example 3.9. Let us return now to the system from Example 3.2, which was given by the following system of behavioural differential equations:

$$\begin{array}{lll} o(x) = 1 & x_a = xy & x_b = 0 \\ o(y) = 0 & y_a = 0 & y_b = 1 \end{array}$$

We will now make the earlier claim, that x can be interpreted as the language

$$\{a^n b^n \mid n \in \mathbb{N}\},$$

precise. In order to do so, consider the following relation

$$R = \{(xy^k, \{a^n b^{n+k}\, n \in \mathbb{N}\}) \mid k \in \mathbb{N}\} \cup \{(y^k, \{b^k\}) \mid k \in \mathbb{N}\}$$

between $\mathbb{B}\langle X\rangle$ (or, equivalently, $\mathcal{P}_\omega(X)$) and $\mathbb{B}\langle\langle A\rangle\rangle$ (or, equivalently, $\mathcal{P}(A^*)$). To see that R is a bisimulation, take an arbitrary $(r, s) \in R$:

- If (r, s) is of the form $(xy^k, \{a^n b^{n+k} \mid n \in \mathbb{N}\})$, then either $k = 0$, giving

  $$o(r) = o(xy^k) = o(x) = 1 = o(\{a^n b^n \mid n \in \mathbb{N}\}) = o(s),$$

  or $k > 0$, giving

  $$o(r) = o(xy^k) = 0 = o(\{a^n b^{n+k} \mid n \in \mathbb{N}\}) = o(s).$$

  Furthermore,

  $$r_a = (xy^k)_a = xy^{k+1} \; R \; \{a^n b^{n+k+1} \mid n \in \mathbb{N}\} = \{a^n b^{n+k} \mid n \in \mathbb{N}\}_a = s_a$$

  and

  $$r_b = (xy^k)_b = y^{k-1} \; R \; \{b^{k-1}\} = \{a^n b^{n+k} \mid n \in \mathbb{N}\}_b = s_b,$$

  completing the case.

- If (r,s) is of the form $(y^k, \{b^k\})$, then
$$o(r) = \textbf{if } k = 0 \textbf{ then } 1 \textbf{ else } 0 = o(s),$$
$$r_a = 0 = s_a,$$
and
$$r_b = (y^k)_b = y^{k-1} \; R \; \{b^{k-1}\} = \{b^k\}_b = s_b,$$
completing this case, too.

From the fact that R is a bisimulation, it now follows directly that $[\![x]\!] = \{a^n b^n \mid n \in \mathbb{N}\}$, as previously claimed.

Example 3.10. As an example of a constructively algebraic power series that is not a language (over the semiring of the natural numbers and over a singleton alphabet), taken from [Rut02], consider the stream defined by the following equation:
$$o(x) = 1 \qquad x' = x^2$$
Its solution $[\![\eta(x)]\!]$ is the stream of Catalan numbers:
$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots \quad \text{(A000108)}$$
The nth element of this stream counts the number of well-bracketed words consisting of n pairs of opening and closing brackets. In Section 3.4, we will show how to derive the above equation from a context-free grammar representing pairs of brackets.

3.2 Context-free grammars and languages

In this section, we will restrict ourselves to the case where the underlying semiring is the Boolean semiring $\mathbb{B}$. We will connect polynomial systems of behavioural differential equations over this semiring to context-free grammars via leftmost derivations.

Recall that, in the case of the Boolean semiring $\mathbb{B}$, we have the following instantiations of the more general notions of linear combinations and polynomials:
$$\begin{array}{rcl} \mathcal{P}_\omega(X) & \cong & \mathrm{Lin}_{\mathbb{B}}(X) \\ \mathcal{P}_\omega(X^*)) & \cong & \mathbb{B}\langle X \rangle \end{array}$$

$$\mathcal{P}(X^*) \quad \cong \quad \mathbb{B}\langle\langle A\rangle\rangle$$

Given a finite alphabet A, a context-free grammar over A consists of a finite set X of *variables* or *nonterminals* together with a mapping:

$$p : X \to \mathcal{P}_\omega(X + A^*))$$

In line with conventions, we write

$$x \to w$$

to denote $w \in p(x)$, and call such a pair a *production rule* of the grammar (X, p).

Given a context-free grammar (X, p) and words $v, w \in (X + A)^*$, we write $v \Rightarrow w$ and say w is derivable from v in a single derivation step, whenever $v = v_1 x v_2$ and $w = v_1 u v_2$ for some production rule $x \to u$ for some $v_1, v_2 \in (X+A)^*$. We say that w is derivable from v in a single *leftmost* derivation step whenever $v_1 \in A^*$.

Let $\Rightarrow^*$ denote the reflexive and transitive closure of $\Rightarrow$. In general, if $v \Rightarrow^* w$, then w is derivable from v using only leftmost derivation steps. Therefore we can restrict our attention to leftmost derivations only. For a context-free grammar (X, p) and any variable $x \in X$, called the starting symbol, we define the language $\mathcal{L}(x) \in \mathcal{P}(A^*)$ generated by (X, p) from x as:

$$\mathcal{L}(x) := \{w \in A^* \mid x \Rightarrow^* w\}$$

A language $L \in \mathcal{P}(A^*)$ is called *context-free* if there exists a context free grammar (X, p) and a variable $x \in X$, such that $L = \mathcal{L}(x)$.

For our coalgebraic treatment of context-free languages it will be convenient to work with context-free grammars with production rules of a specific form. We say that a context-free grammar is in *Greibach normal form* if all of its production rules are of the form

$$x \to aw \quad \text{or} \quad x \to 1$$

where $a \in A$ is an alphabet symbol, and $w \in X^*$ is a (possibly empty) sequence of nonterminal symbols. It is well-known (see e.g. [Gre65]) that for every context-free language L, there exists a context-free grammar (X, p) in Greibach normal form, and some $x \in X$, such that $\mathcal{L}(x) = L$.

Given a context-free grammar (X, p) in Greibach normal form, we associate with it a polynomial system of behavioural differential equations (X, o, δ), defined by

$$o(x) = \textbf{if } x \to 1 \textbf{ then } 1 \textbf{ else } 0$$

and

$$x_a = \sum_{w \in A^* \,|\, x \to aw} w.$$

It is easy to see that this gives a one-to-one bijective correspondence between grammars in Greibach normal form, and finite polynomial systems of behavioural differential equations.

Lemma 3.11. *Given a context-free grammar (X, p) in Greibach normal form and any $s \in X^*$, we have $s \Rightarrow^* 1$ iff $\hat{o}(s) = 1$.*

Proof. Induction on the length of s. □

If $s \Rightarrow^* aw$ is a derivation of a word $w \in A^*$ from a monomial $s \in X^*$, a decomposition can be made:

$$s = uxz \Rightarrow^* xz \Rightarrow atz \Rightarrow aw$$

where $x \to at$ is the first rule used that is not of the form $y \to 1$ in the derivation.

We write $s \overset{\smile}{\to} at$ whenever there are decompositions $s = uxz$ and $t = vz$ for some $u, v, z \in X^*$, $x \in X$, such that $u \Rightarrow^* 1$ and $x \to av$. Hence, $s \Rightarrow^* aw$ if and only if there is a t such that $s \overset{\smile}{\to} at$ and $t \Rightarrow^* w$.

Lemma 3.12. *We have $t \in s_a$ iff $s \overset{\smile}{\to} at$.*

Proof. For the right to left direction, if $s \overset{\smile}{\to} at$, then there are $u, v, z \in X^*$ and $x \in X$ with $s = uxz$, $t = vz$ such that $u \Rightarrow^* 1$ and $x \to av$. From $x \to av$ we directly get $v \in x_a$. Now observe

$$t = vz \in \{uz \,|\, u \in x_a\} \cup \textbf{if } o(x) = 1 \textbf{ then } z_a \textbf{ else } 0 = (xz)_a$$

and hence also, because $u \Rightarrow^* 1$ and hence $\hat{o}(u) = 1$,

$$t \in (uxz)_a = s_a.$$

For the converse, use induction on the length of s.

If $s = 1$, then the antecedent $t \in s_a$ is always false because $1_a = 0$, so the implication holds.

If $s = xz$ for some $x \in X$, $z \in X^*$, then from $t \in s_a$ we either get $t = vz$ for some $v \in x_a$, or we have $o(x) = 1$ and $t \in z_a$. In the first case, we get $s \overset{\smile}{\to} at$ witnessed by the composition $s = 1xz$; in the second case, use the inductive hypothesis to obtain $z \overset{\smile}{\to} at$, now observe there has to be a decomposition $z = uyw$, $t = vw$ with $u \in X^*$ and $\hat{o}(u) = 1$, $y \to av$; now also $s = (xu)yw$ and $\hat{o}(xu) = 1$, giving $s \overset{\smile}{\to} at$. □

Lemma 3.13.
$$\mathcal{L}(s)_a = \bigcup_{t \in s_a} \{\mathcal{L}(t)\}$$

Proof. We have:

$$\begin{aligned}
w \in \mathcal{L}(s)_a \quad & \Leftrightarrow \quad aw \in \mathcal{L}(s) \\
& \Leftrightarrow \quad s \Rightarrow^* aw \\
& \Leftrightarrow \quad \exists t \in X^* : s \overset{\smile}{\to} at \wedge t \Rightarrow^* w \\
& \Leftrightarrow \quad \exists t \in s_a : t \Rightarrow^* w \\
& \Leftrightarrow \quad \exists t \in s_a : w \in \mathcal{L}(t) \\
& \Leftrightarrow \quad w \in \bigcup_{t \in s_a} \{\mathcal{L}(t)\}
\end{aligned}$$
□

This now leads to the main result, the proof of which depends on the bisimulation up to union principle, which is the instantiation of bisimulation up to linearity for the Boolean semiring $\mathbb{B}$.

Proposition 3.14. *For any context-free grammar (X, p) in Greibach normal form and any $s \in X^*$, we have $\mathcal{L}(s) = [\![s]\!]$.*

Proof. We have

$$o(s) = (\textbf{if } s \Rightarrow^* 1 \textbf{ then } 1 \textbf{ else } 0) = O(\mathcal{L}(s))$$

and

$$\mathcal{L}(s)_a \;=\; \bigcup_{t \in s_a} \{\mathcal{L}(t)\} \quad \cup R \quad \bigcup_{t \in s_a} \{t\} \;=\; s_a$$

so R is a bisimulation up to union. It follows that $\mathcal{L}(s) = [\![\mathcal{L}(s)]\!] = [\![s]\!]$. □

Because every context-free language is generated by a grammar in Greibach normal form, we thus directly obtain the following theorem:

Theorem 3.15. *A language $L \in \mathcal{P}(A^*)$ is context-free iff it is constructively $\mathbb{B}$-algebraic.*

3.3 Towards the Greibach normal form

The main aim of this section is to use Lemma 2.22, as a means to construct the Greibach normal form of any *proper* system of equations. This yields a new way of transforming any proper system of equations (or in the case of formal languages: any context-free grammar without empty word productions) into a corresponding system in Greibach normal form: unlike traditional methods of constructing the Greibach normal form (e.g. [HMU06] and [Ros67], this method does not rely on intermediate forms such as the Chomsky normal form.

For this construction, it is useful to first establish another characterization of constructively algebraic power series, replacing the condition of each derivative from a finite set Σ being *polynomial* in Σ, with the condition of each derivative being *rational* in Σ.

3.3.1 A characterization using rational series

For this construction, it is useful to first establish another characterization of constructively algebraic power series, replacing the condition in Proposition 3.6 of each derivative from a finite set Σ being *polynomial* in Σ, with the condition of each derivative being *rational* in Σ.

The equivalence between these conditions can be shown by adding a new variable for each starred subexpression occurring in the derivatives, thereby creating a polynomial system providing the same solution. Or more concisely: by gathering the starred subexpressions and then showing that each derivative of the union of Σ with the valuation **val** of all the starred subexpression is polyniomial in this set again.

Define the operator $\mathbf{star} : \mathfrak{Rat}_S(\Sigma) \to \mathcal{P}(\mathfrak{Rat}_S(\Sigma))$ inductively by:

1. $\mathbf{star}(a) = \mathbf{star}(s) = \mathbf{star}([\sigma]) = \varnothing$;
2. $\mathbf{star}(t \oplus u) = \mathbf{star}(t \otimes u) = \mathbf{star}(t) \cup \mathbf{star}(u)$; and
3. $\mathbf{star}(t^{\bar{*}}) = \mathbf{star}(t) \cup \{t^{\bar{*}}\}$.

Here, the underlying intuition is that, for any expression $s \in \mathfrak{Rat}_S(\Sigma)$, $\mathbf{star}(s)$ is the set of all expressions of the form $t^{\bar{*}}$ used in 'building' the expression s. We also define a 'star-pruning' operator

$$\mathbf{spr} : \mathfrak{Rat}_S(\Sigma) \to \mathfrak{Rat}_S(S\langle\langle A\rangle\rangle)$$

by

1. $\mathbf{spr}(a) = a$, $\mathbf{spr}(s) = s$, $\mathbf{spr}([\sigma]) = [\sigma]$;
2. $\mathbf{spr}(t \oplus u) = (\mathbf{spr}(t) \oplus \mathbf{spr}(u))$, $\mathbf{spr}(t \otimes u) = (\mathbf{spr}(t) \otimes \mathbf{spr}(u))$; and

3. $\mathbf{spr}(s^{\bar{*}}) = \mathbf{val}(s^{\bar{*}})$.

from which we can easily derive that

$$\mathbf{val}(s) = \mathbf{val}(\mathbf{spr}(s)) \tag{3.5}$$

for all expressions $s \in \mathfrak{Rat}_S(\Sigma)$. Furthermore, for any $s \in \mathfrak{Rat}_S(\Sigma)$,

$$\mathbf{spr}(s) \in \mathfrak{Rat}_S(\Sigma \cup \{\mathbf{val}(t) \,|\, t \in \mathbf{star}(s)\}).$$

This leads us to the following characterization of constructively algebraic power series:

Proposition 3.16. *Given a commutative semiring S, a formal power series $\sigma \in S\langle\langle A\rangle\rangle$ is constructively S-algebraic if and only if there is a finite set $\Sigma \subseteq S\langle\langle A\rangle\rangle$ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$ and $a \in A$, τ_a is S-rational in Σ.*

Proof. If σ is constructively S-algebraic, then there is a finite $\Sigma \subseteq S\langle\langle A\rangle\rangle$ such that for each $\tau \in \Sigma$ and each $a \in A$, τ_a is S-polynomial in Σ. It directly follows that each τ_a is S-rational in Σ.

Conversely, suppose that there is a finite $\Sigma \subseteq S\langle\langle A\rangle\rangle$, such that for each $\tau \in \Sigma$ and $a \in A$, τ_a is S-rational in Σ. If this is the case, there must be a $\delta : \Sigma \to \mathfrak{Rat}_S(\Sigma)^A$ such that for each $\tau \in \Sigma$, $\tau_a = \mathbf{val}(\delta(\bar{\tau}, a))$.

We now can extend δ to a S-automaton $(\mathfrak{Rat}_S, \hat{o}, \hat{\delta})$ using the behavioural differential equations:

t	$\hat{o}(t)$	t_a
$[\sigma]$, $\sigma \in \Sigma$	$O(\Sigma)$	$\delta(\sigma, a)$
$s \in S$	s	0
b, $b \in A$	0	**if** $b = a$ **then** u **else** 0
$(u \oplus v)$	$\hat{o}(u) + \hat{o}(u)$	$(u_a \oplus v_a)$
$(u \otimes v)$	$\hat{o}(u) \wedge \hat{o}(v)$	$((u_a \otimes v) \oplus (\hat{o}(u) \otimes v_a))$
$(u^{\bar{*}})$	1	$(u_a \otimes u^{\bar{*}})$

(3.6)

Using induction, we can now show that for each $s \in \mathfrak{Rat}_S(\Sigma)$, $\mathbf{val}(s)_a = \mathbf{val}(\hat{\delta}(s, a))$.

We can gather all the starred expressions in the derivative by

$$T = \bigcup_{a \in A, \tau \in \Sigma} \mathbf{star}(\delta(\tau, a))$$

and extend Σ with the evaluations of these starred expressions, giving a new set $\hat{\Sigma} \subseteq S\langle\langle A\rangle\rangle$, again finite:

$$\hat{\Sigma} := \Sigma \cup \{\mathbf{val}(t) \,|\, t \in T\}$$

Now it can be shown, again using induction, that for any $s \in \mathfrak{Rat}_S(\Sigma)$, if $\mathbf{star}(s) \subseteq T$, then for all $a \in A$, again $\mathbf{star}(\hat{\delta}(s,a)) \subseteq T$.

It now follows that for each $\tau \in \Sigma'$ and $a \in A$, τ_a is S-polynomial in Σ. Because $\sigma \in \Sigma'$, it now follows that σ is constructively S-algebraic. □

3.3.2 Systems of equations and the Greibach normal form

We will now establish a connection between the constructively power series we just defined in terms of behavioural differential equations, and the presentation, common in the theory of weighted automata, of algebraic power series over a semiring S as solutions to certain classes of (*flat*) systems of equations. In the current context, a *system of equations* over an alphabet A will be a pair (X, p), where X is a finite set of nonterminals, and

$$p : X \to S\langle X + A\rangle$$

is a mapping assigning a polynomial over the disjoint union $X + A$ to each $x \in X$. This definition corresponds exactly to S-algebraic systems as presented in e.g. [PS09] and [Fli74]. Such systems can also be regarded as weighted grammars: the mapping p, in this case, can be seen as the set of (weighted) production rules of the grammar.

A system of equations is called *proper* iff, for all $x \in X$ and $y \in Y$

$$[p(x) \Downarrow 1] = 0 \qquad \text{and} \qquad [p(x) \Downarrow y] = 0.$$

If we would regard such systems as weighted grammars, the notion of being proper corresponds to the absence of empty word productions and unit productions in the grammar. Furthermore, such a system is said to be in *Greibach normal form* whenever

$$\mathbf{supp}(p) \subseteq AX^*.$$

We will now give a precise definition of a *solution* to such a system, equivalent to the classical situation, but presented in terms of commuting diagrams: to be precise, a *solution* to a system of equations is a mapping

$$s : X \to S\langle\langle A\rangle\rangle,$$

of variables to power series, such that the diagram

$$\begin{tikzcd} X + A \arrow[r, "{[s,i]}"] \arrow[d, "p+i"'] & S\langle\langle A\rangle\rangle \\ S\langle X + A\rangle \arrow[ru, "{[s,i]^\sharp}"'] & \end{tikzcd}$$

commutes.

This formal presentation captures the intuitive idea that for all $x \in X$, x and $p(x)$ have the same interpretation on $S\langle\langle A\rangle\rangle$.

A solution to a system of equations is called strong *strong* whenever, for all $x \in X$, we have

$$O(s(x)) = 0.$$

It is well-known (see e.g. [PS09]) that proper systems have a unique strong solution. Other than this strong solution, proper systems also may have other solutions. For example, the system $x = x^2$ has (over an arbitrary semiring) solutions $x = 0$ and $x = 1$; but only the first of these solutions is strong. This can be contrasted with polynomial systems of behavioural differential equations, which are guaranteed to always have a unique solution.

In the case where the underlying semiring is the Boolean semiring $\mathbb{B}$, such systems of equations are equivalent to context-free grammars without empty word-productions and without unit-productions. Furthermore, we note that, without the additional condition of properness, these systems of equations do not necessarily have solutions: for example, the (non-proper) system consisting of the single equation $x = x+1$, over the semiring $\mathbb{N}$ does not have any solutions.

We will now use Lemma 2.22 in its full generality to obtain the following result, which is equivalent to [PS09, Theorem 3.2] and [PS09, Theorem 3.15].

Proposition 3.17. *Every proper S-weighted system of equations has exactly one strong solution, which occurs as the final coalgebra mapping of a polynomial system of behavioural differential equations, canonically obtainable from the system, and which thus is constructively S-algebraic.*

Proof. Any solution to a proper S-weighted system over a set of variables $X = \{x_0, \ldots, x_k\}$ will satisfy equations of the form

$$x_i = \sum_{j=0}^{k} x_j q_{ij} + \sum_{a \in A} a r_{ia}$$

where each q_{ij} is rational over X and proper, and each r_{ia} is rational over X.

Under the assumption that such a solution is strong, we can take the derivative to any $a \in A$ to obtain:

$$(x_i)_a = r_{ia} + \sum_{j=0}^{k} (x_j)_a q_{ij}$$

By Lemma 2.22, it now follows that each derivative $(x_i)_a$ is rational over X, so by Proposition 3.16, it now follows that this solution occurs as the unique solution to the corresponding polynomial system of behavioural differential equations. □

The above proposition can be regarded as a construction of the Greibach normal form, as it transforms arbitrary proper systems into polynomial systems of behavioural differential equations, which are in direct correspondence with systems of equations in Greibach normal form.

3.4 Counting derivations in grammars

We will now turn to an application of behavioural differential equations, and bisimulation-based proof techniques (more specifically: bisimulation up to linearity), to a number of combinatorial counting problems. It turns out that a number of familiar sequences, including e.g. the Catalan and Schröder numbers, can easily be described and understood using systems of behavioural differential equations, characterizing these sequences as constructively algebraic streams.

The results in this section can be related to [Rut02], in which a number of counting problems are presented using (both finite and infinite) weighted automata. Compared to the work in that article, we present a more systematic account, giving a uniform technique of obtaining systems of behavioural differential equations directly from a description of the combinatorial structure of a sequence. For example, we can start from the characterization of the Catalan numbers as the number of matching pairs of parentheses of a certain length, and from this characterization directly obtain a system of behavioural differential equations having the Catalan numbers as a solution. Because polynomial systems of behavioural differential equations can be regarded as infinite weighted automata, we can regard this method as a more systematic approach, extending the more ad hoc approach from [Rut02].

Recalling from Section 3.2 that $\Rightarrow^*$ was defined as the transitive closure of $\Rightarrow$, we now observe that

$$v \Rightarrow^* w$$

is true if and only if there is a natural number n, together with a function $f : \{n \in \mathbb{N} \,|\, n \leq n\} \to (X + A)^*$ such that for all $m \in \mathbb{N}$ with $m < n$, $f(m) \Rightarrow f(m+1)$, and moreover $f(0) = v$ and $f(n) = w$.

Given $v, w \in (X + A)^*$, we let $\mathbf{dv}(v, w)$ denote the set of all distinct leftmost derivations $v \Rightarrow^* w$, that is, the set of all such pairs (n, f) witnessing $v \Rightarrow^* w$.

In this section, we will establish that certain power series and streams, representing the degrees of ambiguity of derivations of context-free grammars in Greibach normal form, are again context-free. This will, again, be done using the technique of bisimulation up to linearity. Proposition 3.18 below establishes that, given a context-free grammar in Greibach normal form, presented as a polynomial system of behavioural differential equations (X, o, δ) over the Boolean semiring $\mathbb{B}$, the power series

$$\sum_{w \in A^*} |\mathbf{dv}(v, w)| w$$

is constructively $\mathbb{N}$-algebraic for all words $v \in X^*$. The results in this section were originally proven by Chomsky and Schützenberger in [CS63]; our proofs of these classical results will be bisimulation-based proofs.

Starting from an arbitrary polynomial system of behavioural differential equations

$$(X, o_0, \delta_0)$$

over $\mathbb{B}$, that is, any grammar in Greibach normal form, we can construct another system over $\mathbb{N}$

$$(X, o_1, \delta_1)$$

specified by

$$o_1(x) = e(o_0(x)) \qquad \text{and} \qquad [\delta_1(x, a) \Downarrow v] = e([\delta_0(x, a) \Downarrow v]),$$

where the function $e : \mathbb{B} \to \mathbb{N}$ is defined by $e(0) = 0$ and $e(1) = 1$.

First note we have, for $v \in X^*$, $a \in A$, and $z \in A^*$,

$$|\mathbf{dv}(v, 1)| = e(o_0(v))$$

and

$$|\mathbf{dv}(v, az)| = \sum_{u \in X^*} e([\delta_0(v, a) \Downarrow u])|\mathbf{dv}(u, z)|.$$

Proposition 3.18. *The relation*

$$R = \left\{ \left(v, \sum_{w \in A^*} |\mathbf{dv}(v,w)| w \right) \middle| \, v \in X^* \right\}$$

is a bisimulation up to linearity between the S-linear automata $(\mathbb{N}\langle X\rangle, \hat{o}_1, \hat{\delta}_1)$ and $(\mathbb{N}\langle\!\langle A\rangle\!\rangle, O, \Delta)$.

Proof. We have

$$o_1(v) = e(o_0(v)) = |\mathbf{dv}(v,1)| = O\left(\sum_{w \in A^*} |\mathbf{dv}(v,w)| w \right)$$

and, if $\left(v, \sum_{w \in A^*} |\mathbf{dv}(v,w)|w\right) \in R$, then

$$\begin{array}{rcl}
v_a & = & \\
\displaystyle\sum_{u \in X^*} [v_a \Downarrow u] u & = & \\
\displaystyle\sum_{u \in X^*} e([\delta_0(v,a) \Downarrow u]) u & \Sigma R & \displaystyle\sum_{u \in X^*} e([\delta_0(v,a) \Downarrow u]) \sum_{w \in A^*} |\mathbf{dv}(u,w)| w \\
& = & \displaystyle\sum_{z \in A^*} \sum_{u \in X^*} e([\delta_0(v,a) \Downarrow u]) |\mathbf{dv}(u,z)| z \\
& = & \displaystyle\sum_{z \in A^*} |\mathbf{dv}(v,az)| z \\
& = & \displaystyle\sum_{b \in A} \sum_{z \in A^*} |\mathbf{dv}(v,bz)| (bz)_a + |\mathbf{dv}(v,1)| 1_a \\
& = & \displaystyle\sum_{w \in A^*} |\mathbf{dv}(v,w)| w_a \\
& = & \left(\displaystyle\sum_{w \in A^*} |\mathbf{dv}(v,w)| w \right)_a .
\end{array}$$

□

We can now also, given a system of behavioural differential equations (X, o, δ) over an alphabet A and some $v \in X^*$, create a context-free stream σ, such that for every number $n \in \mathbb{N}$, $\sigma(n)$ is equal to

$$\sum_{w \in A^*, |w| = n} = |\mathbf{dv}(v,w)|.$$

In order to do this, let us construct another polynomial system

$$(X, o_2, \delta_2)$$

defined by

$$o_2(x) = o_1(x) \qquad \text{and} \qquad x' = \sum_{a \in A} \delta_1(x, a).$$

The following proposition establishes that this new system indeed has the intended behaviour:

Proposition 3.19. *The relation*

$$R = \left\{ \left(v, \sum_{w \in A^*} |\mathbf{dv}(v, w)| \mathcal{X}^{|w|} \right) \middle| \, v \in X^* \right\}$$

is a bisimulation up to linearity between the extension $(\mathbb{N}\langle X \rangle, o_2, \delta_2)$ *of the system just given, and* $(\mathbb{N}^{\mathbb{N}}, \mathbf{head}, \mathbf{tail})$.

Proof. We have

$$o_2(v) = o_1(v) = |\mathbf{dv}(v, 1)| = O\left(\sum_{w \in A^*} |\mathbf{dv}(v, w)| \mathcal{X}^{|w|} \right)$$

and, if

$$\left(v, \sum_{w \in A^*} |\mathbf{dv}(v, w)| \mathcal{X}^{|w|} \right) \in R,$$

then also

$$\begin{aligned}
v' &= \\
\sum_{u \in X^*} [v' \Downarrow u] u &= \\
\sum_{u \in X^*} \sum_{a \in A} [v_a \Downarrow u] u & \;\Sigma R\; \sum_{u \in X^*} \sum_{a \in A} [v_a \Downarrow u] \sum_{w \in A^*} |\mathbf{dv}(u, w)| \mathcal{X}^{|w|} \\
&= \sum_{a \in A} \sum_{z \in A^*} \sum_{u \in X^*} [v_a \Downarrow u] |\mathbf{dv}(u, z)| \mathcal{X}^{|w|} \\
&= \sum_{a \in A} \sum_{z \in A^*} |\mathbf{dv}(v, az)| \mathcal{X}^{|z|}
\end{aligned}$$

$$\begin{aligned} &= \sum_{w \in A^*} |\mathbf{dv}(v,w)| \mathcal{X}^{|w|'} \\ &= \left(\sum_{w \in A^*} |\mathbf{dv}(v,w)| \mathcal{X}^{|w|} \right)'. \end{aligned} \qquad \Box$$

This construction now enables us to derive specifications of some well-known number sequences as context-free streams.

Example 3.20. We now return to the Catalan numbers, which we introduced in Example 3.10. One of the ways of characterizing the nth Catalan number, is as to the number of ways to combine n pairs of matching brackets. An unambiguous context-free grammar in Greibach normal form representing matching pairs of brackets is

$$x \to axbx \mid 1$$

which corresponds to the system of equations

$$\begin{array}{lll} o(x) = 1 & x_a = xyx & x_b = 0 \\ o(y) = 0 & y_a = 0 & y_b = 1. \end{array}$$

Using the transformation on which Proposition 3.19 was based, we now obtain another system of equations

$$\begin{array}{ll} o(x) = 1 & x' = xyx \\ o(y) = 0 & y' = 1. \end{array}$$

yielding the stream σ such that for all $n \in \mathbb{N}$, $\sigma(2n)$ is equal to the nth Catalan number, and $\sigma(2n+1) = 0$.

Now, because multiplication of streams over $\mathbb{N}$ is commutative, observe that

$$x'' = (xyx)' = (yx^2)' = y'x^2 + o(y)(x^2)' = y'x^2 = x^2,$$

giving us a new system, over a single variable z, defined by

$$o(z) = 1 \qquad z' = z^2.$$

Because it clearly holds that $[\![z]\!](n) = [\![x]\!](2n)$, it follows immediately that the final homomorphism $[\![-]\!]$ maps z onto the stream of Catalan numbers: thus we have established that the Catalan numbers are a constructively $\mathbb{N}$-algebraic stream.

Example 3.21. Another, closely related, example arises from the following problem: given a $n \times n$ grid, how many different ways are there to go from $(0,0)$ to (n,n) by making steps of the types $(0,1)$, $(1,0)$ and $(1,1)$ that stay below the diagonal? The sequence mapping each $n \in \mathbb{N}$ to the number of such paths from $(0,0)$ to (n,n) is called the sequence of (large) Schröder numbers:

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, \ldots \quad \text{(A006318)}$$

It is easy to see that the nth Schröder number corresponds to the number of derivations of words of length $2n$ given by the following system of behavioural differential equations:

$$\begin{array}{llll} o(x) = 1 & x_a = xyx & x_b = 0 & x_c = zx \\ o(y) = 0 & y_a = 0 & y_b = 1 & y_c = 0 \\ o(z) = 0 & z_a = 0 & z_b = 0 & z_c = 1. \end{array}$$

Here a represents a step of the type $(0,1)$, b represents a step of the type $(1,0)$, and cc (cs can only occur in pairs in the language $[\![x]\!]$) represents steps of the type $(1,1)$.

Using the transformation from Proposition 3.19, we now obtain a new system of equations

$$\begin{array}{ll} o(\bar{x}) = 1 & \bar{x}' = \bar{x}\bar{y}\bar{x} + \bar{z}\bar{x} \\ o(\bar{y}) = 0 & \bar{y}' = 1 \\ o(\bar{z}) = 0 & \bar{z}' = 1 \end{array}$$

and again, by commutativity of multiplication, we now get

$$\bar{x}'' = (\bar{x}\bar{y}\bar{x} + \bar{z}\bar{x})' = (\bar{y}\bar{x}^2)' + (\bar{z}\bar{x})' = \bar{y}'\bar{x}^2 + o(\bar{y})(\bar{x}^2)' + \bar{z}'\bar{x} + o(\bar{z})\bar{x}' = \bar{x}^2 + \bar{x}$$

yielding a new system over a single variable u:

$$o(u) = 1 \qquad u' = u^2 + u$$

which is mapped by $[\![-]\!]$ onto the Schröder numbers.

4

Additional systems and operators

The present chapter further extends the work from Chapters 2 and 3 into a coalgebraic treatment of *pushdown automata* and *weighted pushdown systems*, presented again over arbitrary commutative semirings. Additionally, we introduce the Hadamard product (which can, on the level of power series, be regarded as a *pointwise* product, and on the level of formal languages as the *intersection* operator), and present coalgebraic versions of some preservation properties of the Hadamard product, making use of the earlier defined S-weighted automata as well as S-weighted pushdown systems.

In Section 4.1, we will give a first presentation of pushdown automata and weighted pushdown systems, together with a coalgebraic semantics that corresponds to the condition of *empty stack acceptance.* Next, we will refine our notion of empty stack acceptance into the condition that a configuration is accepting if and only if the stack is empty *and* the pushdown machine is in an accepting state. Our coalgebraic presentation of pushdown automata differs from, and can be contrasted with the coalgebraic presentation from [SBBR13], which is based on $S \times TX^A$-coalgebras, where T is the *nondeterministic state monad.*

Next, in Section 4.2, we introduce the Hadamard product. Using bisimulation techniques, we then give presentations of the main results from [Sch61a], stating that the Hadamard product of two recognizable power series is again recognizable, and that the Hadamard product of a recognizable and a algebraic power series is again algebraic. In contrast with the original presentation from [Sch61a], our presentation of the latter result is directly based on the connection with pushdown systems, and can easily be seen to follow from the results from 4.1.

In the case of formal languages, the general results from Section 4.2 instantiate to the fact that the intersection of two regular languages is again regular, and that the intersection of a regular language with a context-free language is again context-free.

The final section of this chapter introduces the **zip** and **unzip** operators, which respectively *zip* together a finite list of streams into a new stream, and *unzip* this stream back into a finite list of streams. We furthermore prove that, for arbitrary semirings S, $\mathbf{zip}_k$ preserves S-algebraicity. This result is a generalization of a result that appeared in [NR10]. In the following chapter, we will explore the connections between the **zip** and **unzip** operators and the *k-automatic* and *k-regular* sequences.

4.1 Pushdown automata

Given a commutative semiring S, an *S-weighted pushdown system (with empty stack acceptance)* is a triple

$$(Q, X, \delta)$$

where:

1. Q is a finite set, which can be regarded as a set of *states* or *variables*;
2. X is a finite set, which can be regarded as a set of *stack symbols*, *nonterminals*, or again, *variables*;
3. δ is a function from $Q \times X$ to $\mathrm{Lin}_S(Q \times X^*)^A$ called the *transition function.*

When S is the Boolean semiring $\mathbb{B}$, these systems correspond to the usual presentation of *real time* (i.e. without empty word transitions) pushdown automata , with the exception of (again) the absence of an initial state.

We will now extend any such system into an (infinite) S-weighted automaton (or, in the Boolean case, a nondeterministic automaton), which captures the behaviour of the usually defined pushdown automata.

For this extension, we need to define the operator

$$\cdot^\flat : \mathrm{Lin}_S(Q \times X^*) \times X^* \to \mathrm{Lin}_S(Q \times X^*)$$

by

$$[s \cdot^\flat w \Downarrow (q, u)] = (\textbf{if } \exists u . u = zw \textbf{ then } [s \Downarrow (q, z)] \textbf{ else } 0)$$

for all $s \in \mathrm{Lin}_S(Q \times X^*)$, all q in Q, and all $u, w \in X^*$. Given any $q \in Q$ and $v \in X^*$, it directly follows from the definition that

$$(q, v) \cdot^\flat w = (q, vw),$$

or in other words $(q, v) \cdot^\flat w$ is obtained by concatenating w to v. Intuitively, we can now regard $s \cdot^\flat w$ as the unique linear extension of such concatenations.

We now uniquely extend any S-weighted pushdown system (X, Q, δ) to a S-weighted automaton $(Q \times X^*, \bar{o}, \bar{\delta})$ specified by the following behavioural differential equations:

$$\begin{array}{rclcrcl} \bar{o}(q,1) & = & 1 & \qquad & (q,1)_a & = & 0 \\ \bar{o}(q,xw) & = & 0 & & (q,xw)_a & = & (q,x)_a \cdot^\flat w \end{array}$$

The specification of $\bar{o}$ can be seen as a description of the condition of empty stack acceptance.

In the Boolean case, it is easy to see that this captures the expected behaviour of a pushdown automaton. The left component can be seen as representing the state of the machine, and the right component as the stack content. Sums represent nondeterminism.

We will now construct a polynomial system out of this pushdown system, under the assumptions that $o(x) = 0$ for $x \in X$ and $o(q) = 1$ for $q \in Q$, which correspond to empty stack acceptance. The construction which now follows corresponds to (a generalization of) the construction of context-free grammars from pushdown automata presented in [HMU06], presented in a coalgebraic fashion.

The state space of the pushdown system will be

$$Q \times X \times Q$$

triples of elements from Q, X, and again Q.

Define the partial function

$$\mathbf{chain} : X^* \to Q^* \to (Q \times X \times Q)^*$$

so that $\mathbf{chain}(v, w)$ is defined if and only if $|w| = |v| + 1$ and defined by

$$\begin{array}{rcl} \mathbf{chain}(1, q) & = & 1 \\ \mathbf{chain}(xv, qrw) & = & (q, x, r)\mathbf{chain}(v, rw) \end{array}$$

for $x, \in X, v \in X^*, q, r \in Q$, and $w \in Q^*$ whenever this requirement holds.

The intuition of this function can be explained by the equation

$$\mathbf{chain}(x_1x_2x_3, q_0q_1q_2q_3) = (q_0, x_1, q_1)(q_1, x_2, q_2)(q_2, x_3, q_3)$$

and using induction it is easily proven that for all $q \in Q, v, w \in X^*, u, z \in Q^*$ with $|v| = |u|$ and $|w| = |z|$, we have

$$\mathbf{chain}(vw, uqz) = \mathbf{chain}(v, uq) \cdot \mathbf{chain}(w, qz).$$

We now specify the polynomial system $(Q \times X \times Q, o', \delta')$ using the behavioural differential equations

$$o'(q,x,r) = 0$$

and

$$(q,x,r)_a = \sum_{s \in Q, v \in X^+} [(q,x)_a \Downarrow (s,v)] \sum_{u \in Q^{|v|-1}} \mathbf{chain}(v, sur) + [(q,x)_a \Downarrow (r,1)].$$

Proposition 4.1. *The relation*

$$R = \left\{ \left((q,w), \sum_{v \in Q^{|w|}} \mathbf{chain}(w, qv) \right) \middle| \; q \in Q, w \in X^* \right\}$$

is a bisimulation up to linearity between the S-linear automata

$$(\mathrm{Lin}_S(Q \times X^*), \hat{o}, \hat{\delta})$$

and

$$(S\langle Q \times X \times Q \rangle, \hat{o}', \hat{\delta}').$$

Proof. Observe that if $r \in R$, then r has one of the following forms:

$$\begin{array}{rcl} (q,1) & R & 1 \\ (q,xw) & R & \displaystyle\sum_{v \in Q^{|xw|}} \mathbf{chain}(xw, qv) \end{array}$$

In order to show that R is a bisimulation up to linearity, we proceed by showing, using a simple case distinction, that for all $(r_0, r_1) \in R$, $\hat{o}(r_0) = \hat{o}'(r_1)$ and $(r_0)_a \; \Sigma R \; (r_1)_a$.

- **Case** $(q,1) \; R \; 1$. We have $\hat{o}(q,1) = 1 = o(1)$ and

$$(q,1)_a = 0 = 1_a.$$

- **Case** $(q,xw) \; R \displaystyle\sum_{v \in Q^{|xw|}} \mathbf{chain}(xw, qv)$.

We have

$$\hat{o}(q, xw) = 0 = \hat{o}'\left(\sum_{v \in Q^{|xw|}} \mathbf{chain}(xw, qv)\right)$$

and

$$\begin{aligned}
& (q, xw)_a \\
= \;& (q, x)_a \cdot^\flat w \\
= \;& \sum_{r \in Q} [(q, x)_a \Downarrow (r, 1)](r, 1) \cdot^\flat w + \sum_{s \in Q, v \in X^+} [(q, x)_a \Downarrow (s, v)](s, v) \cdot^\flat w \\
= \;& \sum_{r \in Q} [(q, x)_a \Downarrow (r, 1)](r, w) + \sum_{s \in Q, v \in X^+} [(q, x)_a \Downarrow (s, v)](s, vw) \\
\Sigma R \;& \sum_{r \in Q} [(q, x)_a \Downarrow (r, 1)] \sum_{z \in Q^{|w|}} \mathbf{chain}(w, rz) \\
& \quad + \sum_{s \in Q, v \in X^+} [(q, x)_a \Downarrow (s, v)] \sum_{t \in Q^{|vw|}} \mathbf{chain}(vw, st) \\
= \;& \sum_{r \in Q} [(q, x)_a \Downarrow (r, 1)] \sum_{z \in Q^{|w|}} \mathbf{chain}(w, rz) \\
& \quad + \sum_{s \in Q, v \in X^+} [(q, x)_a \Downarrow (s, v)] \sum_{u \in Q^{|v|-1}, r \in Q, z \in Q^{|w|}} \mathbf{chain}(vw, surz) \\
= \;& \sum_{r \in Q} \left(\sum_{s \in Q, v \in X^+} [(q, x)_a \Downarrow (s, v)] \sum_{u \in Q^{|v|-1}} \mathbf{chain}(v, sur) \right. \\
& \quad \left. + [(q, x)_a \Downarrow (r, 1)] \right) \sum_{z \in Q^{|w|}} \mathbf{chain}(w, rz) \\
= \;& \sum_{r \in Q, z \in Q^{|w|}} (q, x, r)_a \mathbf{chain}(w, rz) \\
= \;& \sum_{r \in Q, z \in Q^{|w|}} \mathbf{chain}(xw, qrz)_a \\
= \;& \left(\sum_{t \in Q^{|xw|}} \mathbf{chain}(xw, qt) \right)_a
\end{aligned}$$

□

We thus have shown that any power series accepted by a pushdown automaton (realtime, with empty stack acceptance) is constructively algebraic.

4.1.1 Refining the acceptance condition

We now will introduce an extension of the acceptance conditions, by adding an output function o_0 assigning an output value to states. This extension will be needed for the main result in the next section, on the Hadamard product.

Given a commutative semiring S, a S*-weighted pushdown system (with refined empty stack acceptance)* is a quadruple

$$(Q, X, o_0, \delta)$$

where:

1. Q is a finite set, representing states;
2. X is a finite set, stack symbols;
3. $o_0 : Q \to S$ is an output function on states in Q;
4. δ is, as before a transition function from $Q \times X$ to $\mathrm{Lin}_S(Q \times X^*)^A$.

We again uniquely extend any such system to a S-weighted automaton using the behavioural differential equations:

$$\begin{array}{rclcrcl} \bar{o}(q,1) & = & o_0(q) & \qquad & (q,1)_a & = & 0 \\ \bar{o}(q,xw) & = & 0 & & (q,xw)_a & = & (q,x)_a \cdot^\flat w \end{array}$$

Proposition 4.2. *The relation*

$$\begin{aligned} R = &\{(q,1), o_0(q)\} \\ &\cup \left\{ \left((q,xw), \sum_{p\in Q} o_0(p) \sum_{v \in Q^{|w|}} \mathbf{chain}(xw, qvp) \right) \middle| \; q \in Q, x, w \in X^* \right\} \end{aligned}$$

is a bisimulation up to linearity between the S*-linear automata*

$$(\mathrm{Lin}_S(Q \times X^*), \hat{o}, \hat{\delta})$$

and

$$(S\langle Q \times X \times Q\rangle, \hat{o}', \hat{\delta}').$$

Proof. Observe that if $r \in R$, then r has one of the following forms:

$$\begin{array}{rcl} (q,1) & R & o_0(q) \\ (q,x) & R & \sum_{p\in Q} o_0(p)(q,x,p) \\ (q,xyw) & R & \sum_{p\in Q} o_0(p) \sum_{v\in Q^{|yw|}} \mathbf{chain}(xyw, qvp) \end{array}$$

We again proceed by case distinction:

- **Case** $(q,1)\ R\ o_0(q)$. We have $\hat{o}(q,1) = \bar{o}(q,1) = o_0(q)$ and

$$(q,1)_a = 0 \quad \Sigma R \quad 0 = (o_0(q))_a.$$

- **Case** $(q,x)\ R \sum_{p\in Q} o_0(p)(q,x,p)$.

  We have

$$\hat{o}(q,x) = 0 = \hat{o}'\left(\sum_{p\in Q} o_0(p)(q,x,p)\right)$$

  and

$$\begin{array}{rl} & (q,x)_a \\ = & \sum_{r\in Q}[(q,x)_a \Downarrow (r,1)](r,1) + \sum_{s\in Q, v\in X^+} [(q,x)_a \Downarrow (s,v)](s,v) \\ \Sigma R & \sum_{r\in Q}[(q,x)_a \Downarrow (r,1)]o_0(r) \\ & \quad + \sum_{s\in Q, v\in X^+} [(q,x)_a \Downarrow (s,v)] \sum_{p\in Q} o_0(p) \sum_{u\in Q^{|v|-1}} \mathbf{chain}(v, sup) \\ = & \sum_{p\in Q} o_0(p)[(q,x)_a \Downarrow (p,1)] \\ & \quad + \sum_{p\in Q} o_0(p) \sum_{s\in Q, v\in X^+} [(q,x)_a \Downarrow (s,v)] \sum_{u\in Q^{|v|-1}} \mathbf{chain}(v, sup) \\ = & \sum_{p\in Q} o_0(p)(q,x,p)_a \end{array}$$

- **Case** $(q, xyw)\ R \sum_{p\in Q} o_0(p) \sum_{v\in Q^{|yw|}} \mathbf{chain}(xyw, qvp)$.

  We have

$$\hat{o}(q, xyw) = 0 = \hat{o}'\left(\sum_{p\in Q} o_0(p) \sum_{v\in Q^{|yw|}} \mathbf{chain}(xyw, qvp)\right)$$

  and

$$\begin{aligned}
& (q, xyw)_a \\
= \ & (q, x)_a \cdot^\flat yw \\
= \ & \sum_{r\in Q}[(q, x)_a \Downarrow (r, 1)](r, 1) \cdot^\flat yw + \sum_{s\in Q, v\in X^+} [(q, x)_a \Downarrow (s, v)](s, v) \cdot^\flat yw \\
= \ & \sum_{r\in Q}[(q, x)_a \Downarrow (r, 1)](r, yw) + \sum_{s\in Q, v\in X^+} [(q, x)_a \Downarrow (s, v)](s, vyw) \\
\Sigma R \ & \sum_{r\in Q}[(q, x)_a \Downarrow (r, 1)] \sum_{p\in Q} o_0(p) \sum_{z\in Q^{|w|}} \mathbf{chain}(yw, rzp) \\
& \quad + \sum_{s\in Q, v\in X^+} [(q, x)_a \Downarrow (s, v)] \sum_{p\in Q} o_0(p) \sum_{t\in Q^{|vw|}} \mathbf{chain}(vyw, stp) \\
= \ & \sum_{r\in Q}\left(\sum_{s\in Q, v\in X^+} [(q, x)_a \Downarrow (s, v)] \sum_{u\in Q^{|v|-1}} \mathbf{chain}(v, sur)\right. \\
& \quad \left. + [(q, x)_a \Downarrow (r, 1)]\right) \sum_{p\in Q} o_0(p) \sum_{z\in Q^{|w|}} \mathbf{chain}(yw, rzp) \\
= \ & \sum_{p\in Q} o_0(p) \sum_{r\in Q, z\in Q^{|w|}} (q, x, r)_a \mathbf{chain}(yw, rzp) \\
= \ & \sum_{p\in Q} o_0(p) \sum_{r\in Q, z\in Q^{|w|}} \mathbf{chain}(xyw, qrzp)_a \\
= \ & \sum_{p\in Q} o_0(p) \left(\sum_{v\in Q^{|w+1|}} \mathbf{chain}(xyw, qvp)\right)_a
\end{aligned}$$

□

4.2 The Hadamard product

Let $\odot$ denote the Hadamard product on power series, with $\sigma \odot \tau$ defined on power series $\sigma, \tau \in S\langle\langle A\rangle\rangle$ by

$$[\sigma \odot \tau \Downarrow w] = [\sigma \Downarrow w][\tau \Downarrow w] \qquad \text{for all } w \in A^*$$

and define $\mathbf{1}$ by

$$[\mathbf{1} \Downarrow w] = 1 \qquad \text{for all } w \in A^*.$$

It easy to see that

$$(S\langle\langle A\rangle\rangle, +, \odot, 0, \mathbf{1})$$

is a semiring again. Another way to characterize $\odot$ is as the unique operation on $S\langle\langle A\rangle\rangle$ satisfying the behavioural differential equations

$$O(\sigma \odot \tau) = O(\sigma)O(\tau) \qquad \text{and} \qquad (\sigma \odot \tau)_a = \sigma_a \odot \tau_a.$$

Relying purely on the results from Chapter 2, it is straightforward to show that the Hadamard product of two recognizable power series is again recognizable, relying on the usual product construction:

Proposition 4.3. *If $\sigma \in S\langle\langle A\rangle\rangle$ and $\tau \in S\langle\langle A\rangle\rangle$ are both S-recognizable, then $\sigma \odot \tau$ is S-recognizable.*

Proof. If σ and τ are S-recognizable, there are S-weighted automata (P, o_P, δ_P) and (Q, o_Q, δ_Q), and elements $p, q \in P, Q$ such that $[\![p]\!]_P = \sigma$ and $[\![q]\!]_Q = \tau$.

Now define a S-weighted automaton $(P \times Q, o, \delta)$ as follows:

$$\begin{aligned} o(p,q) &= o_P(p)o_Q(q) \\ (p,q)_a &= \sum_{r,s \in Q} [p_a \Downarrow r][q_a \Downarrow s](r,s) \end{aligned}$$

Now consider the relation

$$R = \{((p,q), [\![p]\!]_P \odot [\![q]\!]_Q)\}$$

and observe that R is a bisimulation up to linearity, as

$$\hat{o}(p,q) = o_P(p)o_Q(q) = O([\![p]\!]_P \odot [\![q]\!]_Q),$$

and

$$(p,q)_a \quad =$$

$$\begin{aligned}\sum_{r,s\in Q}[p_a \Downarrow r][q_a \Downarrow s](r,s) = \quad \Sigma R & \sum_{r,s\in Q}[p_a \Downarrow r][q_a \Downarrow s](\llbracket r\rrbracket_P \odot \llbracket s\rrbracket_Q) \\ = \quad & \llbracket\sum_{r\in Q}[p_a \Downarrow r]r\rrbracket_P \odot \llbracket\sum_{s\in Q}[q_a \Downarrow r]s\rrbracket_Q \\ = \quad & \llbracket p_a\rrbracket_P \odot \llbracket q_a\rrbracket_Q \\ = \quad & (\llbracket p\rrbracket_P \odot \llbracket q\rrbracket_Q)_a\end{aligned}$$

so it follows that $\llbracket(p,q)\rrbracket = \llbracket p\rrbracket_P \odot \llbracket q\rrbracket_Q$ and that $\llbracket(p,q)\rrbracket$ is again S-recognizable. □

We now turn to the main result, which is a new presentation of a result originally proven in [Sch61a], stating that the Hadamard product of a recognizable and a constructively algebraic power series is again constructively algebraic.

Proposition 4.4. *If $\sigma \in S\langle\langle A\rangle\rangle$ is S-recognizable, and $\tau \in S\langle\langle A\rangle\rangle$ is constructively S-algebraic, then $\sigma \odot \tau$ is constructively S-algebraic.*

Proof. If σ is S-recognizable and τ S-algebraic, there is a S-linear system (Q, o_Q, δ_Q) and a weighted system (X, o_X, δ_X) with $o(y) = 0$ for all $x \in X$, and some $q \in Q$ and $x \in X$ such that $\llbracket q\rrbracket_Q = \sigma$ and $\llbracket x\rrbracket_X = \hat{\tau}$ for some τ with $\tau = \hat{\tau} + O(\tau)$.

Define a S-weighted pushdown system (Q, X, o_1, δ) with refined empty stack acceptance by the following differential equations:

$$\begin{aligned} o_1(q) &= o_Q(q) \\ (q,x)_a &= \sum_{r\in Q, v\in X^*}[q_a \Downarrow r][x_a \Downarrow v](r,v)\end{aligned}$$

Now consider the relation

$$R = \{((q,u), \llbracket q\rrbracket \odot \llbracket u\rrbracket)\}$$

and observe that R is a bisimulation up to linearity, as

$$\hat{o}(q,1) = o_1(q) = O(\llbracket q\rrbracket \odot \llbracket 1\rrbracket),$$

$$\hat{o}(q,xt) = 0 = \hat{o}(q)\hat{o}(xt) = O(\llbracket q\rrbracket)O(\llbracket xt\rrbracket) = O(\llbracket q\rrbracket \odot \llbracket xt\rrbracket),$$

$$(q,1)_a = 0 \; \Sigma R \; 0 = \llbracket q\rrbracket_1 \odot 0 = \llbracket q\rrbracket_a \odot \llbracket 1\rrbracket_a = (\llbracket q\rrbracket \odot \llbracket 1\rrbracket)_a,$$

and

$$(q,xu)_a \quad =$$

$$\begin{aligned}
(q,x)_a \cdot^{\flat} u &= \\
\sum_{r\in Q, v\in X^*} [q_a \Downarrow r][t_a \Downarrow u](r, vu) \quad \Sigma R & \sum_{r\in Q, v\in X^*} [q_a \Downarrow r][x_a \Downarrow u](\llbracket r\rrbracket \odot \llbracket vu \rrbracket) \\
&= \llbracket \sum_{r\in Q} [q_a \Downarrow r] r \rrbracket \odot \llbracket \sum_{u\in X^*} [x_a \Downarrow v] vu \rrbracket \\
&= \llbracket q_a \rrbracket \odot \llbracket x_a u \rrbracket \\
&= (\llbracket q \rrbracket \odot \llbracket xu \rrbracket)_a
\end{aligned}$$

In the extension of the corresponding polynomial system, it now follows that

$$\llbracket \sum_{p\in Q} o_0(p)(q,x,p) \rrbracket = \llbracket q \rrbracket \odot \llbracket x \rrbracket = \sigma \odot \hat{\tau}$$

so $\sigma \odot \hat{\tau}$ is constructively S-algebraic, and it follows that $\sigma \odot \tau$ is constructively S-algebraic too. □

However, in general, the Hadamard product of two constructively algebraic power series is *not* constructively algebraic—a simple counterexample in the setting of formal language can be given by the languages

$$\{a^m b^m c^n \mid m,n \in \mathbb{N}\} \qquad \text{and} \qquad \{a^m b^n c^n \mid m,n \in \mathbb{N}\}$$

which are both context-free. The intersection of these two languages is the language

$$\{a^n b^n c^n \mid n \in \mathbb{N}\}$$

which is not context-free.

4.3 Zipping and unzipping

Given two streams $\sigma, \tau \in S^{\mathbb{N}}$ over any set S, we let the stream $\mathbf{zip}(\sigma,\tau)$ be defined using the equations

$$\mathbf{zip}(\sigma,\tau)(2n) = \sigma(n) \qquad \text{and} \qquad \mathbf{zip}(\sigma,\tau)(2n+1) = \tau(n)$$

for all $n \in \mathbb{N}$. Intuitively, we can see $\mathbf{zip}(\sigma,\tau)$ as a stream that alternately takes an element from σ and an element from τ. For example, if $\sigma = (0,0,0,\ldots)$ is the constant stream of zeros, and $\tau = (1,2,3,\ldots)$ is the stream of natural numbers, we get

$$\mathbf{zip}(\sigma,\tau) = (0,1,0,2,0,3,\ldots).$$

Equivalently, we can also define **zip** coinductively with the following system of behavioural differential equations:

$$O(\mathbf{zip}(\sigma,\tau)) = O(\sigma) \qquad \text{and} \qquad (\mathbf{zip}(\sigma,\tau))' = \mathbf{zip}(\tau,\sigma')$$

This definition generalizes as follows: given a finite sequence of streams $\sigma_1,\ldots,\sigma_k \in S^{\mathbb{N}}$, we let the stream $\mathbf{zip}_k(\sigma_1,\ldots,\sigma_k)$ be defined using the equation

$$\mathbf{zip}_k(\sigma_1,\ldots,\sigma_k)(nk+m) = \sigma_{m+1}(n)$$

for all $n \in \mathbb{N}$ and $m \in \mathbb{N}$ with $m < k$. The easily derivable and equivalent coinductive definition now becomes

$$O(\mathbf{zip}_k(\sigma_1,\ldots,\sigma_k) = O(\sigma_1) \quad \text{and} \quad \mathbf{zip}_k(\sigma_1,\ldots,\sigma_k)' = \mathbf{zip}_k(\sigma_2,\ldots,\sigma_k,\sigma_1). \tag{4.1}$$

Similarly, we can define an 'integration rule' for prefixing an element $s \in S$ to a stream specified using the $\mathbf{zip}_k$ operator:

$$s :: \mathbf{zip}_k(\sigma_1,\ldots,\sigma_k) = \mathbf{zip}_k(s :: \sigma_k,\sigma_1,\ldots,\sigma_{k-1}) \tag{4.2}$$

Conversely to the **zip** operator, given a stream σ, we can define streams $\mathbf{even}(\sigma)$ and $\mathbf{odd}(\sigma)$ consisting of the even and odd elements of σ, respectively, as follows:

$$(\mathbf{even}(\sigma))(n) = \sigma(2n) \qquad \text{and} \qquad (\mathbf{odd}(\sigma))(n) = \sigma(2n+1)$$

We can relate **zip**, **even**, and **odd** by the equation

$$\sigma = \mathbf{zip}(\mathbf{even}(\sigma),\mathbf{odd}(\sigma))$$

which is easily verified to hold for all streams σ.

Again, the **even** and **odd** operations can be generalized to the operation $\mathbf{unzip}_{i,k}$, defined for all $i,k \in \mathbb{N}$ with $i < k$ by

$$\mathbf{unzip}_{i,k}(\sigma)(n) = \sigma(kn+i)$$

yielding $\mathbf{even} = \mathbf{unzip}_{0,2}$ and $\mathbf{odd} = \mathbf{unzip}_{1,2}$.

The $\mathbf{zip}_k$ and $\mathbf{unzip}_{i,k}$ functions are again related by the equation

$$\sigma = \mathbf{zip}_k(\mathbf{unzip}_{0,k}(\sigma),\ldots,\mathbf{unzip}_{k-1,k}(\sigma)). \tag{4.3}$$

4.3.1 Algebraicity is preserved by zip

In this section, we will establish a result which states that if streams

$$\sigma_1, \ldots, \sigma_k$$

are constructively S-algebraic, then the stream

$$\mathbf{zip}_k(\sigma_1, \ldots, \sigma_k)$$

is again S-algebraic.

This is a result which holds for all streams that are constructively S-algebraic over any semiring S. In order to prove this result, the following lemma is useful:

Lemma 4.5. *Given any semiring S and streams $\sigma, \tau \in S^{\mathbb{N}}$, we have*

$$\mathbf{zip}(\sigma, \tau) = \mathbf{zip}(\sigma, 0) + \mathcal{X}\mathbf{zip}(\tau, 0),$$

and more generally, for any $k \geq 2$, given streams $\sigma_0, \ldots, \sigma_k \in S^{\mathbb{N}}$, we have

$$\mathbf{zip}_{k+1}(\sigma_0, \ldots, \sigma_k) = \sum_{i=0}^{k} \mathcal{X}^i \mathbf{zip}_{k+1}(\sigma_i, 0, \ldots, 0).$$

Proof. Trivial. □

Proposition 4.6. *Given a commutative semiring S and any $k \geq 2$, if $\sigma \in S^{\mathbb{N}}$ is constructively S-algebraic, then*

$$\mathbf{zip}_k(\sigma, 0, \ldots, 0)$$

is constructively S-algebraic.

Proof. If σ is constructively S-algebraic, there must exist a finite system of polynomial behavioural differential equations (X, o_X, δ_X) in S and a $x \in X$, such that $[\![x]\!] = \sigma$.

Now construct a new system of polynomial b.d.e.'s (Y, o_Y, δ_Y) with

$$Y = \{\bar{x} \mid x \in X\} \cup \{\mathcal{X}\}.$$

Here $f : X \to Y$ is an operation taking each $x \in X$ to the corresponding notational variant $\bar{x} \in Y$, and we naturally extend f to mappings $\bar{f} : X^* \to Y^*$ on words and $\hat{f} : s \in S\langle X \rangle \to S\langle Y \rangle$ on polynomials, built up from the notational variants.

Now define o_Y and δ_Y by

$$o_Y(\bar{x}) = o_X(x) \qquad \text{and} \qquad \bar{x}' = \mathcal{X}^{k-1}\hat{f}(x').$$

We now define the relation $R \subseteq S\langle Y\rangle \times S^{\mathbb{N}}$

$$R = \{(\mathcal{X}^n f(v), \mathcal{X}^n \mathbf{zip}_k([\![v]\!], 0, \ldots, 0)) \mid n \in \mathbb{N}, v \in X^*\}$$

and note that, from the definition of R it directly follows that, whenever $s\ \Sigma R\ t$, then also $\mathcal{X}^n s\ \Sigma R\ \mathcal{X}^n t$.

We will now show that R is a bisimulation up to S-linear combinations. This is done by a case distinction on n:

1. If $n > 0$, observe that

$$o_X(\mathcal{X}^n f(v)) = 0 = o_Y(\mathcal{X}^n \mathbf{zip}_k([\![v]\!], 0, \ldots, 0))$$

and that

$$\begin{array}{rcl} (\mathcal{X}^n f(v))' & = & \\ \mathcal{X}^{n-1} f(v) & \Sigma R & \mathcal{X}^{n-1}\mathbf{zip}_k([\![v]\!], 0, \ldots, 0) \\ & = & (\mathcal{X}^n \mathbf{zip}_k([\![v]\!], 0, \ldots, 0))'. \end{array}$$

2. If $n = 0$, proceed by induction on the length of v. If $|v| = 0$, then $v = 1$, and

$$f(v)' = 1' = 0 \quad \Sigma R \quad 0 = \mathbf{zip}_k([\![1]\!], 0, \ldots, 0)' = \mathbf{zip}_k([\![v]\!], 0, \ldots, 0)'$$

and if $v = xw$ for some $x \in X$ and $w \in X^*$, using the inductive hypothesis that

$$f(w)' \quad \Sigma R \quad \mathbf{zip}_k([\![w]\!], 0, \ldots, 0)',$$

we obtain

$$\begin{array}{rcl} f(xw)' & = & (\bar{x} f(w))' \\ & = & \bar{x}' f(w) + o(\bar{x}) f(w)' \\ & = & \bar{x}' f(w) + o(x) f(w)' \\ & = & \mathcal{X}^{k-1} f(x') f(w) + o(x) f(w)' \\ & = & \mathcal{X}^{k-1} f(x'w) + o(x) f(w)' \end{array}$$

$$\begin{array}{cl}
= & \left(\sum_{z\in X^*} \mathfrak{X}^{k-1}(f(x'w))(f(z))f(z)\right) + o(x)f(w)' \\
\Sigma R & \left(\sum_{z\in X^*} \mathfrak{X}^{k-1}\mathbf{zip}_k([\![(x'w)(z)z]\!]\right) + o(x)f(w)' \\
= & \mathfrak{X}^{k-1}\mathbf{zip}_k([\![x'w]\!],0,\ldots,0) + \mathfrak{X}^{k-1}o(x)\mathbf{zip}_k([\![w]\!],0,\ldots,0)' \\
= & \mathfrak{X}^{k-1}\mathbf{zip}_k([\![x]\!]'[\![w]\!] + o(x)[\![w']\!],0,\ldots,0) \\
= & \mathbf{zip}_k([\![xw]\!],0,\ldots,0)'
\end{array}$$

establishing that

$$f(xw)' \quad \Sigma R \quad \mathbf{zip}_k([\![xw]\!],0,\ldots,0)'.$$

It now follows that whenever $s\,R\,t$, it follows that $s'\,\Sigma R\,t'$, so R is indeed a bisimulation up to linearity. It now follows that $\bar{x}\,\Sigma R\,\mathbf{zip}_k([\![x]\!],0,\ldots,0)$, and thus that $[\![\bar{x}]\!]_Y = \mathbf{zip}_k([\![x]\!],0,\ldots,0))$, so $\mathbf{zip}_k([\![x]\!],0,\ldots,0)$ is constructively S-algebraic. □

We are now equipped with all ingredients required for the following theorem:

Theorem 4.7. *Given a commutative semiring S and any $k \geq 2$, if $\sigma_1 \ldots, \sigma_k \in S^{\mathbb{N}}$ are constructively S-algebraic, then*

$$\mathbf{zip}_k(\sigma_1, \ldots, \sigma_k)$$

is constructively S-algebraic.

Proof. Follows from Proposition 3.8, Lemma 4.5, and Proposition 4.6. □

So far, it remains an open question whether the operator **even** (or equivalently, **odd**), and the generalizations $\mathbf{unzip}_{i,k}$ too, preserve constructive algebraicity. The natural approach here would be to transform the given system into a new system, by setting the derivative of the nonterminals in the new system equal to the second derivatives in the old system. For example, we could try to transform the system

$$o(x) = 1 \qquad \text{and} \qquad x' = x^2$$

yielding the Catalan numbers, into the new system

$$o(\bar{x}) = 1 \qquad \text{and} \qquad \bar{x}' = \bar{x}^3 + \bar{x}.$$

However, the latter system yields the stream

$$1, 2, 10, 66, 498, 4066, 34970, 312066, 2862562, 26824386, \dots \qquad \text{(A027307)}$$

rather than the even-indexed Catalan numbers, and the suggested construction does not work.

5

k-Automatic and k-regular sequences

The present chapter gives a coalgebraic presentation of two, related, families of sequences that can be seen as being generated by automata. The k-automatic sequences, first considered as a class in [Cob72], can be regarded as sequences that are accepted, or generated, by a finite automaton. The k-regular sequences, introduced in [AS92], and further elaborated on in [AS03b], generalize the k-automatic sequences similarly to how recognizable power series generalize simple power series (or, more precisely, language partitions). A comprehensive reference for both k-automatic and k-regular sequences is [AS03a].

This chapter gives a coalgebraic presentation of both classes, relying on the so-called bijective base k numeration system, which differs from the familiar ('standard') base k numeration system. Our presentation can then be contrasted with more traditional presentations of the k-automatic and k-regular sequences, which are given either in terms of the standard base k numeration system, or in terms of the closely related k-kernel. Because of our switch to the bijective base k numeration system, we do not depend anymore on additional notions such as zero-consistency of automata (which was required for the coalgebraic presentation in [KR12]), and furthermore, the notions of k-automaticity and k-regularity now naturally include the case where $k = 1$ as well.

Section 5.1 starts with an introduction of the bijective base k numeration system, providing an, in this context, useful and elegant alternative for the standard base k numeration system. Next, this bijection is used to construct an *isomorphism between final automata* providing two distinct notions of 'stream semantics' and 'power series semantics'.

In Section 5.2, we apply this isomorphism to provide coalgebraic characterizations of k-automatic and k-regular sequences, as solutions to systems of **zip**-behavioural differential equations. The results in this section include, build on, and extend results from e.g. [KR12] and [GEH$^+$12]: the format of **zip**-behavioural differential equations, for k-automatic sequences, can be found in [GEH$^+$12], together with the realization that the class of streams can be assigned a final coalgebra (or final automaton) structure; we consider this fact in a wider context, and use this final coalgebra to additionally obtain a format for the k-regular sequences.

In Section 5.3, we apply the work in the previous section to a family of sequences defined by *divide-and-conquer recurrences*, and specify some conditions under which it follows that such sequences are k-regular.

Section 5.4 concludes by giving some connections between the k-automatic sequences, and algebraic streams. We consider a well-known result due to Christol and a lesser known result due to Fliess, and see how these results fit in the coalgebraic approach presented in this chapter.

5.1 An isomorphism between final automata

Given any $k \in \mathbb{N}$ with $k \geq 1$, consider the alphabet of digits

$$A_k := \{\mathtt{1}, \ldots, \mathtt{k}\}$$

and define a function $\mathbf{val} : A_k \to \mathbb{N}$ by $\mathbf{val}(\mathtt{i}) = i$ for all $i \in \mathbb{N}$ with $1 \leq i \leq k$. This may appear somewhat pedantic and unnecessary, but we need to pay close care to distinguish between concatenation of words of digits, and multiplication of natural numbers, both denoted by the (generally omitted) '$\cdot$' symbol. Likewise, we need to distinguish between the empty word '1', and the digit '$\mathtt{1}$'.

Now consider the mapping

$$\nu_k : (A_k)^* \to \mathbb{N}$$

defined inductively as follows:

$$\nu_k(1) = 0 \qquad \nu_k(dw) = k\nu_k(w) + \mathbf{val}(d)$$

It is easily established that ν_k is a bijection, and that, given a word $d_0 \ldots d_n$ of digits, we have

$$\nu_k(d_0 \ldots d_n) = \sum_{i=1}^{n} \mathbf{val}(d_i)k^i.$$

We regard ν_k as specifying the *bijective base k numeration system*, presented with the least significant digit first. Note that ν_k can, unlike the standard base k numeration system for which there is no unary system entirely analogous to the base k systems for $k > 1$, be defined for all $k \geq 1$.

Furthermore, we will in what follows use a (curried) variant of **unzip**,

$$\mathbf{unzip}_k : S^{\mathbb{N}} \to (S^{\mathbb{N}})^{A_k},$$

defined so that $\mathbf{unzip}_k(\sigma)(i) = \mathbf{unzip}_{\mathbf{val}(i)-1,k}(\sigma)$.

In a similar manner, we can regard $\mathbf{zip}_k$ as being of the type

$$\mathbf{zip}_k : (S^{\mathbb{N}})^{A_k} \to S^{\mathbb{N}}$$

thus creating a bijection between $S^{\mathbb{N}}$ and $(S^{\mathbb{N}})^{A_k}$:

$$\begin{tikzcd} S^{\mathbb{N}} \arrow[r, shift left, "\mathbf{unzip}_k"] & (S^{\mathbb{N}})^{A_k} \arrow[l, shift left, "\mathbf{zip}_k"] \end{tikzcd}$$

In what follows, we will usually simply write $(S^{\mathbb{N}})^k$ instead of $(S^{\mathbb{N}})^{A_k}$.

We now recall from Section 2.1 that, given any finite alphabet A, the automaton

$$\begin{tikzcd} S\langle\langle A\rangle\rangle \arrow[d, "(O,\Delta)"] \\ S \times S\langle\langle A\rangle\rangle^A \end{tikzcd}$$

is a final S-automaton over the alphabet A, or equivalently, a final coalgebra for the functor $S \times -^A$.

Assuming that $A = A_k$ for some $k \in \mathbb{N}$ with $k \geq 1$, we thus have a bijection

$$\begin{tikzcd} (A^k)^* \arrow[r, shift left, "\nu_k"] & \mathbb{N} \arrow[l, shift left, "\nu_k^{-1}"] \end{tikzcd}$$

which extends to a bijection

$$\begin{tikzcd} S^{\mathbb{N}} \arrow[r, shift left, "- \circ \nu_k"] & S\langle\langle A_k\rangle\rangle \arrow[l, shift left, "- \circ \nu_k^{-1}"] \end{tikzcd}$$

because $S\langle\langle A_k\rangle\rangle \cong (A_k)^* \to S$. Observe that the type of ν_k is $(A_k)^* \to \mathbb{N}$, and therefore, $- \circ \nu_k$ takes functions of type $\mathbb{N} \to S$ and maps them to functions $(A_k)^* \to S$ (and vice versa for the inverse).

Now consider the system

$$\begin{tikzcd} S^{\mathbb{N}} \arrow[d, "(\mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})"] \\ S \times (S^{\mathbb{N}})^k \end{tikzcd}$$

which is a $S \times -^{A_k}$ coalgebra as a result of the isomorphism $(S^{\mathbb{N}})^k \cong (S^{\mathbb{N}})^{A_k}$. We will now establish that this coalgebra (or automaton) is again final, and thus isomorphic to the final coalgebra that was presented in Chapter 2.

Proposition 5.1. *The diagram*

$$\begin{tikzcd} S^{\mathbb{N}} \arrow[r, shift left, "- \circ \nu_k"] \arrow[d, "(\mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})"'] & S\langle\langle A_k \rangle\rangle \arrow[l, shift left, "- \circ \nu_k^{-1}"] \arrow[d, "(O, \Delta)"] \\ S \times (S^{\mathbb{N}})^k \arrow[r, shift left, "1_S \times (- \circ \nu_k)^k"] & S \times S\langle\langle A_k \rangle\rangle^k \arrow[l, shift left, "1_S \times (- \circ \nu_k^{-1})^k"] \end{tikzcd}$$

commutes and thus $(S^{\mathbb{N}}, \mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})$ is a final S-automaton over the alphabet A_k.

Proof. This amounts to showing that $- \circ \nu_k$ is an isomorphism of S-automata, for which it is sufficient to show that $- \circ \nu_k$ is a bijective homomorphism: furthermore, we already have established that $- \circ \nu_k$ is bijective.

To show that $- \circ \nu_k$ is a homomorphism, we first have to show that

$$\mathbf{head}(\sigma) = O(\sigma \circ \nu_k),$$

which holds because

$$\mathbf{head}(\sigma) = \sigma(0) = \sigma(\nu_k(1)) = (\sigma \circ \nu_k)(1) = O(\sigma \circ \nu_k).$$

Second, we have to show that, for all i with $1 \leq i \leq k$, $(\sigma \circ \nu_k)_{\mathtt{i}} = (\sigma_{\mathtt{i}} \circ \nu_k)$, or using more explicit notation:

$$\Delta(\sigma \circ \nu_k, \mathtt{i}) = ((\mathbf{unzip}_k \circ \mathbf{tail})(\sigma)(\mathtt{i}) \circ \nu_k)$$

This again holds, because given any $w \in A_k^*$ and $\mathtt{i} \in A_k$, we have:

$$\begin{aligned}
(\sigma \circ \nu_k)_{\mathtt{i}}(w) &= (\sigma \circ \nu_k)(\mathtt{i} \cdot w) \\
&= \sigma(\nu_k(\mathtt{i} \cdot w)) \\
&= \sigma(i + k \cdot \nu_k(w)) \\
&= \sigma'((i-1) + k \cdot \nu_k(w)) \\
&= \mathbf{unzip}_{i-1,k}(\sigma')(\nu_k(w)) \\
&= ((\mathbf{unzip}_{i-1,k} \circ \mathbf{tail})(\sigma) \circ \nu_k)(w) \\
&= (\sigma_{\mathtt{i}} \circ \nu_k)(w)
\end{aligned}$$

□

The observation that $(S^{\mathbb{N}}, \mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})$ is a final coalgebra first appeared in [GEH+12, Proposition 26]: there this result was established directly, rather than by establishing the existence of an isomorphism to a known final coalgebra.

We now introduce the notation $[\![-]\!]_k$ for the unique mapping of an arbitrary S-automaton into the automaton $(S^{\mathbb{N}}, \mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})$, directly yielding the equalities

$$[\![-]\!]_k = [\![-]\!] \circ \nu_k \qquad \text{and} \qquad [\![-]\!] = [\![-]\!]_k \circ \nu_k^{-1}.$$

Furthermore, given any S-automaton (Q, o, δ) over the alphabet A_k, we get an instance of the following commuting diagram:

$$\begin{tikzcd}
Q \arrow[r, dotted, "{[\![-]\!]_k}"] \arrow[d, "{(o,\delta)}"'] & S^{\mathbb{N}} \arrow[d, "{(\mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})}"] \\
S \times Q^k \arrow[r, dotted, "{1_S \times ([\![-]\!]_k)^k}"] & S \times (S^{\mathbb{N}})^k
\end{tikzcd}$$

The commutativity of this diagram can equivalently be captured by means of the equations

$$o(q) = \mathbf{head}([\![q]\!]_k) \qquad \text{and} \qquad [\![q_{\mathtt{i}}]\!]_k = \mathbf{unzip}_{i-1,k}([\![q]\!]'_k) \tag{5.1}$$

which hold for all $q \in Q$, and $i \in \mathbb{N}$ with $1 \leq i \leq k$. Using the earlier bijection between the operations $\mathbf{zip}_k$ and $\mathbf{unzip}_k$, the second of these equations can be restated as

$$[\![q]\!]'_k = \mathbf{zip}_k([\![q_{\mathtt{1}}]\!]_k, \ldots, [\![q_{\mathtt{k}}]\!]_k).$$

5.2 Systems of zip-behavioural differential equations

5.2.1 Simple systems and k-automatic sequences

Given any set S, we call a sequence $\sigma \in S^{\mathbb{N}}$ *k-automatic* whenever there is a finite set $\Sigma \subseteq S^{\mathbb{N}}$ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$, there are $\tau_1, \ldots, \tau_k \in \Sigma$ such that

$$\tau = \mathbf{zip}_k(\tau_1, \ldots, \tau_k).$$

This characterization is equivalent to the condition that the *k-kernel* of σ, that is, the smallest set of streams containing σ and closed under the $\mathbf{unzip}_{i,k}$ operations, is finite.

Conventionally, the k-automatic sequences are defined in terms of finite automata and the standard base k numeration. It is well-known and not overly difficult to prove (see e.g. [AS03a, Theorem 6.6.2]) that the definition using finite automata and the standard base k numeration is in direct correspondence to the definition in terms of the k-kernel.

Existing coalgebraic and coinductive approaches to automatic sequences can be found in [GEH$^+$12] and [KR12]. In [KR12], a coalgebra is presented that is 'relatively' final, i.e. final amongst automata that are *zero-consistent*: again, this presentation is shown to be equivalent to the presentation given above.

The following proposition is essentially a rephrasing of [AS03a, Theorem 5.2.7], which already relies on the bijective base k numeration system.

Proposition 5.2. *For any $k \geq 2$, a sequence $\sigma \in S^{\mathbb{N}}$ is k-automatic if and only if there is a finite S-automaton (Q, o, δ) and a $q \in Q$ such that $[\![q]\!]_k = \sigma$.*

Proof. First assume that $\sigma \in S^{\mathbb{N}}$ is k-automatic. By definition, there is a finite set Σ, such that for each $\tau \in \Sigma$ there are $\tau_1, \ldots, \tau_k \in \Sigma$ such that

$$\tau = \mathbf{zip}_k(\tau_1, \ldots, \tau_k).$$

Taking derivatives, we obtain

$$\tau' = \mathbf{zip}_k(\tau_2, \ldots, \tau_k, \tau_1')$$

and taking derivatives a second time, we obtain

$$\tau'' = \mathbf{zip}_k(\tau_3, \ldots, \tau_k, \tau_1', \tau_2')$$

(or, in the case where $k = 2$, simply $\tau'' = \mathbf{zip}(\tau_1', \tau_2')$). Defining

$$Q := \Sigma \cup \{\tau' \mid \tau \in \Sigma\}$$

it follows that for all $q \in Q$ and all $i \in \mathbb{N}$ with $i < k$, $\mathbf{unzip}_{i,k}(q) \in Q$ again.

This directly implies that $(Q, \mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})$ is a finite S-automaton. This automaton is a sub-automaton of the final automaton, and hence $[\![-]\!]_k$ is simply the identity mapping.

Conversely, assume that there is a finite S-automaton (Q, o, δ) and a $q \in Q$ such that $[\![q]\!]_k = \sigma$. We thus get, for all $q \in Q$

$$[\![q]\!]_k' = \mathbf{zip}_k([\![q_{\mathtt{1}}]\!]_k, \ldots, [\![q_{\mathtt{k}}]\!]_k)$$

Using (4.2), we now get

$$[\![q]\!]_k = o(q) :: [\![q]\!]_k' = \mathbf{zip}_k(o(q) :: [\![q_{\mathtt{k}}]\!]_k, [\![q_{\mathtt{1}}]\!]_k, \ldots, [\![q_{\mathtt{k-1}}]\!]_k)$$

and applying (4.2) another time, we get for an arbitrary $s \in S$:

$$s :: [\![q]\!]_k = \mathbf{zip}_k(s :: [\![q_{\mathtt{k-1}}]\!]_k, o(q) :: [\![q_{\mathtt{k}}]\!]_k, [\![q_{\mathtt{1}}]\!]_k, \ldots, [\![q_{\mathtt{k-2}}]\!]_k)$$

Hence, it follows that the finite set

$$\{[\![q]\!]_k \mid q \in Q\} \cup \{s :: [\![q]\!]_k \mid q \in Q, s \in \{o(r) \mid r \in Q\}\}$$

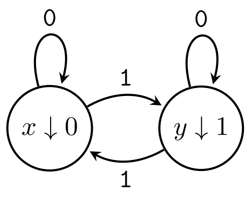
satisfies the defining property of automaticity. □

We now directly obtain the following corollaries:

Corollary 5.3. *Given any $\sigma \in S^{\mathbb{N}}$, σ is k-automatic if and only if $\nu_k \circ \sigma$ is simple.*

Corollary 5.4. *Given any $\sigma \in S^{\mathbb{N}}$, σ is k-automatic if and only if there is a finite $\Sigma \subseteq S^{\mathbb{N}}$ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$ and each i with $0 \leq i < k$, $\mathbf{unzip}_{i,k}(\tau') \in \Sigma$.*

Example 5.5. As an example of a 2-automatic sequence, consider the following zero-consistent 2-automaton over the alphabet B_2:



It is well-known from e.g. [AS03a] that x generates the stream

$$0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, \ldots \quad \text{(A010060)}$$

which is known as Prouhet-Thue-Morse sequence.

This stream can also be characterized as the *morphic sequence* generated by the morphism $p : \mathbb{F}_2 \to (\mathbb{F}_2)^*$ with $p(0) = 01$ and $p(1) = 10$, starting from the symbol 0: if we repeatedly apply this substitution on the (finite) word obtained by the previous substitution, the Prouhet-Thue-Morse sequence is obtained in the limit.

This automaton structure corresponds to the **zip**-equations

$$x = \mathbf{zip}(x, y) \qquad y = \mathbf{zip}(y, x)$$

combined with the outputs

$$o(x) = 0 \qquad o(y) = 1$$

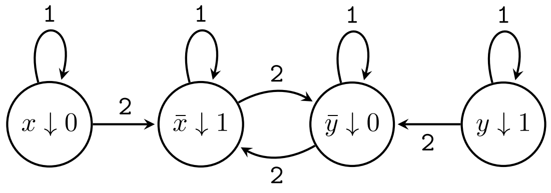
Taking derivatives and second derivatives, we obtain

$$\begin{array}{rcl} x' & = & \mathbf{zip}(y, x') \\ x'' & = & \mathbf{zip}(x', y') \end{array} \qquad \begin{array}{rcl} y' & = & \mathbf{zip}(x, y') \\ y'' & = & \mathbf{zip}(y', x') \end{array}$$

yielding the following system of **zip**-behavioural differential equations (which introduces new variables $\bar{x}, \bar{y}$ to stand for x' and y' respectively:

$$\begin{array}{rcl} o(x) & = & 0 \\ o(y) & = & 1 \\ o(\bar{x}) & = & 1 \\ o(\bar{y}) & = & 0 \end{array} \qquad \begin{array}{rcl} x' & = & \mathbf{zip}(x, \bar{y}) \\ y' & = & \mathbf{zip}(y, \bar{x}) \\ \bar{x}' & = & \mathbf{zip}(\bar{x}, \bar{y}) \\ \bar{y}' & = & \mathbf{zip}(\bar{y}, \bar{x}) \end{array}$$

This system again is equivalent to a finite automaton, this time an automaton with four states over the alphabet A_2:


$$\begin{tikzcd} x \downarrow 0 \arrow[loop above, "1"] \arrow[r, "2"] & \bar{x} \downarrow 1 \arrow[loop above, "1"] \arrow[r, bend left, "2"] & \bar{y} \downarrow 0 \arrow[loop above, "1"] \arrow[l, bend left, "2"] & y \downarrow 1 \arrow[loop above, "1"] \arrow[l, "2"] \end{tikzcd}$$

Example 5.6. Another example is given by the Cantor sequence

$$1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots \quad \text{(A088917)}$$

which is 3-automatic and can be characterized by the $\mathbf{zip}_3$-equations

$$\begin{array}{rclcrcl} o(x) & = & 1 & \qquad & x & = & \mathbf{zip}_3(x, y, x) \\ o(y) & = & 0 & \qquad & y & = & \mathbf{zip}_3(y, y, y) \end{array}$$

The construction now yields the following system of $\mathbf{zip}_3$-behavioural differential equations:

$$\begin{array}{rclcrcl} o(x) & = & 1 & \qquad & x' & = & \mathbf{zip}_3(y, x, \bar{x}) \\ o(y) & = & 0 & \qquad & y' & = & \mathbf{zip}_3(y, y, \bar{y}) \\ o(\bar{x}) & = & 0 & \qquad & \bar{x}' & = & \mathbf{zip}_3(x, \bar{x}, \bar{y}) \\ o(\bar{y}) & = & 0 & \qquad & \bar{y}' & = & \mathbf{zip}_3(y, \bar{y}, \bar{y}) \end{array}$$

5.2.2 Linear systems and k-regular sequences

The notion of a k-regular sequence with values in a ring was introduced in [AS92]. The following definition is (roughly) a direct generalization to sequences with values in a semiring.

Given a semiring S, we call a sequence $\sigma \in S^{\mathbb{N}}$ (S, k)-*regular* (or sometimes simply k-regular) whenever there is a finite set of generators $\Sigma = \{\sigma_0, \ldots \sigma_{n-1}\}$ with $\sigma \in \Sigma$, and an indexed family $a_{h,i,j}$ for all $h, i, j \in \mathbb{N}$ with $h < n$, $i < n$, $j < k$, such that for all $h < n$ and $j < k$

$$\mathbf{unzip}_{j,k}(\sigma_h) = \sum_{i=0}^{n} a_{h,i,j} \sigma_i$$

or equivalently, for all $h < n$:

$$\sigma_h = \mathbf{zip}_k \left(\sum_{i=0}^{n} a_{h,i,0} \sigma_i, \ldots, \sum_{i=0}^{n} a_{h,i,k} \sigma_i \right). \tag{5.2}$$

This is again equivalent to the condition that the k-kernel of σ is contained in a finitely generated S-submodule of $S^{\mathbb{N}}$.

Because of the finality of the automaton $(S^{\mathbb{N}}, \mathbf{head}, \mathbf{unzip}_k \circ \mathbf{tail})$, we obtain, for any S-weighted automaton (X, o, δ), the following diagram, a variant of diagram (2.3):

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^1_X"] \arrow[d, "(o,\delta)"'] & \mathrm{Lin}_S(X) \arrow[r, dotted, "[\![-]\!]_k"] \arrow[dl, "(\hat{o},\hat{\delta})"'] & S^{\mathbb{N}} \arrow[d, "(\mathbf{head},\, \mathbf{unzip}_k \circ \mathbf{tail})"] \\
S \times (\mathrm{Lin}_S(X))^k \arrow[rr, dotted, "1_S \times ([\![-]\!]_k)^k"] & & S \times (S^{\mathbb{N}})^k
\end{tikzcd} \tag{5.3}$$

From this diagram, we immediately can derive the following lemma:

Lemma 5.7. *Given an S-weighted A_k-automaton (X, o, δ), $[\![-]\!]_k$ is the unique linear mapping* $\mathrm{Lin}_S(X) \to S^{\mathbb{N}}$ *satisfying, for each $x \in X$, the equations*

$$\mathbf{head}([\![\eta^1(x)]\!]_k) = o(x) \qquad \textit{and} \qquad [\![\eta^1(x)]\!]'_k = \mathbf{zip}_k([\![\delta(x,\mathtt{1})]\!]_k, \ldots, [\![\delta(x,\mathtt{k})]\!]_k).$$

Hence, whenever we are concerned with the stream semantics $[\![-]\!]_k$, we are justified in presenting a weighted automaton (X, o, δ) as a system of **zip**-behavioural differential equations, containing for each $x \in X$ equations

$$\mathbf{head}(x) = o(x) \qquad \text{and} \qquad x' = \mathbf{zip}(\delta(x,\mathtt{1}), \ldots, \delta(x,\mathtt{k})).$$

The preceding lemma now tells us that the mapping $[\![-]\!]_k$ gives us the unique solution to such a system of behavioural differential equations.

We will now show that k-regular sequences are obtained precisely by the stream semantics of finite S-weighted A_k-automata. It will follow that k-regular sequences are in bijective correspondence with recognizable formal power series via k-adic numeration. The following proposition can be seen as a generalization of Proposition 5.2, extended by taking into account linearity:

Proposition 5.8. *Given a semiring S and any $k \geq 2$, a sequence $\sigma \in S^{\mathbb{N}}$ is S,k-regular if and only if there is a finite S-weighted automaton (X, o, δ) over the alphabet A_k and an $x \in X$, such that $[\![x]\!]_k = \sigma$.*

Proof. First assume that σ is S,k-regular. In this case, there is a finite set of sequences $\Sigma = \{\sigma_0, \ldots, \sigma_{n-1}\}$ with $\sigma \in \Sigma$, and scalars a_h and $b_{h,i,j}$ in S indexed over $h < n$, $i < n$, $j < k$, such that for all $h < n$:

$$\sigma_h = \mathbf{zip}_k \left(\sum_{i<n} a_{h,i,0}\sigma_i, \ldots, \sum_{i<n} a_{h,i,k-1}\sigma_i \right)$$

Taking the derivative and second derivative of each σ_h using (4.1), we obtain:

$$\begin{aligned}\sigma'_h &= \mathbf{zip}_k\left(\sum_{i<n} a_{h,i,1}\sigma_i, \ldots, \sum_{i<n} a_{h,i,0}\sigma'_i\right)\\ \sigma''_h &= \mathbf{zip}_k\left(\sum_{i<n} a_{h,i,2}\sigma_i, \ldots, \sum_{i<n} a_{h,i,0}\sigma'_i, \sum_{i\le n} a_{h,i,1}\sigma'_i\right)\end{aligned}$$

Hence, for each $\sigma \in \Sigma^+ := \Sigma \cup \{\sigma' \mid \sigma \in \Sigma\}$ and $j < k$, $\mathbf{unzip}_{j,k}(\sigma')$ is a S-linear combination of elements from Σ^+, and hence there is a finite S-weighted A_k-automaton (X, o, δ) and an $x \in X$, such that $[\![x]\!]_k = \sigma$ by Lemma 5.7.

Conversely, assume that there is a finite S-weighted automaton (X, o, δ) over A_k, and a state $x \in X$, such that $[\![\eta^1(x)]\!]_k = \sigma$. By Lemma 5.7, we have

$$[\![x]\!]'_k = \mathbf{zip}_k([\![\delta(x, \mathtt{1})]\!]_k, \ldots, [\![\delta(x, \mathtt{k})]\!]_k)$$

and by (4.1) that

$$[\![x]\!]_k = \mathbf{zip}_k(o(x) + \mathcal{X}[\![\delta(x, \mathtt{k})]\!]_k, [\![\delta(x, \mathtt{1})]\!]_k, \ldots, [\![\delta(x, \mathtt{k\text{-}1})]\!]_k). \tag{5.4}$$

Using the fact that $(\mathcal{X}[\![x]\!]_k)' = [\![x]\!]_k$, and applying again (4.1), we obtain:

$$\mathcal{X}[\![x]\!]_k = \mathbf{zip}_k(\mathcal{X}[\![x_{\mathtt{k\text{-}1}}]\!]_k, o(x) + \mathcal{X}[\![x_{\mathtt{k}}]\!]_k, [\![x_{\mathtt{1}}]\!]_k, \ldots, [\![x_{\mathtt{k\text{-}2}}]\!]_k) \tag{5.5}$$

By defining the set of generators

$$\Sigma = \{[\![x]\!]_k \mid x \in X\} \cup \{\mathcal{X}[\![x]\!]_k \mid x \in X\} \cup \{(1, 0, 0, \ldots)\}$$

the equations (5.4) and (5.5) show (via the zip-unzip isomorphism (4.3)) that for each generator $\sigma \in \Sigma$ and $j < k$, $\mathbf{unzip}_{j,k}(\sigma)$ is an S-linear combination of the generators. It follows from the definition that $[\![x]\!]_k$ is k-regular for all $x \in X$. □

This also gives us the following corollary, establishing the connection to the recognizable power series:

Corollary 5.9. *A sequence $\sigma \in S^{\mathbb{N}}$ is k-regular if and only if $\sigma \circ \nu_k$ is recognizable.*

This corollary is analogous to [AS92, Theorem 4.3], which says that $\sigma \in \mathbb{Z}^{\mathbb{N}}$ is k-regular if and only if the formal power series $\sum_{n<\omega} \sigma(n)\bar{\xi}(n)$ is rational (or equivalently, recognizable), where $\bar{\xi}\colon \mathbb{N} \to B_k^* \backslash B_k^* 0$ is the inverse of the bijection obtained by restricting the standard base k numeration $\xi\colon B_k^* \to \mathbb{N}$ to words not ending in `0`.

Again, we obtain a semantic characterization in terms of finite sets of formal power series:

Corollary 5.10. *Given any $\sigma \in S^{\mathbb{N}}$, σ is S,k-regular if and only if there is a finite $\Sigma \subseteq S^{\mathbb{N}}$ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$ and each i with $0 \le i < k$, $\mathbf{unzip}_{i,k}(\tau')$ is S-linear in Σ.*

Example 5.11. We illustrate Proposition 5.8 with a well-known 2-regular sequence, which the composer Per Nørgård used in a variety of his compositions, and which he called the *infinity sequence*[1]:

$$0, 1, -1, 2, 1, 0, -2, 3, -1, 2, 0, 1, 2, -1, -3, 4, \ldots \quad \text{(A004718)}$$

This sequence can be characterized uniquely by the following equations:

$$\begin{array}{rclcrcl} o(x) & = & 0 & \qquad & x & = & \mathbf{zip}(-x, x+y) \\ o(y) & = & 1 & & y & = & \mathbf{zip}(y, y) \end{array}$$

(with x denoting σ). The **zip**-equations on the right-hand side are a system in the format of (5.2) and hence the sequence is 2-regular. Taking derivatives and second derivatives of the **zip**-equations, we now get using (4.1):

$$\begin{array}{rclcrcl} x' & = & \mathbf{zip}(x+y, -x') & \qquad & x'' & = & \mathbf{zip}(-x', x'+y') \\ y' & = & \mathbf{zip}(y, y') & & y'' & = & \mathbf{zip}(y', y') \end{array}$$

We can now compute the initial values of x' and y' as

$$\begin{array}{l} o(x') = o(\mathbf{zip}(x+y, -x')) = o(x+y) = o(x) + o(y) = 1 \\ o(y') = o(\mathbf{zip}(y, y)) = o(y) = 1. \end{array}$$

Introducing new variables z and w representing x' and y' respectively, we now can specify a weighted automaton as the unique solution to the following system of **zip**-equations:

$$\begin{array}{rclcrcl} o(x) & = & 0 & \qquad & x' & = & \mathbf{zip}(x+y, -z) \\ o(y) & = & 1 & & y' & = & \mathbf{zip}(y, w) \\ o(z) & = & 1 & & z' & = & \mathbf{zip}(-z, z+w) \\ o(w) & = & 1 & & w' & = & \mathbf{zip}(w, w) \end{array}$$

[1]`http://pernoergaard.dk/eng/strukturer/uendelig/uindhold.html`

In this automaton, the final homomorphism $[\![-]\!]_k$ maps x to Nørgård's infinity sequence. We remark, however, that this weighted automaton is not minimal, as y and w both are mapped onto the constant sequence of ones.

A corresponding weighted automaton, with inputs in A_2 (presented here restricted to just the variables z and w), can be derived immediately from the system of behavioural differential equations, as follows:

$$\begin{tikzcd}
z \downarrow 1 \arrow[loop above, "1|-1"] \arrow[loop below, "2|1"] \arrow[r, "2|1"] & w \downarrow 1 \arrow[loop above, "2|1"] \arrow[loop below, "1|1"]
\end{tikzcd}$$

Example 5.12. Another example, which can be constructed in the same manner as the previous example, is given by the following $\mathbb{N}$-weighted A_2-automaton:

$$\begin{array}{rclcrcl}
o(x) & = & 1 & \qquad & x' & = & \mathbf{zip}(x, x) \\
o(y) & = & 1 & & y' & = & \mathbf{zip}(2y, 2y + x) \\
o(z) & = & 1 & & z' & = & \mathbf{zip}(z, x + y)
\end{array}$$

Here, the final homomorphism $[\![-]\!]_2$ maps x onto the constant stream of ones, y onto the stream of natural numbers, and z onto Kimberling's sequence:

$$1, 1, 2, 1, 3, 2, 4, 1, 5, 3, 6, 2, 7, 4, 8, 1, \ldots \quad \text{(A003602)}$$

5.3 Divide and conquer recurrences

Divide and conquer recurrences, which have been considered for example in [GKP94] and [Ste03], can somewhat informally be seen as—in the case of $k = 2$, to which we will restrict ourselves in this section—sequences σ where $\sigma(0)$ is given, and for each n, $\sigma(n)$ is defined in terms of $\sigma(\lfloor (n-1)/2 \rfloor)$, $\sigma(\lceil (n-1)/2 \rceil)$, and polynomials in n.

In this section, we will establish a close link between divide and conquer recurrences satisfying a number of 'natural' conditions, and the k-regular sequences, by showing that their sequences occur as 2-regular sequences.

We will restrict ourselves to special (more precisely defined) restricted versions of divide and conquer recurrences. To be precise, we will consider recurrences of the form

$$\sigma(2n) = a_2\sigma(n-1) + a_3\sigma(n) + \tau_1(n) \qquad \sigma(2n+1) = a_1\sigma(n) + \tau_0(n)$$

where a_1, a_2, a_3 are scalars from the semiring S, and τ_1 and τ_2 are themselves 2-regular sequences. We furthermore hold the assumption that $\sigma(0) = 0$ (we will later see that this assumption can be relaxed).

Now observe that

$$\sigma(2n+2) = \sigma(2(n+1)) = a_2\sigma(n) + a_3\sigma(n+1) + \tau_1'(n).$$

As a result of the equalities $\sigma(2n+1) = (\mathbf{even}(\sigma'))(n)$ and $\sigma(2n+2) = (\mathbf{odd}(\sigma'))(n)$ we now derive

$$\begin{array}{rcl}(\mathbf{even}(\sigma'))(n) & = & (a_1\sigma + \tau_0)(n) \\ (\mathbf{odd}(\sigma'))(n) & = & (a_2\sigma + a_3\sigma' + \tau_1')(n)\end{array}$$

and hence also

$$\mathbf{even}(\sigma') = a_1\sigma + \tau_0 \qquad \mathbf{odd}(\sigma') = a_2\sigma + a_3\sigma' + \tau_1'.$$

A large amount of combinatorial problems can be expressed by means of divide and conquer recurrences of this type, and can be transformed using the above construction. An overview of many of these examples can be found at `http://oeis.org/somedcgf.html`. For some of these examples, Haskell-specifications corresponding to the behavioural differential equations can be found in Appendix C.

One of the questions asked on `http://oeis.org/somedcgf.html`, is whether all the examples presented there are indeed 2-regular. We will soon see that this question can be answered positively.

Example 5.13. As an example illustrating the construction, the recurrence given by

$$\sigma(0) = 0 \qquad \sigma(2n) = \sigma(n) + \sigma(n-1) + 2n - 2 \qquad \sigma(2n+1) = 2\sigma(n) + 2n - 1$$

specifying the sorting numbers (OEIS A001855) can first be transformed into

$$\sigma(0) = 0 \qquad \sigma(2n+1) = 2\sigma(n) + 2n - 1 \qquad \sigma(2n+2) = \sigma(n+1) + \sigma(n) + 2n$$

and from there into the behavioural differential equation:

$$\mathbf{head}(\sigma) = 0 \quad \mathbf{even}(\sigma') = 2\sigma + 2\cdot\mathbf{nats} - \mathbf{ones} \quad \mathbf{odd}(\sigma') = \sigma' + \sigma + 2\cdot\mathbf{nats}$$

We can now establish that σ is again 2-regular:

Proposition 5.14. *Let τ_0 and τ_1 be 2-regular sequences over a semiring S, and let a_0, a_1, a_2, and a_3 be elements of S. Then there is a unique sequence σ satisfying*

$$\sigma(0) = a_0 \qquad \mathbf{even}(\sigma') = a_1\sigma + \tau_0 \qquad \mathbf{odd}(\sigma') = a_2\sigma + a_3\sigma' + \tau_1$$

which is again 2-regular.

Proof. If τ_0 and τ_1 are 2-regular, there are finite weighted automata (X_0, o_0, δ_0) and (X_1, o_1, δ_1) with elements $x_0 \in X_0$, $x_1 \in X_1$ such that $[\![x_0]\!]_2 = \tau_0$ and $[\![x_1]\!]_2 = \tau_1$.

Observe that, if σ satisfies the above equation, we can directly derive:

$$\sigma'(0) = a_1a_0 + o(\tau_0) \qquad \mathbf{even}(\sigma'') = a_2\sigma + a_3\sigma' + \tau_1 \quad \mathbf{odd}(\sigma'') = a_1\sigma' + \tau_0'$$

We thus specify a system $(X_0 \cup X_1 \cup \{y, z\}, o, \delta)$ satisfying the behavioral differential equations for X_0 and X_1 as before, and additionally:

$$\begin{array}{rclrcl} o(y) & = & a_0 & y' & = & \mathbf{zip}(a_1y + o(x_0), a_2y + a_3z + x_1) \\ o(z) & = & a_1a_0 + o(x_0) & z' & = & \mathbf{zip}(a_2y + a_3z + x_1, a_1z + x_0') \end{array}$$

By Lemma 5.7, this system has a unique solution, in which $[\![y]\!]_2$ satisfies the equations for σ and $[\![z]\!]_2 = [\![y]\!]_2'$. Because, given systems for τ_0 and τ_1, any solution to the original equation for σ has to satisfy all equations in the composite system, the solution for σ also is unique. $\square$

5.4 Theorems by Fliess and Christol

We now turn to the connection between the notion of constructive algebraicity, introduced in Chapter 3, and the classical notion of an *algebraic* power series (over a single variable, hence, a stream) over any field F, generalizing the notion of an *algebraic function.*

The main result of this section states that if a stream $\sigma \in (\mathbb{F}_q)^{\mathbb{N}}$ is q-automatic, then it is constructively $\mathbb{F}_q$-algebraic. This is a known result, obtained in a new way, and can be related both to Christol's well-known theorem, stating that any $\sigma \in (\mathbb{F}_q)^{\mathbb{N}}$ is q-automatic if and only if it is $\mathbb{F}_q$-algebraic, and to a lesser-known result due to Fliess, stating that any $\sigma \in F^{\mathbb{N}}$ (for any perfect field F) is $\mathbb{F}_q$-algebraic if and only if it is constructively $\mathbb{F}_q$-algebraic.

First, we recall some basic facts about fields (which can be found in [LB99] or other textbooks on algebra). If there are elements $n \in \mathbb{N}$ with $n \neq 0$ such that $h(n) = 0_F$, we say F has *characteristic* p if p is the smallest number with this condition; if there are no such elements, we say that F has characteristic 0. Whenever F has characteristic $p \neq 0$, p is necessarily a prime number.

For each prime number p and each natural number $k \geq 1$, there is exactly one finite field of size p^k, denoted by $\mathbb{F}_{p^k}$; the characteristic of such a field is equal to p.

A field F is called *perfect* when F has either characteristic 0, or when F has characteristic p and, for every $f \in F$, there is a $g \in F$ such that $f = g^p$. (Here we inductively define $g^0 = 1$ and $g^{n+1} = g \cdot g^n$.) The fields $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, and all finite fields are perfect: an example of a field that is not perfect is the field of Laurent series over a finite field.

We call a stream $\sigma \in F^{\mathbb{N}}$ algebraic over F, or *F-algebraic*, if and only if there exist polynomial F-streams $p_1, \ldots, p_n$, with at least one p_i not equal to zero, such that:

$$\sum_{i \leq n} p_i \sigma^i = 0$$

This is precisely the classical definition (see e.g. [BR11, Section 4.4]) adapted to the terminology of streams.

The following result, which first appeared as [Fli74, Proposition 7], establishes that streams are constructively F-algebraic if and only if they are F-algebraic. (In [Fli74], the notion of constructive algebraicity is defined in terms of strong solutions to proper systems of equations.)

Proposition 5.15. *Let F be a perfect field. Then a stream $\sigma \in F^{\mathbb{N}}$ is F-algebraic if and only if it is constructively F-algebraic.*

The proof relies on first using a generalisation by Christol (see [Chr70]) of Furstenberg's theorem (see [Fur67]), to transform algebraic streams into diagonals of rational power series in two commuting variables, which in turn can be transformed into a systems of equations using a construction given in [Fli74]. Combining these results with the results from Section 3.3.2, it directly follows that streams over perfect fields are F-algebraic if and only if they occur as the solution to some polynomial system of behavioural differential equations.

The following lemma is a translation into stream calculus notation of the generating function equation

$$A^q(X) = A(X^q)$$

which is well-known to hold for finite fields (see e.g. [AS03a, Exercise 2.16]).

Lemma 5.16. *Given a prime power q and any $\sigma \in (\mathbb{F}_q)^{\mathbb{N}}$, we have*

$$\sigma^q = \mathbf{zip}_q(\sigma, 0, \dots, 0).$$

The following proposition corresponds to one of the directions of Christol's theorem, with the exception that we prove σ to be constructively $\mathbb{F}_q$-algebraic, rather than traditionally $\mathbb{F}_q$-algebraic:

Proposition 5.17. *For any prime power q and any $\sigma \in (\mathbb{F}_q)^{\mathbb{N}}$, if σ is q-automatic, then σ is constructively $\mathbb{F}_q$-algebraic.*

Proof. If σ is q-automatic, then there is a finite set Σ, such that for any $\tau \in \Sigma$ there are $\sigma_1, \dots, \sigma_q \in \Sigma$ such that

$$\tau' = \mathbf{zip}_q(\sigma_1, \dots, \sigma_q).$$

With Lemma 4.5 we now obtain

$$\tau' = \sum_{i=1}^{q} \mathcal{X}^{i-1} \mathbf{zip}_q(\sigma_i, 0, \dots, 0)$$

and Lemma 5.16 now gives

$$\tau' = \sum_{i=1}^{q} \mathcal{X}^{i-1} (\sigma_i)^q.$$

Now it directly follows that for every $\tau \in \Sigma$, τ' is rational over Σ. From Proposition 3.16 it now follows that σ is constructively $\mathbb{F}_q$-algebraic. □

Furthermore, if we combine this result with the (classical) other direction of Christol's theorem, i.e. the result that if a stream is $\mathbb{F}_q$-algebraic, it is q-automatic, we obtain the result from [Fli74]—for finite fields only—that, if a stream is $\mathbb{F}_q$-algebraic, it is constructively $\mathbb{F}_q$-algebraic. Instead of going via diagonals of rational series, as is done in [Fli74], we now obtain the same result taking a route via the q-automatic sequences.

Example 5.18. Returning to Example 5.5, the construction from Proposition 5.17 now yields the system of behavioural differential equations

$$\begin{array}{rclcrcl}
o(x) & = & 0 & \qquad & x' & = & x^2 + \mathcal{X}w^2 \\
o(y) & = & 1 & & y' & = & y^2 + \mathcal{X}z^2 \\
o(z) & = & 1 & & z' & = & z^2 + \mathcal{X}w^2 \\
o(w) & = & 0 & & w' & = & w^2 + \mathcal{X}z^2
\end{array}$$

with x witnessing that the Thue-Morse sequence is $\mathbb{F}_2$-algebraic.

Example 5.19. Likewise, the $\mathbf{zip}_3$-behavioural differential equations from Example 5.6 can be transformed in the following behavioural differential equations, showing that the Cantor sequence is constructively $\mathbb{F}_3$-algebraic

$$\begin{array}{rclcrcl}
o(x) & = & 1 & \qquad & x' & = & y^3 + \mathcal{X}x^3 + \mathcal{X}^2\bar{x}^3 \\
o(y) & = & 0 & & y' & = & y^3 + \mathcal{X}y^3 + \mathcal{X}^2\bar{y}^3 \\
o(\bar{x}) & = & 0 & & \bar{x}' & = & x^3 + \mathcal{X}\bar{x}^3 + \mathcal{X}^2\bar{y}^3 \\
o(\bar{y}) & = & 0 & & \bar{y}' & = & y^3 + \mathcal{X}\bar{y}^3 + \mathcal{X}^2\bar{y}^3
\end{array}$$

6

Term algebras and μ-expressions

In this chapter, we will turn to two more ways of coalgebraically describing the constructively algebraic power series, by means of *term algebras* and by means of *μ-expressions*. The presentations again instantiate to the context-free languages when $\mathbb{B}$ is considered as the underlying semiring, and the presentation in this result essentially can be seen as relying on a syntactic translation of the approach from Chapter 3.

Term algebras, which are treated in Section 6.1 provide a connection between the polynomial systems presented in Chapter 3 and the world of *universal algebra*. As we will see in the next chapter, term algebras also provide a closer connection to the coalgebraic work on bialgebraic semantics and coalgebraic bisimulation up to (see e.g. [Kli11] and [RBR13]).

In Section 6.2, we introduce μ-expressions, and construct S-automata capturing these μ-expressions and their coalgebraic interpretation. The constructions and proofs in this section are more streamlined versions of those presented in [WBR13], which in turn were based on the work in [Mil10], [SBR10], and [Sil10]. This work contrasts with more traditional work (e.g. [ÉL05]) on μ-expressions in formal language theory, which studies least and greatest fixed points in a setting of idempotent semirings and lattices. In contrast, our μ-expressions can be seen as *unique* fixed point expressions over arbitrary semirings, yielding unique solutions.

6.1 Algebraic power series via term algebras

In this section, we introduce term algebras for a signature containing constants for elements of the underlying semiring, two binary operators $\oplus$ and $\otimes$, and an operator . prefixing alphabet symbols to terms. For the remainder of this section, we again fix a finite alphabet A.

Given a semiring S and a finite set X of nonterminals, we let the set $\mathfrak{T}_S(X)$

of terms over X be defined by the following specification in Backus-Naur form:

$$\mathfrak{T}_S(X) \ni t \ ::= \ s \in S \,|\, [x],\ x \in X \,|\, (a.t),\ a \in A \,|\, (t \oplus t) \,|\, (t \otimes t)$$

Examples of terms over $\{x, y\}$ include 1, $(a.[x])$, and $(((a.0) \otimes [x]) \oplus [y])$. We note that these terms are purely syntactic objects, and assume no precedence rules: in order to disambiguate, we always will use parentheses.

Given a finite set X of nonterminals, a *syntactic system of behavioural differential equations* is a coalgebra for the functor $S \times \mathfrak{T}_S(X)^A$, i.e. a system

$$X \xrightarrow{(o, \delta)} S \times \mathfrak{T}_S(X)^A$$

assinging to each nonterminal an output value, and a function from A to $\mathfrak{T}_S(X)$. Following earlier conventions, we commonly again write x_a for $\delta(x)(a)$, and call it the a-derivative of x.

Example 6.1. As an example over the semiring $\mathbb{B}$, consider the syntactic system of behavioural differential equations over the alphabet $\{a, b\}$ and the set of variables $\{x, y\}$ specified by:

$$o(x) = 1 \quad x_a = ([x] \otimes (b.0)) \quad x_b = 0$$

We will soon see that this example corresponds to the polynomial system of behavioural differential equations from Example 3.2, and that x has the language $a^n b^n$ as a solution.

Once again, we can extend any syntactic system of behavioural differential equations (X, o, δ) to an S-automaton

$$(\hat{o}, \hat{\delta}) : \mathfrak{T}_T(X) \to S \times \mathfrak{T}_T(X)^A,$$

by inductively defining the value of the mappings $\hat{o}$ and $\hat{\delta}$ on all terms $t \in \mathfrak{T}_T(X)$ as follows:

t	$\hat{o}(t)$	t_a
$[x],\ x \in X$	$o(x)$	x_a
$s \in S$	s	0
$(b.u),\ b \in A$	0	**if** $b = a$ **then** u **else** 0
$(u \oplus v)$	$\hat{o}(u) + \hat{o}(u)$	$(u_a \oplus v_a)$
$(u \otimes v)$	$\hat{o}(u) \wedge \hat{o}(v)$	$((u_a \otimes v) \oplus (\hat{o}(u) \otimes v_a))$

(6.1)

We can, again, combine the syntactic system (X, o, δ) and the S-automaton $(\mathfrak{T}_S(X), \hat{o}, \hat{\delta})$ in the following commuting diagram, together with the final mapping from $(\mathfrak{T}_S(X), \hat{o}, \hat{\delta})$ to the final S-automaton:

$$\begin{tikzcd}
X \arrow[r, hook, "{[-]}"] \arrow[d, "{(o,\delta)}"'] & \mathfrak{T}_T(X) \arrow[r, dotted, "{[\![-]\!]}"] \arrow[dl, "{(\hat{o},\hat{\delta})}"'] & S\langle\!\langle A \rangle\!\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times \mathfrak{T}_T(X)^A \arrow[rr, dotted, "{1_S \times [\![-]\!]^A}"] & & S \times S\langle\!\langle A \rangle\!\rangle
\end{tikzcd}$$

We again call the composition $[\![[-]]\!] : X \to \mathcal{P}(A^*)$ of the final homomorphism $[\![-]\!]$ with the injection $[-]$ from variables to their corresponding terms the *solution* to the system (X, o, δ).

We will now give a simple transformation from syntactic systems of behavioural differential equations, that is, coalgebras for the functor $S \times \mathfrak{T}_S(-)^A$, into polynomial systems of behavioural differential equations, that is, coalgebras for the functor $S \times S\langle - \rangle^A$.

We first define a mapping $f : \mathfrak{T}_T(X) \to S\langle X \rangle$ assigning terms to their corresponding polynomials by induction on terms, as follows:

$t \in \mathfrak{T}_T(X)$	$f(t)$
$[x],\ x \in X$	x
$s \in S$	s
$(a.u),\ a \in A$	$af(u)$
$(u \oplus v)$	$f(u) + f(v)$
$(u \otimes v)$	$f(u)f(v)$

(6.2)

Note that, because every polynomial can be constructed in a finite number of steps, it directly follows that f is surjective.

Now, suppose we are given a syntactic system of behavioural differential equations

$$X \xrightarrow{(o,\delta)} S \times \mathfrak{T}_S(X)^A.$$

We can associate with this system a polynomial system of behavioural differential equations $(X + A, p, \gamma)$ by specifying:

$y \in Y$	$p(y)$	$\gamma(y,a)$
$x \in X$	$o(x)$	$f(\delta(x,a))$
$b \in A$	0	**if** $b = a$ **then** 1 **else** 0

It is easy to check that f is a morphism of S-automata:

Proposition 6.2. *The mapping f is a morphism between the S-automata $(\mathfrak{T}_T(X), \hat{o}, \hat{\delta})$ and $(S\langle X+A\rangle, \hat{p}, \hat{\gamma})$ or, in other words, the diagram*

$$\begin{tikzcd}
\mathfrak{T}_T(X) \arrow[r, "f"] \arrow[d, "(\hat{o},\hat{\delta})"'] & S\langle X+A\rangle \arrow[d, "(\hat{p},\hat{\gamma})"] \\
S \times \mathfrak{T}_T(X) \arrow[r, "1_S \times f^A"] & S \times S\langle X+A\rangle^A
\end{tikzcd}$$

commutes.

Proof. We need to check that for all $t \in \mathfrak{T}_T(X)$

$$\hat{o}(t) = \hat{p}(f(t)), \tag{6.3}$$

and for all $\sigma \in \mathfrak{T}_T(X)$ and $a \in A$

$$f(t_a) = \hat{\gamma}(f(t), a) \tag{6.4}$$

(for clarity writing the derivative $\hat{\gamma}$ explicitly while using subscript notation for $\hat{\delta}$). We do this by structural induction on terms.

- If $t = [x]$ for some $x \in X$, observe

  $$\hat{o}([x]) = o(x) = p(x) = \hat{p}(x) = \hat{p}(f([x]))$$

  and

  $$f([x]_a) = f(x_a) = \gamma(x,a) = \hat{\gamma}(x,a) = \hat{\gamma}(f([x]),a).$$

- If $t = s$ for some $s \in S$, observe

  $$\hat{o}(s) = s = \hat{p}(s) = \hat{p}(f(s))$$

  and

  $$f(s_a) = f(0) = 0 = \hat{\gamma}(s,a) = \hat{\gamma}(f(s),a).$$

- If $t = (b.u)$ for some $b \in A$ and $u \in \mathfrak{T}_S(X)$, use the inductive hypothesis that (6.3) and (6.4) for u and observe
$$\hat{o}(b.u) = 0 = p(b)\hat{p}(f(u)) = \hat{p}(bf(u)) = \hat{p}(f(b.u)).$$
and
$$\begin{aligned} f((b.u)_a) &= f(\textbf{if } b = a \textbf{ then } u \textbf{ else } 0) \\ &= \textbf{if } b = a \textbf{ then } f(u) \textbf{ else } 0) \\ &= \gamma(b,a)f(u) + p(b)\hat{\gamma}(f(u),a) \\ &= \hat{\gamma}(bf(u),a) \\ &= \hat{\gamma}(f(b.u),a). \end{aligned}$$

- If $t = (u \oplus v)$ for some u and v, use the inductive hypothesis that (6.3) and (6.4) hold for u and v. Observe
$$\hat{o}(u \oplus v) = \hat{o}(u) + \hat{o}(v) = \hat{p}(f(u)) + \hat{p}(f(v)) = \hat{p}(f(u) + f(v)) = \hat{p}(f(u \oplus v))$$
and
$$\begin{aligned} f((u \oplus v)_a) &= f(u_a \oplus v_a) \\ &= f(u_a) + f(v_a) \\ &= \hat{\gamma}(f(u),a) + \hat{\gamma}(f(v),a) \\ &= \hat{\gamma}(f(u \oplus v),a). \end{aligned}$$

- If $t = (u \otimes v)$ for some u and v, use the inductive hypothesis that (6.3) and (6.4) hold for u and v. Observe
$$\hat{o}(u \otimes v) = \hat{o}(u)\hat{o}(v) = \hat{p}(f(u))\hat{p}(f(v)) = \hat{p}(f(u)f(v)) = \hat{p}(f(u \otimes v))$$
and
$$\begin{aligned} f((u \otimes v)_a) &= f(((u_a \otimes v) \oplus (\hat{o}(u) \otimes v_a))) \\ &= f(u_a)f(v) + f(\hat{o}(u))f(v_a) \\ &= \hat{\gamma}(f(u),a)f(v) + \hat{p}(f(u))\hat{\gamma}(v,a) \\ &= \hat{\gamma}(f(u)f(v),a) \\ &= \hat{\gamma}(f(u \otimes v),a). \end{aligned}$$
$\square$

Denoting the unique mapping from $(\mathfrak{T}_T(X), \hat{o}, \hat{\delta})$ into the final automaton with $[\![-]\!]$ and denoting the unique mapping from $(S\langle X+A\rangle, \hat{p}, \hat{\gamma})$ with $[\![-]\!]_1$, we get

$$[\![-]\!] = [\![-]\!]_1 \circ f$$

because of unicity and thus, because $[\![-]\!]_1$ is a semiring morphism, we also get

$$\begin{aligned}[\![(s \oplus t)]\!] &= [\![s]\!] + [\![t]\!] \\ [\![(s \otimes t)]\!] &= [\![s]\!][\![t]\!].\end{aligned}$$

Thus, the syntactic operators $\oplus$ and $\otimes$ are interpreted as the semiring operations $+$ and $\cdot$ of $S\langle X+A\rangle$ and of $S\langle\!\langle A\rangle\!\rangle$.

The equality $[\![-]\!] = [\![-]\!]_1 \circ f$, together with the surjectivity of f, also gives us the following result:

Proposition 6.3. *A power series $\sigma \in S\langle\!\langle A\rangle\!\rangle$ is constructively S-algebraic if and only if there is a finite syntactic system (X, o, δ) and a term $t \in \mathfrak{T}_T(X)$ such that $[\![t]\!] = \sigma$ w.r.t. the extension $(\mathfrak{T}_T(X), \hat{o}, \hat{\delta})$.*

Proof. If there is such a system, then $f(t)$ yields a polynomial in the associated system, and by $[\![f(t)]\!]_1 = [\![t]\!] = \sigma$ and Proposition 3.6, it now follows that σ is constructively algebraic.

Conversely, if σ is constructively S-algebraic, there is a polynomial system of behavioural differential equations (X, p^-, γ^-) and a polynomial $q \in S\langle X\rangle$ such that $[\![t]\!] = \sigma$. We can extend this system to another polynomial system $(X+A, p, \gamma)$ and because f is surjective, we can construct some corresponding syntactic system (X, o, δ) that yields the system $(X+A, p, \gamma)$ as its associated polynomial system. Now take some $t \in \mathfrak{T}_S(X)$ with $f(t) = q$ and it follows that again $[\![t]\!] = \sigma$. $\square$

We will finish this section with a simple and well-known result (which can also be derived from the categorical framework that we will present in the next chapter), namely that the bisimilarity relation on any automaton is a congruence.

Lemma 6.4. *The bisimilarity relation $\sim$ on any S-automaton $(\mathfrak{T}_S(X), \hat{o}, \hat{\delta})$ is a congruence, in other words, if $s \sim t$ and $u \sim v$, then also $(s \oplus u) \sim (t \oplus v)$ and $(s \otimes u) \sim (t \otimes v)$.*

Proof. Recall that for any s, t, $s \sim t$ if and only if $[\![s]\!] \sim [\![t]\!]$. Now observe

$$[\![(s \oplus u)]\!] = [\![s]\!] + [\![u]\!] = [\![t]\!] + [\![v]\!] = [\![(t \oplus v)]\!]$$

and thus $(s \oplus u) \sim (t \oplus v)$. The case of $\otimes$ goes in the same way. $\square$

6.2 Algebraic power series via μ-expressions

We define the set of terms t (henceforth to be called *μ-expressions*) and guarded terms g over a semiring S, an alphabet A and an arbitrary infinite set Z of variables as follows:

$$\begin{array}{rcl} \mathfrak{T}_\mu^- \ni t & ::= & s \in S \,|\, x \in Z \,|\, (a.t),\ a \in A \,|\, (t \oplus t) \,|\, (t \otimes t) \,|\, \mu x.g \\ \mathfrak{T}_{\gamma\mu}^- \ni g & ::= & s \in S \,|\, (a.t),\ a \in A \,|\, (g \oplus g) \end{array}$$

We now let $\mathfrak{T}_\mu$ denote the set of all closed μ-expressions, and $\mathfrak{T}_\mu^-$ the set of all μ-expressions. Similarly, we let $\mathfrak{T}_{\gamma\mu}$ denote the set of closed and guarded μ-expressions, and $\mathfrak{T}_{\gamma\mu}^-$ the set of all guarded μ-expressions.

Note that, as a result of the specification, every guarded μ-expression also is a μ-expression (but not vice versa), and thus $\mathfrak{T}_{\gamma\mu} \subset \mathfrak{T}_\mu$ and also $\mathfrak{T}_{\gamma\mu}^- \subset \mathfrak{T}_\mu^-$.

The notions of *free* and *bound* variables and of *subexpression* can now be defined in the usual way. Given expressions t and u and a variable x, let

$$t[u/x]$$

the expression obtained by replacing all free occurrences of x in t with the expression u.

Moreover, define a *pruning* operator $\mathbf{pr} : \mathfrak{T}_\mu \to \mathfrak{T}(Z)$ as follows:

$$\begin{array}{rcl} \mathbf{pr}(s) & = & s \\ \mathbf{pr}(x) & = & x \\ \mathbf{pr}(a.t) & = & (a.\mathbf{pr}(t)) \\ \mathbf{pr}(t \oplus u) & = & (\mathbf{pr}(t) \oplus \mathbf{pr}(u)) \\ \mathbf{pr}(t \otimes u) & = & (\mathbf{pr}(t) \otimes \mathbf{pr}(u)) \\ \mathbf{pr}(\mu x.g) & = & x \end{array}$$

Finally, given any μ-expression t, let $\mathbf{vars}(t) \in Z$ be the set of variables that occur in $\mathbf{pr}(t)$.

We will, in order to avoid considerations related to α-renaming, restrict ourselves to μ-expressions that have the followng property: we call a μ-expression t *nice* whenever the following two conditions hold:

1. If t has subexpressions $\mu x.u$ and $\mu x.v$ for some variable $x \in Z$, then $\mathbf{pr}(u) = \mathbf{pr}(v)$.

2. If t has a subexpression $\mu x.u$, then u has no subexpressions of the form $\mu x.v$.

For example, the expression

$$\mathbf{t} := (\mu x.((a.\mu y.(b.x))) \oplus \mu y.((b.\mu x.(a.y)))) \tag{6.5}$$

is nice (every subexpression bound by x has the pruning $(a.y)$ and every subexpression bound by y has the pruning $(b.x)$, but the expression

$$\mathbf{u} := (\mu x.((a.\mu y.(b.x))) \oplus \mu y.((b.y)))$$

is not, as y is bound by expressions with prunings $(b \oplus x)$ and $(b \oplus y)$.

If t has no two subexpressions binding the same variable, it is clear that these two conditions hold. We can therefore, using α renaming, transform any μ-expression into an equivalent μ-expression that is nice.

If t is nice and closed, we can define a unique function

$$\xi_t : \mathbf{vars}(t) \to \mathfrak{T}_\gamma(\mathbf{vars}(t))$$

assigning to each variable in t the unique pruning of the μ-expressions bound by it.

In the earlier example from 6.5, note that $\xi_\mathbf{t}$ can be specified as

$$\begin{aligned} \xi_\mathbf{t}(x) &= (a.y) \\ \xi_\mathbf{t}(y) &= (b.x) \end{aligned}$$

Given any $X \subseteq Z$ and any assignment $\zeta : X \to \mathfrak{T}_\gamma(X)$, we can now gather the set of all closed μ-expressions that yield an assignment which is a subset of ξ, as follows:

$$\mathfrak{T}_\mu^\zeta = \{u \,|\, u \text{ is nice and } \xi_u \subseteq \zeta\}$$

We next define the closure $\mathbf{cl}_\zeta$ with respect to some set of assignment ξ. Let $\mathbf{cl}_\zeta(t)$ be the unique closed μ-expression that can be obtained by repeated substitutions of the form $[\zeta(x)/x]$.

We have a bijection

$$\begin{tikzcd} \mathfrak{T}(\mathbf{dom}(\zeta)) \arrow[r, shift left, "\mathbf{cl}_\zeta"] & \mathfrak{T}_\mu^\zeta \arrow[l, shift left, "\mathbf{pr}"] \end{tikzcd}$$

And now can assign an automaton (o^ζ, δ^ζ) on $\mathfrak{T}^\zeta_\mu$ as follows:

$t \in \mathfrak{T}^\zeta_\mu$	$o^\zeta(t) \in S$	$\delta^\zeta(t, a) \in \mathfrak{T}^\zeta_\mu$
$s \in S$	s	0
$(b.u),\ b \in A$	0	**if** $b = a$ **then** u **else** 0
$(u \oplus v)$	$o(u) + o(v)$	$(u_a \oplus v_a)$
$(u \otimes v)$	$o(u)o(v)$	$((u_a \otimes v) \oplus (o(u) \otimes v_a))$
$\mu x.u$	$o(u)$	$\mathbf{cl}(u_a)$

(6.6)

or, in other words, a $S \times -^A$-coalgebra:

$$\mathfrak{T}^\zeta_\mu \xrightarrow{(o^\zeta, \delta^\zeta)} S \times (\mathfrak{T}^\zeta_\mu)^A$$

6.2.1 An isomorphism between μ-expressions and term algebras

For this section, assume an assignment ζ and let $X \subseteq (Z)$ denote $\mathbf{dom}(\zeta)$.

$$\begin{tikzcd}
X \arrow[r, hook, "{[-]}"] & \mathfrak{T}(X) \arrow[r, shift left, "\mathbf{cl}_\zeta"] & \mathfrak{T}^\zeta_\mu \arrow[l, shift left, "\mathbf{pr}"] \arrow[d, "{(o^\zeta, \delta^\zeta)}"] \\
& S \times \mathfrak{T}(X)^A \arrow[r, shift left, "1_S \times (\mathbf{cl}_\zeta)^A"] & S \times (\mathfrak{T}^\zeta_\mu)^A \arrow[l, shift left, "1_S \times \mathbf{pr}^A"]
\end{tikzcd}$$

We now can use this diagram to construct a syntactic system of behavioural differential equations (o, δ) on X, by specifying

$$o = o^\zeta \circ \mathbf{cl}_\zeta \circ [-]$$

and

$$\delta = \mathbf{pr}^A \circ \delta^\zeta \circ \mathbf{cl}_\zeta \circ [-].$$

As a direct consequence of this definition and the bijection between $\mathbf{cl}_\zeta$ and

$\mathbf{pr}$, the diagram

$$\begin{tikzcd}
X \arrow[r, hook, "{[-]}"] \arrow[d, "{(o,\delta)}"'] & \mathfrak{T}(X) \arrow[r, "\mathbf{cl}_\zeta"] & \mathfrak{T}^\zeta_\mu \arrow[d, "{(o^\zeta,\delta^\zeta)}"] \\
S \times \mathfrak{T}(X)^A \arrow[rr, "1_S \times (\mathbf{cl}_\zeta)^A"] & & S \times (\mathfrak{T}^\zeta_\mu)^A
\end{tikzcd}$$

again commutes.

Note the following identities:

$$\begin{aligned}
\mathbf{cl}(s \oplus t) &= (\mathbf{cl}(s) \oplus \mathbf{cl}(t)) \\
\mathbf{cl}(a.t) &= (a.\mathbf{cl}(t)) \\
\mathbf{cl}(s \otimes t) &= (\mathbf{cl}(s) \oplus \mathbf{cl}(t))
\end{aligned}$$

Proposition 6.5. *The function $\mathbf{cl}_\zeta$ is an isomorphism of S-automata between $(\mathfrak{T}(X), \hat{o}, \hat{\delta})$ and $(\mathfrak{T}^\zeta_\mu, o^\zeta, \delta^\zeta)$, or in other words, the diagram*

$$\begin{tikzcd}
\mathfrak{T}(X) \arrow[r, shift left, "\mathbf{cl}_\zeta"] \arrow[d, "{(\hat{o},\hat{\delta})}"'] & \mathfrak{T}^\zeta_\mu \arrow[l, shift left, "\mathbf{pr}"] \arrow[d, "{(o^\zeta,\delta^\zeta)}"] \\
S \times \mathfrak{T}(X)^A \arrow[r, shift left, "1_S \times (\mathbf{cl}_\zeta)^A"] & S \times (\mathfrak{T}^\zeta_\mu)^A \arrow[l, shift left, "1_S \times \mathbf{pr}^A"]
\end{tikzcd}$$

commutes. Hence, for any $t \in \mathfrak{T}^\zeta_\mu$, $[\![t]\!]$ is constructively S-algebraic.

Proof. We know that $\mathbf{cl}_\zeta$ is a bijection, so it suffices to show that $\mathbf{cl}_\zeta$ is a morphism of S-automata. This is done by structural induction on terms, in combination with reading off the specifications (6.1) and (6.6). Hence $[\![t]\!] = [\![\mathbf{pr}(t)]\!]$ for all $t \in \mathfrak{T}^\zeta_\mu$, and by Proposition 6.3 it now follows that t is constructively S-algebraic. □

6.2.2 From terms to μ-expressions

For the converse direction, first need to define a syntactic analogue of the indexed summation operator $\sum$, denoted by

$$\bigoplus_{i=1}^{k} t_i$$

and defined as follows: If $k = 1$, then

$$\bigoplus_{i=1}^{1} t_i = t_1$$

and if $k > 1$, then

$$\bigoplus_{i=1}^{k} t_i = \left(\bigoplus_{i=1}^{k-1} t_i \oplus t_k \right).$$

Given a system (X, o, δ) on a finite set $X \subseteq Z$, now consider the assignment

$$\zeta(x) = \left(o(x) \oplus \bigoplus_{i=1}^{k} (a_i.x_a) \right).$$

This assignment leads to another system (X, p, γ). Observe that $o = p$.

We now define the *zero-closure* **zcl** of a term $t \in \mathfrak{T}(X)$ as the smallest set of terms containing t, such that, whenever $u \in \mathbf{zcl}(x)$, then also $(u \oplus 0) \in \mathbf{zcl}(x)$ and $(0 \oplus u) \in \mathbf{zcl}(x)$. We can describe **zcl** using Backus Naur notation as

$$\mathbf{zcl}(t) \ni u \quad ::= \quad t \,|\, (u \oplus 0) \,|\, (0 \oplus u)$$

or equivalently in set-theoretic notation:

$$\mathbf{zcl}(t) := \bigcap \{ U \subseteq \mathfrak{T}(X) \,|\, t \in U \wedge \forall u \in \mathfrak{T}(X)(u \in U \to \{(u \oplus 0), (0 \oplus u)\} \subseteq U) \}$$

Lemma 6.6. *The relation $\{(t, u) \,|\, u \in \mathbf{zcl}(t)\}$ is a bisimulation on any S-automaton $(\mathfrak{T}(X), \hat{o}, \hat{\delta})$ obtained from a syntactic system of behavioural differential equations (X, o, δ).*

Proof. It is easily seen (e.g. by induction) that for all $u \in \mathbf{zcl}(t)$, $\hat{o}(u) = \hat{o}(t)$ and that $u_a \in \mathbf{zcl}(t_a)$. $\square$

Proposition 6.7. *The identity relation $R := \{(t,t) \,|\, t \in \mathfrak{T}(X)\}$ is a bisimulation up to bisimilarity between (X, o, δ) and (X, p, γ).*

Proof. We again proceed by structural induction on terms. Because $o = p$ and thus $\hat{o} = \hat{p}$, we only have to check the derivative cases.

- If $t = [x]$ for some $x \in X$, it follows that $\gamma(x, a) \in \mathbf{zcl}(t_a)$, so $t_a \sim \hat{\gamma}(x, a)$ and thus we have
$$[x]_a \; R \; [x]_a \sim \hat{\gamma}([x], a).$$
- If $t = s$ for some $s \in S$, it follows that
$$s_a = 0 \; R \; 0 = \hat{\gamma}(s, a).$$
- If $t = (b.u)$ for some $b \in A$ and $u \in \mathfrak{T}(X)$, observe
$$(b.u)_a \; R \; (b.u)_a = \hat{\gamma}((b.u), a).$$
- If $t = (u \oplus v)$ for terms u and v, use the inductive hypothesis that $u_a \sim \hat{\gamma}(u, a)$ and $v_a \sim \hat{\gamma}(v, a)$. Because of Proposition 6.4, it now follows that
$$(u_a \oplus v_a) \sim (\hat{\gamma}(u, a) \oplus \hat{\gamma}(v, a))$$
and thus
$$(u \oplus v)_a = (u_a \oplus v_a) \; R \; (u_a \oplus v_a) \sim (\hat{\gamma}(u, a) \oplus \hat{\gamma}(v, a)) = \hat{\gamma}((u \oplus v), a).$$
- The case where $t = (u \otimes v)$ is entirely analogous to the previous case. □

Now, finally, we have established the identity between power series characterizable by μ-expressions and the constructively S-algebraic series:

Theorem 6.8. *A formal power series $\sigma \in S\langle\langle A \rangle\rangle$ is constructively S-algebraic if and only if there is a nice and closed μ-expression $t \in \mathfrak{T}_\mu$ such that $[\![t]\!] = \sigma$ with respect to the automaton $(\mathfrak{T}_\mu^{\xi_t}, o^{\xi_t}, \delta^{\xi_t})$.*

Proof. If σ is constructively S-algebraic, there is a by Proposition 6.3 there is a syntactic system of behavioural differential equations (X, o, δ) and a term $t \in \mathfrak{T}(X)$ such that $[\![t]\!] = \sigma$. Using the main construction from this subsection, we now obtain a new system (X, p, γ) and an assignment ζ such that $[\![\mathbf{cl}_\zeta(t)]\!] = \sigma$ w.r.t. $(\mathfrak{T}_\mu^{\xi_t}, o^{\xi_t}, \delta^{\xi_t})$.

The converse case is a part of the statement of Proposition 6.5. □

7

Distributive laws and bialgebras

We will, in this chapter, explore the connections between the concrete coalgebraic presentations of the rational and constructively algebraic power series from Chapters 2 and 3, and the more abstract, categorical, framework of *distributive laws* and *bialgebras.*

In Section 7.1, we first give the relevant definitions of bialgebras and distributive laws, followed by stating the relevant results from this framework. Comprehensive introductions to the framework, providing a more general context, can be found in e.g. [Bar04], [Jac06], and [Kli11]. We also describe the *generalized powerset construction* from [SBBR10] and [SBBR13] as a generalized method of obtaining diagrams similar to (2.3) and (3.1.2) as well as lemmata similar to Lemma 2.15 and Lemma 3.5. The section is concluded with a brief survey of the case where T is an arbitrary monad and F is the functor for S-automata for some set S, and propose generalizations of the concepts of rationality, constructive algebraicity, and k-regularity, allowing us to talk about λ-algebraic elements of $S\langle\langle A\rangle\rangle$, and λ, k-algebraic streams in $S^{\mathbb{N}}$.

Several examples, as well as a striking counterexample, to this general framework, are presented in Section 7.2. On one hand, it turns out that, whereas weighted and nondeterministic automata can, without the occurrence of any problems, be seen as an instance of this framework (as first observed in [Bar04]), the same does not hold—at least not directly—for the polynomial systems from Chapter 3. More specifically, this case fails to be a direct instance of a distributive law, as a result the absence of a suitable algebra structure on $S \times S\langle X\rangle^A$. One way in which the context-free languages (with possible generalization to constructively algebraic power series) can be reconciled with the framework of distributive laws, is by considering a co-pointed functor: this approach has been considered in [BHKR13].

Finally, in Section 7.3, we propose *Brzozowski bialgebras* as another method of dealing with the encountered problems and counterexamples, alternative to the options of altering the product rule and using co-pointed functors. We end with the observation that, albeit with a different proof than the purely abstract proof for λ-bialgebras, the crucial lemma still holds at the same level of generality. Furthermore, it turns out that categories of Brzozowski bialgebras share many similarities with categories of λ-bialgebras.

7.1 Distributive laws and λ-bialgebras

We will briefly summarize the main relevant constructions from [Bar04], [Jac06], [Kli11], and [SBBR10].

Given a monad (T, μ, η) and an endofunctor F on any category $\mathbf{C}$, a *distributive law* of the monad T over F is a natural transformation

$$\lambda : TF \Rightarrow FT$$

such that the two diagrams of natural transformations

$$\begin{tikzcd}
F \arrow[r, Rightarrow, "\eta F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[d, Rightarrow, "\lambda"] \\
 & FT
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
TTF \arrow[rr, Rightarrow, "\mu F"] \arrow[d, Rightarrow, "T\lambda"'] & & TF \arrow[d, Rightarrow, "\lambda"] \\
TFT \arrow[r, Rightarrow, "\lambda T"] & FTT \arrow[r, Rightarrow, "F\mu"] & FTT
\end{tikzcd}$$

commute.

Furthermore, given a distributive law $\lambda : TF \Rightarrow FT$, λ-*bialgebra* (X, α, γ) consists of a coalgebra (X, γ) for the functor F, an algebra (X, α) for the monad T, such that the diagram

$$\begin{tikzcd}
TX \arrow[r, "\alpha"] \arrow[d, "T\gamma"'] & X \arrow[r, "\gamma"] & FX \\
TFX \arrow[rr, "\lambda_X"] & & FTX \arrow[u, "F\alpha"']
\end{tikzcd}$$

commutes.

We turn our attention to a number of elementary results about distributive laws, found e.g. in Lemmata 3.2.5 and 3.4.21 of [Bar04]:

Proposition 7.1. *Let $\lambda : TF \Rightarrow FT$ be a distributive law. We have the following:*

1. *Given any algebra (X, α) for the monad T, the T-algebra $(FX, F\alpha \circ \lambda_X)$ is again an algebra for the monad T.*

2. *Given any λ-bialgebra (X, α, γ), γ is an algebra morphism between (X, α) and $(FX, F\alpha \circ \lambda_X)$.*

3. *Given algebras (X, α) and (Y, β) for the monad T and any algebra morphism $f : (X, \alpha) \to (Y, \beta)$, $Ff : (FX, F\alpha \circ \lambda_X) \to (FY, F\beta \circ \lambda_Y)$ is again an algebra morphism.*

Proof. 1. follows from the fact that the diagrams

$$\begin{tikzcd}
FX \arrow[r, "\mathrm{id}_{FX}"] \arrow[d, "\eta_{FX}"'] \arrow[dr, "F\eta_X"] & FX \\
TFX \arrow[r, "\lambda_X"] & FTX \arrow[u, "F\alpha"']
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
TTFX \arrow[rr, "\mu_{FX}"] \arrow[d, "T\lambda_X"'] & & TFX \arrow[d, "\lambda_X"] \\
TFTX \arrow[r, "\lambda_{TX}"] \arrow[d, "TF\alpha"'] & FTTX \arrow[r, "F\mu_X"] \arrow[d, "FT\alpha"'] & FTX \arrow[d, "F\alpha"] \\
TFX \arrow[r, "\lambda_X"] & FTX \arrow[r, "F\alpha"] & FX
\end{tikzcd}$$

commute.

2. is a direct consequence of the defining diagram of λ-bialgebras, and

3. follows from the following commuting diagram:

$$\begin{tikzcd}
TFX \arrow[r, "\lambda_X"] \arrow[d, "TFf"'] & FTX \arrow[r, "F\alpha"] \arrow[d, "FTf"] & TX \arrow[d, "Tf"] \\
TFY \arrow[r, "\lambda_Y"] & FTY \arrow[r, "F\beta"] & TY
\end{tikzcd}$$

□

Instantiating the free algebra (TX, μ_X) in this result, we thus get, for any FT-coalgebra δ, a unique T-algebra mapping $\hat{\delta}$ extending δ as follows:

$$\begin{tikzcd}
X \arrow[r, hook, "\eta_X"] \arrow[d, "\delta"'] & TX \arrow[dl, dotted, "\hat{\delta}"'] \\
FTX &
\end{tikzcd}$$

This leads us to the following categorical generalization of Lemma 2.15, which is essentially a reformulation of [Jac06, Lemma 2]:

Lemma 7.2. *Given a distributive law λ of a monad (T, μ, η) over an endofunctor F, a λ-bialgebra (Q, α, γ) and any FT-coalgebra (X, δ), if $f : X \to Q$ makes the diagram*

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\delta"'] & Q \arrow[d, "\gamma"] \\
FTX \arrow[r, "F\hat{f}"] & FQ
\end{tikzcd}$$

commute, then the unique T-algebra morphism $\hat{f} : TX \to Q$ extending f makes the diagram

$$\begin{tikzcd}
X \arrow[r, hook, "\eta_X"] \arrow[d, "\delta"'] & TX \arrow[r, "\hat{f}"] \arrow[dl, "\hat{\delta}"'] & Q \arrow[d, "\gamma"] \\
FTX \arrow[rr, "F\hat{f}"] & & FQ
\end{tikzcd}$$

commute.

Proof. Identical to the proof of Lemma 2.15 after making the following substitutions:

$(\hat{o},\hat{\delta})$	$\Rightarrow$	$\hat{\delta}$
(p,γ)	$\Rightarrow$	γ
$1_S \times f^A$	$\Rightarrow$	Ff
$\mathrm{Lin}_S(X)$	$\Rightarrow$	TX
S-linear mapping	$\Rightarrow$	F-algebra morphism
S-linear automaton	$\Rightarrow$	λ-bialgebra

□

When F has a final coalgebra (Ω,ω), as an instance of this diagram we thus obtain, as shown in [SBBR10] and [SBBR13], the following diagram for any FT-coalgebra (X,δ)

$$\begin{tikzcd}
X \arrow[r, hook, "\eta_X"] \arrow[d, "\delta"'] & TX \arrow[r, dotted, "{[\![-]\!]}"] \arrow[dl, "\hat{\delta}"'] & \Omega \arrow[d, "\omega"] \\
FTX \arrow[rr, dotted, "{F[\![-]\!]}"] & & F\Omega
\end{tikzcd}$$

yielding a notion of final coalgebra semantics for λ-bialgebras via what is called the *generalized powerset construction.*

7.1.1 Distributive laws over automata

Let us now restrict ourselves to the case where F is of the form $S\times -^A$, i.e. where the F-coalgebras are S-automata, while letting T be an arbitrary monad on **Set**. We will call such a distributive law a *distributive law of the monad T over S-automata.*

We call a $\sigma \in S\langle\langle A\rangle\rangle$ *T-describable* in some finite set $\Sigma \subseteq S\langle\langle A\rangle\rangle$ whenever there is some $t \in T(\Sigma)$ such that $\alpha \circ T\iota = \sigma$ (letting ι again denote the inclusion of Σ into $S\langle\langle A\rangle\rangle$, and letting α be the algebra structure of the final λ-bialgebra on $S\langle\langle A\rangle\rangle$).

If we additionally say that any $\sigma \in S\langle\langle A\rangle\rangle$ is *λ-algebraic* if and only if there is a finite $S\times T(-)^A$-coalgebra (X,o,δ) and some $x \in X$ such that $[\![\eta_X(x)]\!] = \sigma$, we again obtain the following result thanks to Lemma 7.2:

Proposition 7.3. *Given a distributive law λ of a monad T over S-automata, an S-language partition $\sigma \in S\langle\langle A\rangle\rangle$ is λ-algebraic if and only if there is a finite set*

$\Sigma \subseteq S\langle\langle A\rangle\rangle$ with $\sigma \in \Sigma$, such that for each $\tau \in \Sigma$ and $a \in A$, τ_a is T-describable in Σ.

Proof. Like Proposition 2.16 and Proposition 3.6. □

Using the isomorphism between $S\langle\langle A_k\rangle\rangle$ and $S^{\mathbb{N}}$ described in Chapter 5, we can furthermore generalize the notions of k-automaticity and k-regularity as follows: we call a $\sigma \in S^{\mathbb{N}}$ λ, k-algebraic whenever $\nu_k \circ \sigma \in S\langle\langle A\rangle\rangle$ is λ-algebraic. This definition again directly leads to a result corresponding to Corollary 5.9.

Furthermore, we have the following elementary but general proposition about the S-simple language partitions and k-automatic sequences:

Proposition 7.4. *Let λ be a distributive law of a monad T over S-automata. If $\sigma \in S\langle\langle A\rangle\rangle$ is S-simple, then σ is λ-algebraic. Likewise, if $\sigma \in S^{\mathbb{N}}$ is k-automatic, then σ is λ, k-algebraic.*

Proof. If σ is S-simple, then there is a finite automaton (X, δ^-) and some $x \in X$ with $[\![x]\!] = \sigma$. Instantiate the antecedent diagram from Lemma 7.2 with $\delta = T\delta^- \circ \eta_X$ for δ, $[\![-]\!]$ for f, the final S-automaton for (Q, γ), and $F = S \times -^A$. It now directly follows from Lemma 7.2 that σ is S, λ-algebraic. The case for k-automatic sequences goes analogously via the isomorphism from Chapter 5. □

7.2 Examples of distributive laws

7.2.1 Weighted automata

We now turn to the observation that the functor $\mathrm{Lin}_S(-)$ is in fact a monad. Recalling that every element $t \in \mathrm{Lin}_S(X)$ can be written as

$$\sum_{i=1}^{n} s_i x_i$$

in some way using the S-module structure of $\mathrm{Lin}_S(X)$, μ_X can be defined as

$$\mu_X \left[\sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij} x_{ij}\right]\right] = \left[\sum_{i=1}^{n} \sum_{j=1}^{m_i} s_i t_{ij} x_{ij}\right]$$

together with η_X mapping any x to the linear combination $[x]$, and checking that this indeed is a monad is easy. For example, checking that the multiplication law for monads holds amounts to showing that, in the below expression, removing the outer brackets first and then the inner brackets yields the same result as removing the inner brackets first and then the outer brackets:

$$\sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij} \left[\sum_{k=1}^{l_{ij}} u_{ijk} x_{ijk}\right]\right]$$

Furthermore, the category of $\mathrm{Lin}_S(-)$-algebras is the same as the category of S-modules.

The distributive law

$$\lambda : \mathrm{Lin}_S(-)(S \times -^A) \Rightarrow (S \times -^A)\mathrm{Lin}_S(-)$$

can be given componentwise by

$$\lambda_X \left(\sum_{i=1}^{n} s_i(o_i, d_i)\right) = \left(\sum_{i=1}^{n} s_i o_i, a \mapsto \sum_{i=1}^{n} s_i d_i(a)\right).$$

We verify that this natural transformation indeed is a distributive law equationally:

$$\begin{aligned}
&(1_S \times (\lambda_X)^A) \circ \eta^1_{S \times X^A}(o, d) \\
&= (1_S \times (\lambda_X)^A)[(o, d)] \\
&= (o, a \mapsto \eta_X(d(a))) \\
&= (1_S \times (\eta_X)^A)(o, d)
\end{aligned}$$

$$\begin{aligned}
&\lambda_X \circ \mu^1_{S \times X^A} \left(\sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij}(o_{ij}, d_{ij})\right]\right) \\
&= \lambda_X \left(\sum_{i=1}^{n}\sum_{j=1}^{m_i} s_i t_{ij}(o_{ij}, d_{ij})\right) \\
&= \left(\sum_{i=1}^{n}\sum_{j=1}^{m_i} s_i t_{ij} o_{ij}, a \mapsto \sum_{i=1}^{n}\sum_{j=1}^{m_i} s_i t_{ij} d_{ij}(a)\right) \\
&= (S \times (\mu^1_X)^A) \left(\sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij} o_{ij}\right], a \mapsto \sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij} d_{ij}(a))\right]\right)
\end{aligned}$$

$$= (S \times (\mu^{\mathrm{l}}_X)^A) \circ \lambda_{\mathrm{Lin}_S(X)} \left(\sum_{i=1}^{n} s_i \left(\sum_{j=1}^{m_i} t_{ij} o_{ij}, a \mapsto \sum_{j=1}^{m_i} t_{ij} d_{ij}(a)) \right) \right)$$

$$= (S \times (\mu^{\mathrm{l}}_X)^A) \circ \lambda_{\mathrm{Lin}_S(X)} \circ \mathrm{Lin}_S(\lambda_X) \left(\sum_{i=1}^{n} s_i \left[\sum_{j=1}^{m_i} t_{ij} (o_{ij}, d_{ij}) \right] \right)$$

It is now easily verified that the λ-bialgebras become precisely the S-linear automata, whereas the FT-coalgebras instantiate as S-weighted automata. As a consequence, Lemma 2.15 is indeed an instance of Lemma 7.2, as S-linear automata are λ-bialgebras for the above distributive law.

7.2.2 Polynomial systems

Having the knowledge that weighted automata are indeed an instance of the framework of bialgebras, it seems natural to expect that the same fact holds for the presentation of Chapter 3 of context-free languages and constructively algebraic power series.

If S is commutative, we can indeed assign the structure of a monad to the functor $S\langle - \rangle$. Like in the case of linear combinations, the multiplication of the monad can be given by (intuitively) removing brackets from polynomials of polynomials, applying the distribution of multiplication over addition, and bringing the coefficients in the underlying semiring to the front, resulting in new polynomials. Note that, in order to be able to (meaningfully) bring the coefficients to the front, it is again required that S is commutative.

However, we are faced with additional hurdles, directly resulting from the fact that the derivatives $\delta(-, a)$ to alphabet symbols are in fact *not* semiring morphisms.

Indeed, we can see the presentation in Chapter 3, again, in terms of the monad $S\langle - \rangle$ providing the algebraic structure, and the functor $S \times -^A$ providing the coalgebraic structure.

Counterexample 7.5. Consider the following system:

$$\begin{tikzcd}
\varnothing \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "{(o,\delta)}"'] & S\langle \varnothing \rangle \arrow[r, "f^\sharp"] \arrow[dl, "{(\hat{o},\hat{\delta})}"'] & S\langle\langle A \rangle\rangle \arrow[d, "{(O,\Delta)}"] \\
S \times S\langle \varnothing \rangle^A \arrow[rr, "1_S \times (f^\sharp)^A"] & & S \times S\langle\langle A \rangle\rangle^A
\end{tikzcd}$$

We have $S\langle\varnothing\rangle = S$, and thus observe that $S \times S^A$ contains the pair $(1, b \mapsto \textbf{if } b = a \textbf{ then } 1 \textbf{ else } 0)$ which is, via the composition

$$(O, \Delta)^{-1} \circ (1_S \times (f^\sharp)^A),$$

mapped onto the power series a, but, at the same time, there are *no* elements of $S \times S^A$ that are mapped onto aa. Hence, $S \times S^A$ lacks the $S\langle-\rangle$-algebra structure that would be present, had this extension been obtained by a distributive law $\lambda : S\langle S \times -^A\rangle \Rightarrow S \times S\langle-\rangle^A$.

From this counterexample, it directly follows that Lemma 3.5 is, unlike Lemma 2.15, not an instance of Lemma 7.2. There are, however, still possibilities to regard the constructively algebraic power series as resulting from a distributive law, which we will consider in the next section.

7.2.3 Syntactic systems

We will now give a short summary of the situation where T is a term algebra over some signature. The case of context-free languages has, in a more refined setting using *co-pointed functors*, been explored in [BHKR13]. We assume S to be a *signature functor*, reflecting the algebraic operations and their arity. Rather than going into formal definitions here, let us illustrate this with the signature functor

$$\mathrm{S}(X) = S + A \times X + X^2 + X^2$$

corresponding to the signature employed in Section 6.1, consisting of elements of the semiring S (which can be regarded as nullary operations), pairs of elements from A and X (which can be regarded as a A-indexed family of unary operations, representing the . operator), and two binary operations corresponding to $\oplus$ and $\otimes$.

Each signature functor S uniquely extends to a term algebra $\mathfrak{T}_\mathrm{S}$, which has the structure of a monad. This monad $(\mathfrak{T}_\mathrm{S}, \eta, \mu)$ is also called the *free monad* generated by the functor S.

As a consequence of the next, crucial, result, any natural transformation from SF to $F\mathfrak{T}_S$ can be extended to a distributive law of the monad $\mathfrak{T}_S$ over F:

Proposition 7.6. *If S is any signature functor, there is a bijective correspondence between natural transformations*

$$\lambda^- : \mathrm{S}F \Rightarrow F\mathfrak{T}_\mathrm{S}$$

and distributive laws

$$\lambda : \mathfrak{T}_{\mathrm{S}}F \Rightarrow F\mathfrak{T}_{\mathrm{S}}$$

of the monad T over F.

Proof. Found in [Bar04, Lemma 3.4.24]. □

As an example, the natural transformation $\lambda_X^- = (o_X, \delta_X)$ can be specified by the equations

$t \in \mathrm{S}(S \times -^A)$	$o_X(t)$	t_a
$s \in S$	s	0
$(b.(o_u, d_u)),\ b \in A$	0	**if** $b = a$ **then** $\mathbf{int}(o_u, d_u)$ **else** 0
$((o_u, d_u) \oplus (o_v, d_v))$	$o_u + o_v$	$(d_u(a) \oplus d_v(a))$
$((o_u, d_u) \otimes (o_v, d_v))$	$o_u o_v$	$((d_u(a) \otimes \mathbf{int}(o_u, d_u)) \oplus (o_u \otimes d_v(a)))$

where **int** is defined on output derivative pairs as follows:

$$\mathbf{int}(o, d) := \left(o \oplus \left(\bigoplus_{a \in A}(a.d(a))\right)\right).$$

This function can be seen as 'integrating' output-derivative pairs to corresponding terms: in the current setting, this is necessary because, as a result of the type of the natural transformation, we cannot use the variables in the specification. We remark, however, that this issue can be solved by resorting to *co-pointed* functors: for the specific case of context-free languages, this was done in [BHKR13].

Except for the presence of the **int** operator in the specification, the above specification is essentially the same as the specification in Section 6.1. In fact, because for all $\sigma \in S\langle\langle A\rangle\rangle$ the equality

$$\sigma = \left(O(\sigma) \oplus \bigoplus_{a \in A}(a.\sigma_a)\right).$$

holds true, it directly follows that the final λ-bialgebra will be identical in its algebraic structure to the $\mathfrak{T}_S$-structure given on $S\langle\langle A\rangle\rangle$ in Section 6.1, giving an interpretation of the syntactic operators corresponding to the semiring structure with the convolution product. From this it also follows that any $\sigma \in S\langle\langle A\rangle\rangle$ is $\mathfrak{T}_{\mathrm{S}}$-describable in Σ if and only if it is polynomial in Σ, leading to the conclusion that, with respect to this distributive law, the notions of λ-algebraic and constructively S-algebraic coincide.

7.3 Brzozowski bialgebras

Given a commutative semiring S, an *S-algebra* is a tuple

$$(X, 0, 1, +, \cdot, \cdot_s)$$

such that $(X, +, 1, +, \cdot)$ is a semiring, and $(X, 0, +, \cdot_s)$ is a S-module. Moreover, the semiring and S-module structure interact as follows:

$$(s \cdot_s 1) \cdot t \quad = \quad s \cdot_s t \quad = \quad t \cdot (s \cdot_s 1).$$

S-algebras again form a category, with the morphisms given by mappings that are both semimodule and semiring morphisms. Given any S-algebra M, any set X, and a function $f : X \to M$, there moreover is always a unique S-algebra $f^\sharp : S\langle X\rangle \to M$ making the diagram

$$\begin{tikzcd} X \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "f"'] & S\langle X\rangle \arrow[dl, dotted, "f^\sharp"] \\ M & \end{tikzcd} \tag{7.1}$$

commute. In fact, the category of S-algebras and their morphisms is equivalent to the Eilenberg-Moore category of $S\langle - \rangle$-algebras, described in Section 7.2.2.

Given a commutative semiring S, we define a *Brzozowski S-bialgebra* as a tuple

$$(X, 0, 1, +, \cdot, \cdot_s, o, \delta)$$

such that $(X, 0, 1, +, \cdot, \cdot_s)$ is an S-algebra, (X, o, δ) is an S-linear automaton, o is a semiring morphism, and moreover the product rule

$$1_a = 0 \qquad \text{and} \qquad (st)_a = s_a t + o(s) \cdot_s t_a$$

is satisfied.

We can now state a more general version of Lemma 3.5, generalizing the result from the final S-automaton to arbitrary Brzozowski bialgebras:

Lemma 7.7. *Given any polynomial system (X, o, δ) with coefficients in S and any Brzozowski S-bialgebra (Q, p, γ), if a function $f : X \to Q$ makes the diagram*

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "(o,\delta)"'] & Q \arrow[d, "(p,\gamma)"] \\
S \times S\langle X\rangle^A \arrow[r, "1_S \times (f^\sharp)^A"] & S \times Q^A
\end{tikzcd}$$

commute, then the unique S-algebra morphism $f^\sharp : S\langle X\rangle \to S\langle\langle A\rangle\rangle$ extending f makes the diagram

$$\begin{tikzcd}
X \arrow[r, hook, "\eta^{\mathrm{p}}_X"] \arrow[d, "(o,\delta)"'] & S\langle X\rangle \arrow[r, "f^\sharp"] \arrow[dl, "(\hat{o},\hat{\delta})"'] & Q \arrow[d, "(p,\gamma)"] \\
S \times S\langle X\rangle^A \arrow[rr, "1_S \times (f^\sharp)^A"] & & S \times Q^A
\end{tikzcd}$$

commute.

Proof. Identical to the proof of Lemma 3.5, with the exception of removal of the last sentence of the proof and replacement of all instances of $S\langle\langle A\rangle\rangle$, O, and Δ with Q, p, and γ. □

Furthermore, given any Brzozowski S-bialgebra, the mapping $[\![-]\!]$ into the final automaton is a semiring morphism:

Proposition 7.8. *Given any Brzozowski S-bialgebra (Q, o, δ), $[\![-]\!]$ is a S-algebra morphism.*

Proof. We already know from Chapter 2 that $[\![-]\!]$ is an S-linear mapping, as every Brzozowski S-bialgebra is an S-linear automaton. Now consider the relation

$$R = \{(st, [\![s]\!][\![t]\!] \mid s, t \in Q\} \cup \{(1, 1)\}$$

The requirement on output values follows from $o(1) = O(1)$ and

$$o(st) = o(s)o(t) = O([\![s]\!])O([\![t]\!]) = O([\![s]\!][\![t]\!])$$

For the requirement on derivatives, in case of the pair $(1,1)$, we get $1_a = 0 \, \Sigma R \, 0 = 1_a$, and if $(st, [\![s]\!][\![t]\!]) \in R$, we get:

$$\begin{aligned} (st)_a &= s_a t + o(s) t_a \\ &\Sigma R \; [\![s_a]\!][\![t]\!] + [\![o(s)]\!][\![t_a]\!] \\ &= [\![s]\!]_a [\![t]\!] + o([\![s]\!])[\![t_a]\!] \\ &= ([\![s]\!][\![t]\!])_a \end{aligned}$$

Now it follows that $[\![1]\!] = 1$ and $[\![st]\!] = [\![s]\!][\![t]\!]$, establishing that $[\![-]\!]$ is a semiring morphism. □

Hence, it follows that the final automaton, together with its S-algebra structure, is a final Brzozowski bialgebra. Dually, the Brzozowski S-bialgebra on S with $o(s) = s$ and $s_a = 0$ for all $s \in S$ is an initial object in the category of Brzozowski S-bialgebras, adding a unique S-automaton structure to the initial S-algebra (i.e.: S).

It now follows that the categories of Brzozowski bialgebras are, in a number of ways, very similar to categories of λ-bialgebras:

- The objects are both coalgebras for a functor and algebras for a monad, satisfying interaction rules specified by a *product rule* or *distributive law.*
- There is an initial as well as a final S-Brzozowski bialgebra for any commutative semiring S.
- The statement of Lemma 7.7 corresponds to the statement of Lemma 7.2 (although its proof, relying on induction, differs).

However, unlike in the case of λ-bialgebras, Brzozowski S-bialgebras do *not* lead to an S-algebra structure on $S \times S\langle X\rangle^A$ (the representant of FTX), as shown in Counterexample 7.5.

As a final remark concluding this chapter, we observe that if we replace Brzozowski's product rule in the definition of a Brzozowski bialgebra with the Leibniz product rule (presented here in the case of a single variable)

$$(st)' = s't + st'$$

the resulting definition is equivalent to the usual definition of a *differential S-algebra* (see for example [AMT09][1]), combined with a semiring morphism o into

[1]or `http://en.wikipedia.org/wiki/Differential_algebra`

S (again regarded as output). If we define the category of differential S-algebras together with output functions o analogously, there again is a final object

$$(S\langle\langle A\rangle\rangle, 0, 1, +, \times, \cdot_S, O, \Delta)$$

where $(S\langle\langle A\rangle\rangle, 0, +, \cdot_S, O, \Delta)$ is the final S-linear automaton and where $\times$ denotes the *shuffle product*, which can be given explicitly (for streams) by

$$(\sigma \times \tau)(n) = \sum_{k=0}^{n} \binom{n}{k} \sigma(n-k)\tau(k)$$

as is shown in [Rut03a]. In a way, we thus can regard Brzozowski bialgebras as differential algebras (with output) for Brzozowski's product rule.

Leaving further details as future work, we expect that it should, without significant problems or obstacles, be possible to translate some of the main constructions and results from Chapter 3 and this chapter analogously for differential S-algebras with outputs. In particular, we conjecture that the analogous statement of Lemma 7.7 again holds in this setting.

8

Stream calculus in Haskell

We now take a look at how the formats for behavioural differential equations presented in Chapters 2, 3, and 5 can be implemented in the functional programming language Haskell. Haskell provides an elegant and highly usable setting for coinductive techniques in general, in particular including coinductive stream calculi. The connection between coinduction, coinductive stream calculus in Haskell, and formal power series as streams has been explored in for example [McI99], [McI01], [DvE04], and [Hin11].

In this chapter, we will recall some of the constructions from this earlier work, and show how the various classes of streams described in Chapters 2, 3, and 5 can be described in a way that is essentially 'bialgebraic'—i.e. incorporating both algebraic and coalgebraic structure.

This chapter does not provide an introduction to the programming language Haskell. Nor will we say much, in explicit terms, about the type system of Haskell (in fact, all type declarations are omitted in the code presented in this chapter), only focussing on it when especially relevant. General introductions to the Haskell programming languages can be found in e.g. [HF92], [DI06], or [OGS08].

As alternatives to Haskell or Haskell-based tools for working with streams, various other tools for reasoning with stream calculus exist. CIRC [LGCR09] and Streambox [ZE11] are two such tools: both focus on automatic proving of stream equality by means of bisimulation techniques.

8.1 Coinduction in Haskell

To start with an elementary example, consider the following Haskell specification:

```
x = 1:2:3:x
```

This specification uses the cons operator `:` (corresponding to the operator :: from Chapter 3) to specify a variable x type `[Integer]`. A variable of this type represents either a finite list or an infinite stream of the type `Integer`.

With this specification, it is directly possible to obtain initial segments of the stream `x` as follows:

```
>>> take 10 x
[1,2,3,1,2,3,1,2,3,1]
```

This definition works because of Haskell's *lazy evaluation*, corresponding to the idea that expressions are, at any point in time, only evaluated in so far as this is required. This, in combination with the *guardedness* of the equation (here reflecting the fact that, on the right hand side, the variable `x` only reoccurs behind the cons operator in its own definition), enables us to access as many elements of the stream as we like, without any requirement to 'compute' the entire stream.

Eventually periodic, or simple, streams (over any type whatsoever) can always be specified using specifications of this type.

QStream and OEIS integration The package QStream, which can be found at

`http://homepages.cwi.nl/~winter/qstream/`

provides an easy interface for the stream calculus presented in this chapter. It is comparable to, and inspired by e.g. McIlroy's implementation[1].

The functions `o` and `d`, standing for output and derivative are simply defined as synonyms for `head` and `tail`:

```
o = head
d = tail
```

Furthermore, a function `dd` with two arguments is defined, giving the nth derivative of a stream:

```
dd n s = iterate d s !! n
```

In addition to the general stream calculus implementations provided by McIlroy and Hinze, QStream also offers helper functions for integration with the OEIS. The most elementary function, `info`, simply looks up a stream, based on the first 15 (or any other arbitrary number) elements. For the periodic stream defined above, this gives:

```
>>> info x
Simple periodic sequence: repeat 1,2,3. (A010882)
```

[1]`http://www.cs.dartmouth.edu/~doug/powser.html`

8.2 Streams with algebraic structure

In order to be able to produce more interesting classes of streams than the eventually periodic streams, we coinductively define a few basic operations on streams. We are here concerned with streams where the underlying type is a `Num` type, or a numerical type. Numerical types in Haskell can, essentially, be seen as being (roughly) the analogue of rings, together with a signum and absolute value function.

Following the example of [McI99], we let streams be a `Num` type, enabling us to reap the fruits of type coercion, use the standard operators `+` and `*`, and furthermore directly inherit functions such as `sum`, `^`, etc.

We first define a scalar product operator `*!`, purely for pragmatic reasons, and playing the same role as the scalar product of modules, as it computes the scalar product much faster than the full convolution product defined by Brzozowski's product rule. We define this as an infix operator of precedence 7, the same precedence as the multiplication operator `*`.

```
infixl 7 *!

(*!) k = map (k*)
```

We now extend any numerical type `a` to a numerical type on `[a]`, as follows:

```
instance Num a => Num [a] where
  fromInteger = i . fromInteger
  negate = map negate
  (+) = zipWith (+)
  s * t = o s * o t : d s * t + o s *! d t
  signum = error "undefined"
  abs = error "undefined"
```

Here the signum and absolute value functions are left undefined, because it is, on streams, generally impossible to give definitions for these two functions satisfying the equation, required to hold for any `Num` type, relating the two. (We thus take a pragmatic approach here.) If we had not defined the scalar product `*!`, the product could be defined by

```
  s * t = o s * o t : d s * t + i (o s) * d t
```

resulting in slower computation of the product.

Fractional types, involving multiplicative inverses, can be extended similarly, by defining the `recip` operator:

```
instance Fractional a => Fractional [a] where
  fromRational = i . fromRational
  recip s = recip (o s) : recip (o s) *! recip s * d s
```

This extension of the `Num` type directly allows us to represent rational and algebraic streams in Haskell using elegant and clear notation. In addition to the examples given below, Appendix C presents a selection of coinductive streams for the various classes, together with the corresponding OEIS entries.

Rational streams The system from Example 2.17 can directly be translated to the Haskell equation,

```
fibs = 0 : 1 : fibs + d fibs
```

coinductively specifying the Fibonacci numbers, and directly corresponding to the system of behavioural differential equations:

$$o(x) = 0 \qquad o(y) = 1 \qquad x' = y \qquad y' = x + y$$

As additional examples of rational streams, consider:

```
dups = 1 : 2 *! dups
hypercube = 1 : 2 *! (hypercube + dups)
```

Here `dups` consists of the powers of 2: (1,2,4,8,...), and the nth element of `hypercube`, is equal to the number of edges in a n-dimensional hypercube.

It is also possible to define complete systems of streams at once. As an example, the following stream systems yield, respectively, diagonal rows from Pascal's triangle, and the Stirling numbers of the 2nd kind:

```
pascal n = 1 : sum [ pascal i | i <- [1..n] ]
stirling2 n = 1 : sum [ i *! stirling2 i | i <- [1..n] ]
```

These Haskell specifications are in direct correspondence to the systems of behavioural differential equations

$$\begin{aligned} o(p_n) &= 1 \qquad & p'_n &= \sum_{i=1}^{n} p_i \qquad & (n \in \mathbb{N}) \\ o(s_n) &= 1 \qquad & s'_n &= \sum_{i=1}^{n} i s_i \qquad & (n \in \mathbb{N}) \end{aligned}$$

giving respectively

$$[\![p_n]\!](k) = \binom{n+k}{k} \qquad \text{and} \qquad [\![s_n]\!](k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^{n+k}$$

which can furthermore easily be derived from the familiar recurrence relations for these sequences.

QStream provides three additional helper functions for systems of streams: `rangeinfo`, `maketable` and `oeistable`, which display information about a range of streams in a system or which display tables containing an initial part of a system of streams.

The use of these functions is best illustrated with the following examples (`maketable` is identical to `oeistable` with the exception that it doesn't look up the streams at the OEIS):

```
>>> rangeinfo stirling2 [1..5]
1: The simplest sequence of positive numbers: the all 1's
sequence. (A000012)
2: 2^n - 1. (Sometimes called Mersenne numbers, although that
name is usually reserved for A001348.) (A000225)
3: Stirling numbers of second kind S(n,3). (A000392)
4: Stirling numbers of the second kind, S(n,4). (A000453)
5: Stirling numbers of the second kind, S(n,5). (A000481)
```

```
>>> oeistable pascal 10 5
 1:  1  1  1   1   1    1    1    1    1    1   (A000012)
 2:  1  2  3   4   5    6    7    8    9   10   (A000027)
 3:  1  3  6  10  15   21   28   36   45   55   (A000217)
 4:  1  4 10  20  35   56   84  120  165  220   (A000292)
 5:  1  5 15  35  70  126  210  330  495  715   (A000332)
```

Algebraic streams Similarly to rational streams, systems of behavioural differential equations for algebraic streams again directly correspond to coinductive definitions in Haskell, where the full product `*` can be used in the derivatives of specifications. A well-known example of an algebraic stream is the specification

```
cats = 1 : cats^2
```

corresponding to the system of behavioural differential equations from Example 3.10 and again yielding Catalan numbers. (This coinductive definition of the Catalan numbers originated from [DvE04].)

Likewise, the large Schröder numbers from Example 3.21 can directly be represented in Haskell using the equation

```
lschroeder = 1 : lschroeder + lschroeder^2
```

Sometimes, systems of behavioural differential equations, and the corresponding specifications in Haskell, are much simpler in form than the sometimes more familiar explicit formulas for these sequences. For example, the number of m-ary search trees on n keys is equal to the nth element of the stream `searchtrees m`, specified by:

```
searchtrees m = take (m-1) ones ++ searchtrees m^m
```

This equation can easily be derived from the generating function

$$A(X) = \sum_{j=0}^{m-2} X^j + X^{m-1} A^m(X),$$

found in [FD97], where a (much lengthier) corresponding explicit formula is also provided.

Transcendental streams Another example, this time of an infinite polynomial system, can be constructed for the stream (for a fixed n) of which the kth element is the number of terms in the λ-calculus of size k with n free De Bruijn indices. This number can be given by the recurrence (as shown in [GL14], to which we also refer for a discussion of what is meant exactly by the size of a λ-term)

$$\begin{aligned} \sigma_n(0) &= n \\ \sigma_n(k+1) &= \sigma_{n+1}(k) + \sum_{i=0}^{k} \sigma_k(i)\sigma_k(k-i) \end{aligned}$$

from which the behavioural differential equation

$$o(\sigma_n) = n \qquad \sigma_n' = \sigma_{n+1} + \sigma_n^2$$

and the Haskell specification

```
lambda n = n : lambda (n+1) + lambda n^2
```

can be derived immediately. This is an infinite system, so we are not guaranteed that it is algebraic. In fact, each of the streams σ_i is *transcendental*[2], which can be shown by establishing that each of these streams dominates the counting function of any context-free grammar.

k-automatic and k-regular sequences In order to be able to represent the classes of k-automatic and k-regular sequences coinductively in Haskell, we need one additional ingredient—the $\mathbf{zip}_k$ operations. In fact, a **zip** operation of arbitrary arity can be specified in Haskell using the equation

```
xzip (s:t) = head s : xzip (t ++ [tail s])
```

allowing us to coinductively specify all of the examples from Chapter 5. The streams from Examples 5.5, 5.6, 5.11 and 5.12 can now be coinductively represented using the following equations:

```
tm = 0 : xzip [ones - tm, d tm]
cantor = 1 : xzip [0, cantor, d cantor]
noergaard = 1 : xzip [-noergaard, noergaard + ones]
kimberling = 1 : xzip [kimberling, nats + ones]
```

Note that the specification for the Thue-Morse sequence here is a slight modification of the specification given in Example 5.5: the second variable of the system has been replaced by the equivalent `ones - tm`, allowing us to give a specification in a single line.

Hadamard and shuffle products The Hadamard and shuffle products can, again, be easily specified in Haskell, using analogues of the corresponding behavioural differential equations. Because the Hadamard product is pointwise, it can be specified simply by:

```
hadamard = zipWith (*)
```

The shuffle product can be specified in a similar manner to the convolution product, again in correspondence to the behavioural differential equation:

```
shuffle s t = o s * o t : shuffle (d s) t + shuffle s (d t)
```

[2]i.e. not algebraic

Moessner's construction We have now seen that Haskell easily allows for coinductive specification of streams, and will now turn to an example of a coinductive construction involving additional operators. For the case of $k = 2$, Moessner theorem states that, starting from the natural numbers

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots$$

the squares can be obtained by first removing each even element

$$1, 3, 5, 7, 9, 11, 13, 15, 17, 19, \dots$$

and then taking partial sums:

$$1, 4, 9, 16, 25, 36, 49, 64, 81, 100, \dots$$

The more general version states that by first removing each kth element, then taking partial sums, and then performing the corresponding construction for $k - 1$, the kth powers are obtained. This result was originally proven in [Moe51] and more recently a coinductive proof was given in [NR11] using bisimulation techniques. A formalized proof of Moessner's theorem, again based on bisimulation techniques, has recently been given in Coq in [KPS14].

The partial sum and drop operators can be implemented in Haskell using the lines

```
psum s = o s : repeat (o s) + psum (d s)
xdrop k s = take (k-1) s ++ dd k s
```

and Moessner's construction itself can now be specified as follows (for arbitrary k):

```
moessner k = foldr (.) id [psum . xdrop i | i<-[1..k]]
```

The statement of Moessner's theorem now can be stated as

> 'The stream `moessner k ones` is equal to the stream of all kth powers.'

Haskell is, without the appropriate extensions, unable to give us a formal proof of this theorem, but at least it gives some empirical validation:

```
>>> take 10 (moessner 3 ones)
[1,8,27,64,125,216,343,512,729,1000]
```

8.3 Point-free definitions for stream calculus

To conclude this chapter, we will give alternate specifications of the coinductive operators using *point-free* style, that is, a style in which functions are specified purely in terms of other functions and basic constructions and combinators, without directly referencing the argument. The general motivation for using such a style lies in the fact that point-free style is considered more abstract and elegant, as well as providing a closer link to the language of category theory.

In order to be able to present a point-free specification of the stream calculus in Haskell, we specify a function `lift` as follows:

```
lift f g h x = f (g x) (h x)
```

For example, `lift (:)`, the lifting of the cons operator, specifies a function taking first a function from some type `s` to another type t, and then a function from `s` to `[t]`, and returning again a function from `s` to `[t]`.

This allows us to extend the `Num` type to any type of function that has a `Num` type as its domain:

```
instance Num a => Num (b -> a) where
  fromInteger = const . fromInteger
  (+) = lift (+)
  (*) = lift (*)
  negate = (.) negate
  abs = (.) abs
  signum = (.) signum
```

This allows us to perform basic arithmetic operations on functions with a `Num` type as codomain. For example, if we declare functions `f x = 3*x` and `g x = 4*x + 1`, we can now evaluate, for example:

```
>>> (f*g + 2*g + 1) 2
73
```

With these liftings, it is possible to replace our coinductive definitions of `(*)` and `recip` with an equivalent, point-free definition:

```
(*) = curry (lift (:) (o.fst * o.snd)
                      (d.fst * snd + i.o.fst * d.snd))

recip = lift (:) (recip.o) (-i.recip.o * d * recip)
```

These definitions are again coinductive: in the case of `*`, note that the first occurrence of `*` on the right hand side is the (assumed) multiplication of the underlying `Num` type, and the second and third are guarded corecursive occurrences of (the lifted versions) of the operator which is being defined on streams of the underlying type.

9

Further directions

We now conclude with a number of ideas for future work, further building on the results and approach from this dissertation.

In Chapter 2, our definition of the star operator can be regarded as being based on the discrete topology. One possible direction for future work is to investigate the possibilities of generalization to other topologies and metric spaces, in the setting of *topological semirings.* This may lead to further generalizations of 2.22, with the possibility of applications in ε-elimination (i.e. elimination of empty word-transitions). More generally, it may be worthwhile to try to further refine those results in Chapters 3 and 5 that relate to Arden's rule and the Greibach normal form, in the setting of idempotent semirings, creating a connection to the world of Kleene algebra (see e.g. [Koz90]).

It is known from e.g. [Wor09] that algebras over a semiring S can be regarded as monoids in the (monoidal) category of S-modules. Likewise, it appears to be that Brzozowski bialgebras can somehow be seen as monoids in the category of S-linear automata (and similarly for differential algebras with output). This raises the question about the possibilities to regard the category of S-linear automata as a monoidal category. One possible direction for future work is investigation of possibilities to see the category of S-linear automata as a monoidal category, and connect this to the various types of products that can be defined coinductively.

The work in Chapter 3 can be seen as a presentation of the second level of the Chomsky hierarchy, the context-free languages, giving (together with the existing presentation of the regular languages) a coalgebraic picture of the first two levels of the Chomsky hierarchy. It remains to be seen to which extension it is possible to give similar characterizations of the remaining two levels, the context-sensitive languages. With regards to the recursively enumerable languages, we note that it is impossible to obtain them, in any computional manner whatsoever, as this family extends the computable languages, ruling out any 'reasonable' co- or bialgebraic treatment of this class. It may be worthwhile to investigate if we can give some (one, or more) coalgebraic characterization of the computable languages, and possibly connect this to coalgebraic models of computation and/or coalgebraic semantics of programming languages. We note that recently, some work towards a coalgebraic Chomsky hierarchy has been presented in [GMS14].

A final possibility for future work, on a more applied level, is investigating whether the results on (weighted) context-free grammars could lead to applications in the area of natural language parsing (see e.g. [JM00] for an introduction to the field). For this purpose, we can rely on the Viterbi semiring defined by

$$([0,1], \cdot, \max, 0, 1),$$

which would give, when used as the underlying semiring for a polynomial system, the probability of the most likely derivation. This application would entail a combination of the theoretically oriented work from this dissertation with more practical techniques: for efficiency reasons, we may for example consider pruning techniques.

A

Algebra

We recall the essential definitions of the algebraic structures *monoids* and *semirings*, as well as of *modules* over a semiring.

A.1 Monoids

A *monoid* $(M, \cdot, 1)$ consists of a set M, together with a distinguished element $1 \in M$, and a binary multiplication operator $\cdot : M \times M \to M$, such that for all $m, n, o \in M$ we have:

$$\begin{array}{rcl} m \cdot (n \cdot o) & = & (m \cdot n) \cdot o \\ m \cdot 1 & = & m \\ 1 \cdot m & = & m \end{array}$$

We call a monoid *commutative* if for all $m, n \in M$, $mn = nm$. (Following usual conventions of omitting the multiplication symbol) Often commutative monoids will be represented in additive notation, using the symbols 0 and $+$ rather than 1 and $\cdot$. Moreover, a monoid is called *idempotent* if, for all m, M, $mm = m$.

Given two monoids $(M, \cdot_M, 1_M)$ and $(N, \cdot_N, 1_N)$, a function $f : M \to N$ is a *monoid morphism* whenever:

$$\begin{array}{rcl} f(m \cdot_M n) & = & f(m) \cdot_N f(n) \\ f(1_M) & = & 1_N \end{array}$$

Given any (finite or infinite) alphabet X, let X^* denote the set of all *words* over X, that is, lists of finite sequences of elements from X. We use the symbol 1 to denote the empty word—that is, the sole word of length zero, and given words

$$v = x_1 \dots x_m \qquad \text{and} \qquad w = y_1 \dots y_n,$$

let the product $v \cdot w$ be defined by the concatenation

$$vw = x_1 \ldots x_m y_1 \ldots y_n.$$

It is easy to see that the unit and associativity laws for monoids hold here, turning the structure $(X^*, \cdot, 1)$ into a monoid.

Now, given any X, let us define $\eta^{\mathrm{w}}_X : X \to X^*$ by mapping each symbol $x \in X$ to the singleton word $x \in X^*$. This gives rise to the following universal mapping property:

Proposition A.1. *Given a set X, a monoid $(M, \cdot_M, 1_M)$, and a function $f : X \to M$, there is a unique monoid morphism $\bar{f} : X^* \to M$ such that $\bar{f} \circ \eta^{\mathrm{w}}_X = f$.*

Proof. See e.g. [Awo10, Proposition 1.9] □

In other words, X^* is the *free monoid* on X.

A.2 Semirings

A *semiring* $(S, +, \cdot, 0, 1)$ consists of a set S, such that $(S, \cdot, 1)$ is a monoid, and $(S, +, 0)$ is a commutative monoid, moreover satisfying:

$$\begin{array}{rcl} r \cdot (s + t) & = & r \cdot s + r \cdot t \\ (r + s) \cdot t & = & r \cdot t + s \cdot s \\ 0 \cdot r & = & 0 \\ r \cdot 0 & = & 0 \end{array}$$

A semiring $(S, +, \cdot, 0, 1)$ is *commutative* whenever the multiplicative monoid $(S, \cdot, 1)$ is commutative, and *idempotent* the additive monoid $(S, +, 0)$ is idempotent.

Given semirings S and T, a function $f : S \to T$ is called a *semiring morphism* whenever it is a monoid morphism with respect to both the additive and multiplicative monoid underlying the semiring.

Two important instances of semirings are the semiring $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$

$$(\mathbb{N}, +, \cdot, 0, 1)$$

of natural numbers, which is commutative, and the Boolean semiring $\mathbb{B} = \{0, 1\}$

$$(\mathbb{B}, \vee, \wedge, 0, 1).$$

which is commutative as well as idempotent.

Semirings importantly include *rings* such as $\mathbb{Z}$ (which are semirings where every element x has a unique inverse $-x$ such that $x + (-x) = 0$), and *fields* such as $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ (which are commutative rings where every nonzero element x has a unique multiplicative inverse x^{-1} such that $xx^{-1} = 1$).

A.3 Modules

This section introduces *modules* over a semiring S, which can be regarded as semiring-valued spaces in the same way as vector spaces can be regarded as field-valued spaces for fields F.

Remark A.2. These are, in the context of semirings, usually called semimodules. Following [BR11], we do not use the term *semimodule*, but simply module, also because the definitions are equivalent when S is is a ring.

Given a semiring S, a *left S-module* $(M, +, \cdot, 0)$ or a left module over S, consists of a commutative monoid $(M, +, 0)$ and an operation $\cdot : S \to M \to M$, such that for all $r, s \in S$ and $m, n \in M$:

$$\begin{array}{rcl@{\qquad}rcl} r \cdot (m+n) & = & r \cdot m + r \cdot n & 1 \cdot m & = & m \\ (r+s) \cdot m & = & r \cdot m + s \cdot m & 0 \cdot m & = & 0 \\ (rs) \cdot m & = & r \cdot (s \cdot m) & r \cdot 0 & = & 0 \end{array}$$

Analogous to this definition, a *right S-module* $(M, +, \cdot, 0)$ consists of a commutative monoid $(M, +, 0)$ and an operation $\cdot : M \to S \to M$, such that for all $r, s \in S$ and $m, n \in M$:

$$\begin{array}{rcl@{\qquad}rcl} (m+n) \cdot r & = & m \cdot r + n \cdot r & m \cdot 1 & = & m \\ m \cdot (r+s) & = & m \cdot r + m \cdot s & m \cdot 0 & = & 0 \\ m \cdot (rs) & = & (s \cdot m) \cdot r & 0 \cdot r & = & 0 \end{array}$$

When S is a commutative semiring, the notions of left and right S-modules, and of left and right-linear mappings are equivalent, and are simply called *S-modules.*

Given two left S-modules $(M, +_M, 0_M, \cdot_M)$ and $(N, +_N, 0_N, \cdot_N)$, a function $f : M \to N$ is called a *left S-linear mapping* whenever for all $m, n \in M$ and $s \in S$:

$$f(0_M) = 0_N$$

$$\begin{aligned} f(m +_M n) &= f(m) +_N f(n) \\ f(s \cdot_M m) &= s \cdot_N f(m) \end{aligned}$$

Again, the notion of a *right S-linear mapping* can be defined analogously, and when S is commutative, the two notions coincide.

In the specific case of the Boolean semiring $\mathbb{B}$, a $\mathbb{B}$-module is the same thing as a monoid that is both commutative and idempotent, which is also called a *bounded join-semilattice*.

B

Category theory and universal coalgebra

This section, like the previous section on algebra, presents the basic definitions, together with some relevant examples. Thorough introductions to category theory can be found in e.g. [Awo10] and [Mac71]; a good and comprehensive reference to universal coalgebra is [Rut00].

B.1 Categories and functors

A *category* $\mathbf{C}$ consists of:

1. A class $\mathbf{C_0}$ of objects and a class $\mathbf{C_1}$ of morphisms (or arrows);

2. Operators **dom** and **cod** assigning to each morphism $f \in \mathbf{C_1}$ a domain $\mathbf{dom}(f) \in \mathbf{C_0}$ and a codomain $\mathbf{dom}(f) \in \mathbf{C_0}$ (we use the notation $f : X \to Y$ to denote $\mathbf{dom}(f) = X \wedge \mathbf{cod}(f) = Y$);

3. A composition operator $\circ$ assigning to morphisms $f : X \to Y$ and $g : Y \to Z$ a composition $g \circ f : X \to Z$, satisfying the condition that for all $f : X \to Y$, $g : Y \to Z$, $h : Z \to W$, we have

$$h \circ (g \circ f) = (h \circ g) \circ f$$

4. For each $X \in \mathbf{C}_0$, there a morphism $1_X : X \to X \in \mathbf{C}_1$ satisfying

$$1 \circ f_Y = f = f \circ 1_X$$

   for all $f : X \to Y$.

Some relevant categories are the category of sets and functions between them, the category of monoids/semirings and monoid/semiring morphisms between them, categories of S-modules and linear mappings between them.

Given a category $\mathbf{C}$, a diagram consists of a pictorial representation of objects and morphisms in $\mathbf{C}$, e.g.

$$\begin{tikzcd} X \arrow[d, "f"'] \arrow[dr, "h"] & \\ Y \arrow[r, "g"] & Z \end{tikzcd} \tag{B.1}$$

with the source and target of an arrow indicating the domain and codomain. A diagram is said to *commute* if for any two objects X, Y, and paths of arrows

$$f_1, \ldots, f_m \qquad \text{and} \qquad g_1, \ldots, g_m$$

going from X to Y, we have

$$f_m \circ \ldots \circ f_1 = g_n \circ \ldots \circ g_1.$$

In the above example, the diagram commutes if and only if $g \circ f = h$.

We call a morphism $f : X \to Y$ an *isomorphism* whenever there is a $g : Y \to X$ such that $f \circ g = 1_Y$ and $g \circ f = 1_X$, or equivalently, whenever the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, shift left, "f"] & Y \arrow[l, shift left, "g"] \end{tikzcd}$$

We call an object $X \in \mathbf{C_0}$ *initial* whenever for every object $Y \in \mathbf{C_0}$, there is a unique morphism from X to Y, and dually, an object $X \in \mathbf{C_0}$ is called *final* whenever for every object $Y \in \mathbf{C_0}$, there is a unique morphism from Y to X. For example, $\mathbb{N}$ is an initial semiring, and $\mathbb{B}$ is an initial idempotent semiring. Initial objects are unique up to isomorphism.

Given categories $\mathbf{C}$ and $\mathbf{D}$, a *functor* $F : \mathbf{C} \to \mathbf{D}$ consists of mappings $F : \mathbf{C_0} \to \mathbf{D_0}$ and $F : \mathbf{C_1} \to \mathbf{D_1}$ (overloading the symbol F and often omitting the parentheses surrounding the argument of F), such that, for all morphisms $f, g, h \in \mathbf{C}_1$ and all $X \in \mathbf{C}_0$:

1. $\mathbf{dom}(Ff) = F\mathbf{dom}(f)$
2. $\mathbf{cod}(Ff) = F\mathbf{cod}(f)$
3. $F(g \circ h) = F(g) \circ F(h)$ (whenever $g \circ h$ is defined)

4. $F(1_X) = 1_{FX}$

A functor $F : \mathbf{C} \to \mathbf{C}$ going from a category to itself is called an *endofunctor*.

Given a category $\mathbf{C}$ and an endofunctor F on $\mathbf{C}$, a *F-algebra* (X, α) consists of an object $X \in \mathbf{C}_0$, together with a $\mathbf{C}$-morphism $\alpha : TX \to X$. We call a $\mathbf{C}$-morphism $f : X \to Y$ a *T-algebra morphism* from a F-algebra (X, α) to a F-algebra (Y, β) whenever the diagram

$$\begin{tikzcd}
FX \arrow[r, "Ff"] \arrow[d, "\alpha"'] & FY \arrow[d, "\beta"] \\
X \arrow[r, "f"] & Y
\end{tikzcd}$$

commutes. It is now easy to see that F-algebras and their morphisms again form a category.

Dually, given a category $\mathbf{C}$ and an endofunctor F on $\mathbf{C}$, an *F-coalgebra* (X, γ) consists of an object $X \in \mathbf{C}_0$ and a $\mathbf{C}$-morphism $\gamma : X \to FX$. A $\mathbf{C}$-morphism $f : X \to Y$ is called a *F-coalgebra morphism* from a F-coalgebra (X, γ) to a F-coalgebra (Y, δ) whenever the diagram

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\gamma"'] & Y \arrow[d, "\delta"] \\
FX \arrow[r, "Ff"] & FY
\end{tikzcd}$$

commutes. Once again, F-coalgebras and their morphisms form a category.

The following proposition instantiates in Chapter 2 as 'bijective S-automata morphisms are isomorphisms of S-automata':

Proposition B.1. *If a $\mathbf{C}$-morphism f is both an $\mathbf{C}$-isomorphism and a F-coalgebra morphism, then f is an isomorphism in the category of F-coalgebras.*

Proof. See [Rut00, Proposition 2.3] (the proof works for arbitrary categories). □

B.2 Natural transformations and monads

Given categories $\mathbf{C}$ and $\mathbf{D}$ and functors $F, G : \mathbf{C} \to \mathbf{D}$, a *natural transformation* $\theta : F \Rightarrow G$ is a $\mathbf{C_0}$-indexed family of morphisms $\theta_X : FX \to GX$, such that for any morphism $f : X \to Y$ in $\mathbf{C_1}$, the diagram

$$\begin{tikzcd} FX \arrow[r, "\theta_X"] \arrow[d, "Ff"'] & GX \arrow[d, "Gf"] \\ FY \arrow[r, "\theta_Y"] & GY \end{tikzcd}$$

commutes.

Given a category $\mathbf{C}$, a *monad* is a triple (T, η, μ), where T is an endofunctor on $\mathbf{C}$, $\eta : 1_{\mathbf{C}} \to T$ is a natural transformation, called the *unit* of the monad, and $\mu : T^2 \to T$ is a natural transformation, called the *multiplication* of the monad, and moreover for all $X \in \mathbf{C}_0$, the two diagrams

$$\begin{tikzcd} T^3X \arrow[r, "\mu_{TX}"] \arrow[d, "T\mu_X"'] & T^2X \arrow[d, "\mu_X"] \\ T^2X \arrow[r, "\mu_X"] & TX \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[r, "T\eta_X"] \arrow[d, "\eta^{TX}"'] \arrow[dr, "1_{TX}"] & T^2X \arrow[d, "\mu_X"] \\ T^2X \arrow[r, "\mu_X"] & TX \end{tikzcd}$$

commute.

It is common to show these diagrams as diagrams of natural transformations, which we indicate with double arrows:

$$\begin{tikzcd} T^3 \arrow[r, Rightarrow, "\mu T"] \arrow[d, Rightarrow, "T\mu"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\ T^2 \arrow[r, Rightarrow, "\mu"] & T \end{tikzcd} \qquad \begin{tikzcd} T \arrow[r, Rightarrow, "T\eta"] \arrow[d, Rightarrow, "\eta T"'] \arrow[dr, Rightarrow, "1"] & T^2 \arrow[d, Rightarrow, "\mu"] \\ T^2 \arrow[r, Rightarrow, "\mu"] & T \end{tikzcd}$$

Given a monad T over a category $\mathbf{C}$, an *(Eilenberg-Moore) algebra for the* *monad* T is a T-algebra (X, α) such that the diagrams

$$\begin{tikzcd} X \arrow[d, "\eta_X"'] \arrow[dr, "1_X"] & \\ TX \arrow[r, "\alpha"] & X \end{tikzcd} \qquad \begin{tikzcd} T^2X \arrow[r, "\mu_X"] \arrow[d, "T\alpha"'] & TX \arrow[d, "\alpha"] \\ TX \arrow[r, "\alpha"] & X \end{tikzcd}$$

commute. Algebras for the monad T, and the T-algebra morphisms between them again form a category, called the *Eilenberg-Moore category* of T.

When (X, α) is an algebra for the monad (T, η, μ) in a category $\mathbf{C}$, Y is any object in $\mathbf{C}$, and $f : Y \to X$ a morphism in $\mathbf{C}$, there is a unique T-algebra morphism $\hat{f}$ extending f from TX to Y, in other words, $\hat{f}$ is the unique T-algebra morphism making the diagram

$$\begin{tikzcd} Y \arrow[r, hook, "\eta_Y"] \arrow[d, "f"'] & TY \arrow[dl, dotted, "\hat{f}"] \\ X & \end{tikzcd}$$

commute. Moreover, $\hat{f}$ can be given by $\alpha \circ Tf$.

C

A suite of streams

You are standing at the end of a road before a small brick building. Around you is a forest. A small stream flows out of the building and down a gully.

– Colossal Cave Adventure[1]

This appendix shows a selection of coinductively defined streams, together with the Haskell specification (from which behavioural differential equations can directly be derived), and their entry in the OEIS.

Some of the specifications below have been derived from the generation function specifications given in [Plo92], where generating functions for 1031 integer sequences are provided; others have been derived from specifications in terms of a grammar, a (divide and conquer recurrence) or have been found by accident. Many more examples of coinductively defined streams can be found in the file `Catalog.hs` in the QStream package.

In the cases where systems of streams are considered, the notation $n \mapsto$ followed by an OEIS index indicates that the nth component of the system is equal to the sequence with that index. The notation $n \uparrow$ indicates that the equation does not actually evaluate to a stream, because the specification is not guarded (in the case of n).

C.1 Rational streams

The constant stream of ones: A000012

```
ones = 1 : ones
```

The natural numbers: A000027

```
nats = 1 : ones + nats
```

[1]One of the first computer games

Fibonacci numbers: A000045

```
fibonacci = 1 : 1 : fibonacci + d fibonacci
```

Lucas numbers: A000032

```
lucas = 2 : 1 : lucas + d lucas
```

Powers of 2*:* A000079

```
dups = 1 : 2 * dups
```

Lazy caterer's sequence: A000124

```
caterer = 4 : caterer + nats + 2 * ones
```

Triangular numbers: A000217

```
triang = 1 : triang + d nats
```

Stream of squares: A000290

```
squares = 1 : squares + 2 * nats + ones
```

Stream of cubes: A000578

```
cubes = 1 : cubes + 3 * squares + 3 * nats + ones
```

Narayana's cows sequence: A000930

```
narayana = 1 : 1 : 1 : dd 2 narayana + narayana
```

Jacobsthal sequence: A001045

```
jacobsthal = 0 : 1 : 2 * jacobsthal + d jacobsthal
```

Number of edges in a k-dimensional hypercube: A001787

```
hypercube = 1 : 2 * (hypercube + dups)
```

Number of walks of length n between non-adjacent nodes on the Petersen graph: A091002

```
petersen = 1 : 3 * petersen + d petersen_
petersen_ = 1 : ones - 2 * petersen_
```

Powers of n:
$1 \mapsto$ `ones`, $2 \mapsto$ `dups`, $3 \mapsto$ A000244, $4 \mapsto$ A000302, $5 \mapsto$ A000351, ...

```
powersof n = 1 : n * powersof n
```

Pascal's triangle:
$1 \mapsto$ `ones`, $2 \mapsto$ `nats`, $3 \mapsto$ `triang`, $4 \mapsto$ A000292, $5 \mapsto$ A000332, ...

```
pascal n = 1 : sum [pascal i | i<-[1..n]]
pascal2 = (ones^) -- (alternate version)
```

Stirling numbers of the 2nd kind:
$1 \mapsto$ `ones`, $2 \mapsto$ A000225, $3 \mapsto$ A000392, $4 \mapsto$ A000453, $5 \mapsto$ A000481, ...

```
stirling2 n = 1 : sum [i * stirling2 i | i<-[1..n]]
```

Bernoulli's triangle:
$1 \mapsto$ `ones`, $2 \mapsto$ `nats`, $3 \mapsto$ `caterer`, $4 \mapsto$ A000125, $5 \mapsto$ A000127, ...

```
bernoulli n = 2^(n-1) : sum [bernoulli i | i<-[1..n]]
```

nth Powers:
$1 \mapsto$ `nats`, $2 \mapsto$ `squares`, $3 \mapsto$ `cubes`, $4 \mapsto$ A000583, $5 \mapsto$ A000584, ...

```
nthpow n = 1 : sum [choose n i * nthpow i | i <-[0..n]]
```

C.2 Algebraic streams

Catalan numbers: A000108

```
catalan = 1 : catalan^2
```

Central binomial coefficients: A000984

```
binomial = 1 : 2 * catalan * binomial
```

Large Schröder numbers: A006318

```
lschroeder = 1 : lschroeder^2 + lschroeder
```

Small Schröder numbers: A001003

```
sschroeder = 1 : sschroeder * lschroeder
```

Central Delannoy numbers: A001850

```
delannoy = 1 : 2 * lschroeder * delannoy + delannoy
```

Motzkin numbers: A001006

```
motzkin = 1 : 1 : motzkin^2 + d motzkin
```

Riordan numbers: A005043

```
riordan = 1 : 0 : riordan * motzkin
```

Number of ternary search trees on k keys: A019497

```
ternary = 1 : 1 : ternary^3
```

Pfaff-Fuss sequences:
$1 \mapsto$ `ones`, $2 \mapsto$ `catalan`, $3 \mapsto$ A001764, $4 \mapsto$ A002293, $5 \mapsto$ A002294, ...

```
pfafffuss n = 1 : pfafffuss n^n
```

Number of n-ary search trees on k keys:
$1 \uparrow$, $2 \mapsto$ `catalan`, $3 \mapsto$ `ternary`, $4 \mapsto$ A019498, $5 \mapsto$ A19499, ...

```
searchtrees n = take (n-1) ones ++ searchtrees n^n
```

C.3 k-Automatic and k-regular streams

Prouhet-Thue-Morse sequence: A010060

```
ptm = 0 : xzip [ones - ptm, d ptm]
```

Cantor sequence: A088917

```
cantor = 1 : xzip [0, cantor, d cantor]
```

Nørgård's infinity sequence: A004718

```
noergaard = 1 : xzip [-noergaard, noergaard + ones]
```

Kimberling's sequence: A003602

```
kimberling = 1 : xzip [kimberling, nats + ones]
```

Number of 1*s in the binary expansion of k:* A000120

```
count1 = 1 : xzip [count1, count1 + ones]
```

Moser-de Bruijn sequence (sums of distinct powers of 4): A000695

```
mdb = 1 : xzip [4 * mdb, 4 * mdb + ones]
```

Sorting numbers (maximal number of comparisons for sorting k elements by binary insertion): A001855

```
srt = 0 : xzip [2 * (srt + nats) - ones, d srt + srt + 2 * nats]
```

Josephus problem sequence: A006257

```
josephus = 1 : xzip [2 * josephus - ones, 2 * josephus + ones]
```

Numbers whose set of base n digits is $\{0, 1\}$:
$1 \mapsto$ `count1`, $2 \mapsto$ `nats`, $3 \mapsto$ A005836, $4 \mapsto$ `mdb`, $5 \mapsto$ A033042, ...

```
digits01 n = 1 : xzip [n * digits01 n, n * digits01 + ones]
```

C.4 Transcendental streams

Factorials: A000142

```
factorials = 1 : hadamard factorials nats
```

Bell or exponential numbers: number of ways to partition a set of n labeled elements: A000110

```
bell = 1 : shuffle bell ones
```

Number of λ-terms of size k with at most n free de Bruijn indices:
$1 \mapsto$ A220894, $2 \mapsto$ A220895, $3 \mapsto$ A220896, ...

```
lambda n = n : lambda n^2 + lambda (n+1)
```

Samenvatting

De *automatentheorie* is een vakgebied binnen de theoretische informatica, waarin abstracte machines, ofwel automaten, bestudeerd worden. De automatentheorie is nauw verbonden met de studie van *formele talen*, en met klassen van formele talen die door diverse typen automaten en formele grammatica's herkend of beschreven kunnen worden. Belangrijke voorbeelden van zulke klassen worden gegeven door de *reguliere* en *contextvrije* talen, die tevens de eerste twee niveaus van de Chomskyhiërarchie vormen.

In dit proefschrift worden diverse klassen uit de automatentheorie bestudeerd vanuit een *coalgebraïsch* oogpunt. Coalgebra biedt een abstracte kijk op een variëteit aan toestandsgebaseerde systemen, die geworteld is in de *categorietheorie*, een deel van de wiskunde waarin de structurele overeenkomsten tussen verschillende wiskundige theorieën op een abstract niveau bestudeerd worden.

Een ander voorbeeld van automatentheoretische klassen die in dit proefschrift bestudeerd worden, naast de formele talen, zijn de *stromen*[2], ofwel oneindige rijen die *coïnductief* beschreven worden. Meer in het algemeen worden er klassen bestudeerd van formele machtsreeksen in niet-commuterende variabelen: zowel formele talen als stromen kunnen gezien worden als voorbeelden hiervan. Zulke coïnductieve beschrijvingen hebben, in het algemeen, de vorm van (systemen van) *gedragsdifferentiaalvergelijkingen*[3]. In veel gevallen bestaat er een correspondentie tussen een bepaald formaat van (gedrags)differentiaalvergelijkingen, en een daarmee overeenkomende automatentheoretische klasse.

In hoofdstuk 2 wordt een coalgebraïsche presentatie gegeven van deterministische, niet-deterministische, en gewogen automaten, die de reguliere talen en hun generalisaties beschrijven. In dit hoofdstuk worden vooral bestaande resultaten en ideeën (van o.a. Brzozowski, Schützenberger, Rutten, Bonsangue en Silva) gepresenteerd. Deze presentatie wordt in hoofdstuk 3 uitgebreid naar de contextvrije talen en de generalisaties ervan, die coalgebraïsch beschreven kun-

[2]Eng. *streams*
[3]Eng. *behavioural differential equations*

nen worden in een formaat dat correspondereert met contextvrije grammatica's in de Greibach-normaalvorm.

Op de resultaten uit de twee voorgaande hoofdstukken wordt in hoofdstuk 4 verder gebouwd met een bestudering van de $\mathbf{zip}_k$-operator en het Hadamard-product. De $\mathbf{zip}_k$-operator is een operator die k stromen als argument neemt, en beurtelings het initiële element van elk van die stromen neemt, om zo een nieuwe stroom te vormen. Het Hadamardproduct kan gezien worden als een puntsgewijs product, en wordt gebruikt als een basis voor een coalgebraïsche beschrijving van *pushdownautomaten*, die ook in dit hoofdstuk gegeven wordt.

In hoofdstuk 5 wordt er vervolgens een coalgebraïsche beschrijving gegeven van de *k-automatische* en *k-reguliere* rijen, gebaseerd op de eerder gegeven $\mathbf{zip}_k$ operator. Hierna wordt er in hoofdstuk 6 gekeken naar term-algebra's en μ-expressies, die een alternatieve methode geven om de contextvrije talen (en de generalisaties ervan) coalgebraïsch te beschrijven.

In hoofdstuk 7 wordt een deel van het eerdere materiaal beschouwd vanuit een *bialgebraïsch* perspectief, en wordt er een connectie gelegd met het abstracte raamwerk van *λ-bialgebra's* en *distributieve wetten.* In het bijzonder wordt er gekeken naar diverse manieren waarop een synthese gevormd kan worden tussen het werk uit hoofdstuk 3 en bialgebra's, en de problemen die zich hierbij voordoen.

Ten slotte wordt in hoofdstuk 8 een implementatie van de coïnductieve stromencalculus in de functionele programmeertaal Haskell gegeven, waarmee de gepresenteerde formaten uit de eerdere hoofdstukken direct beschreven kunnen worden. Aansluitend hierop wordt in appendix C het materiaal uit de eerdere hoofdstukken geïllustreerd met een aantal interessante voorbeelden van oneindige rijen, coïnductief beschreven als stromen (in een notatie ontleend aan de Haskell-implementatie uit hoofdstuk 8).

Bibliography

[AMT09] Katsutoshi Amano, Akira Masuoka, and Mitsuhiro Takeuchi. Hopf algebraic approach to Picard-Vessiot theory. In M. Hazewinkel, editor, *Handbook of Algebra*, volume 6 of *Handbook of Algebra*, pages 127–171. North-Holland, 2009.

[AS92] Jean-Paul Allouche and Jeffrey O. Shallit. The ring of k-regular sequences. *Theoretical Computer Science*, 98:163–197, 1992.

[AS03a] Jean-Paul Allouche and Jeffrey O. Shallit. *Automatic Sequences—Theory, Applications, Generalizations.* Cambridge University Press, 2003.

[AS03b] Jean-Paul Allouche and Jeffrey O. Shallit. The ring of k-regular sequences, II. *Theoretical Computer Science*, 307:3–29, 2003.

[Awo10] Steve Awodey. *Category Theory.* Oxford University Press, 2010.

[Bar04] Falk Bartels. *On Generalized Coinduction and Probabilistic Specification Formats.* PhD thesis, Vrije Universiteit Amsterdam, 2004.

[BBB$^+$12] Filippo Bonchi, Marcello M. Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Information and Computation*, 211:77–105, 2012.

[BHKR13] Marcello M. Bonsangue, Helle H. Hansen, Alexander Kurz, and Jurriaan Rot. Presenting distributive laws. In Heckel and Milius [HM13], pages 95–109.

[BR11] Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series with Applications.* Cambridge University Press, 2011.

[BRW12] Marcello M. Bonsangue, Jan Rutten, and Joost Winter. Defining context-free power series coalgebraically. In Dirk Pattinson and Lutz Schröder, editors, *CMCS*, volume 7399 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2012.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11:481–494, 1964.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, June 1959.

[Chr70] Gilles Christol. Sur une opération analogue à l'opération de Cartier en caractéristique nulle. *C. R. Acad. Sc. Paris.*, t. 271, série A:1–3, 1970.

[Cob72] A. Cobham. Uniform tag sequences. *Math. Systems Theory*, 6:164–192, 1972.

[Con71] John H. Conway. *Regular Algebra and Finite Machines.* Printed in GB by William Clowes & Sons Ltd, 1971.

[CS63] Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In Paul Braffort and David Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North-Holland, 1963.

[DI06] Hal Daumé III. Yet another haskell tutorial, 2006.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata.* Springer, 2009.

[DvE04] Kees Doets and Jan van Eijck. *The Haskell Road to Logic, Maths And Programming.* College Publications, London, 2004.

[Eil76] Samuel Eilenberg. *Automata, Languages, and Machines.* Academic Press, Inc., 1976.

[ÉL05] Zoltán Ésik and Hans Leiß. Algebraically complete semirings and Greibach normal form. *Annals of Pure and Applied Logic*, 133(1-3):173–203, 2005.

[FD97] James Allen Fill and Robert P. Dobrow. The number of m-ary search trees on n keys. *Combinatorics, Probability & Computing*, 6(4):435–453, 1997.

[Fli74] Michel Fliess. Sur divers produits de séries formelles. *Bulletin de la S.M.F.*, 102:181–191, 1974.

[FR83] Michel Fliess and Cristophe Reutenauer. Théorie de Picard-Vossiot des systèmes réguliers (ou bilinéaires). In *Mathematical tools and models for control, systems analysis and signal processing, Vol. 3*, pages 557–581. CNRS, 1983.

[Fur67] Harry Furstenberg. Algebraic functions over finite fields. *Journal of Algebra*, 7(2):271–277, 1967.

[GEH$^+$12] Clemens Grabmayer, Jörg Endrullis, Dimitri Hendriks, Jan Willem Klop, and Lawrence S. Moss. Automatic sequences and zip-specifications. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 335–344, 2012.

[GH11] Neil Ghani and Peter Hancock. An algebraic foundation and implementation of induction recursion and indexed induction recursion, 2011. Draft paper.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 1994.

[GL14] Katarzyna Grygiel and Pierre Lescanne. Counting terms in the binary lambda calculus. *CoRR*, abs/1401.0379, 2014.

[GMS14] Sergey Goncharov, Stefan Milius, and Alexandra Silva. Towards a coalgebraic Chomsky hierarchy. *CoRR*, abs/1401.5277, 2014.

[Gre65] Sheila A. Greibach. A new normal-form theorem for context-free, phrase structure grammars. *Journal of the Association for Computing Machinery*, 12:42–52, 1965.

[HF92] P. Hudak and J. Fasel. A gentle introduction to Haskell. *ACM SIGPLAN Notices*, 27(5), May 1992.

[Hin11] Ralf Hinze. Concrete stream calculus—an extended study. *Journal of Functional Programming*, 20(5-6):463–535, 2011.

[HKRW14] Helle H. Hansen, Clemens Kupke, Jan Rutten, and Joost Winter. A final coalgebra for k-regular sequences, 2014. Accepted for publication in Prakash Panangaden's festschrift.

[HM13] Reiko Heckel and Stefan Milius, editors. *Algebra and Coalgebra in Computer Science—5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, volume 8089 of *Lecture Notes in Computer Science*. Springer, 2013.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd edition)*. Addison-Wesley, 2006.

[Jac06] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Essays Dedicated to Joseph A. Goguen*, pages 375–404, 2006.

[JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[Kli11] Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412(38):5043–5069, 2011.

[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.

[KPS14] Robbert Krebbers, Louis Parland, and Alexandra Silva. Moessner's theorem: an exercise in coinductive reasoning in Coq, 2014. Draft paper.

[KR12] Clemens Kupke and Jan Rutten. On the final coalgebra of automatic sequences. In Robert L. Constable and Alexandra Silva, editors, *Logic and Program Semantics*, volume 7230 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2012.

[LB99] Saunders Mac Lane and Garrett B. Birkhoff. *Algebra*. AMS Chelsea Pub., 1999.

[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, *CALCO*, volume 5728 of *Lecture Notes in Computer Science*, pages 433–442. Springer, 2009.

[Mac71] Saunders MacLane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[McI99] M. Douglas McIlroy. Power series, power serious. *Journal of Functional Programming*, 9(3):325–337, May 1999.

[McI01] M. Douglas McIlroy. The music of streams. *Inf. Process. Lett.*, 77(2-4):189–195, 2001.

[Mil10] Stefan Milius. A sound and complete calculus for finite stream circuits. In *LICS*, pages 421–430. IEEE Computer Society, 2010.

[Moe51] A. Moessner. Eine Bemerkung über die Potenzen der natürlichen Zahlen. *Sitzungsberichten der Bayerischen Akademie der Wissenschaften, Matematischnaturwissenschaftlische Klasse 1952*, 29, March 1951.

[NR10] Milad Niqui and Jan Rutten. Sampling, splitting and merging in coinductive stream calculus. In Claude Bolduc, Jules Desharnais, and Béchir Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *Lecture Notes in Computer Science*, pages 310–330. Springer, 2010.

[NR11] Milad Niqui and Jan J. M. M. Rutten. A proof of moessner's theorem by coinduction. *Higher-Order and Symbolic Computation*, 24(3):191–206, 2011.

[OGS08] Bryan O'Sullivan, John Goerzen, and Don Stewart. *Real World Haskell.* O'Reilly Media, Inc., 1st edition, 2008.

[Plo92] Simon Plouffe. Approximations de séries génératrices et quelques conjectures. Master's thesis, Université du Québec à Montréal, 1992.

[Pou13] Damien Pous. Coalgebraic up-to techniques. In Heckel and Milius [HM13], pages 34–35.

[PS09] Ion Petre and Arto Salomaa. Algebraic systems and pushdown automata. In Droste et al. [DKV09], pages 257–289.

[RBR13] Jurriaan Rot, Marcello M. Bonsangue, and Jan Rutten. Coalgebraic bisimulation-up-to. In Peter van Emde Boas, Frans C. A. Groen, Giuseppe F. Italiano, Jerzy R. Nawrocki, and Harald Sack, editors, *SOFSEM*, volume 7741 of *Lecture Notes in Computer Science*, pages 369–381. Springer, 2013.

[Ros67] Daniel J. Rosenkrantz. Matrix equations and normal forms for context-free grammars. *J. ACM*, 14(3):501–507, 1967.

[Rut98] Jan Rutten. Automata and coinduction (an exercise in coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998.

[Rut99] Jan Rutten. Automata, power series, and coinduction: Taking input derivatives seriously. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 645–654. Springer, 1999.

[Rut00] Jan Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[Rut02] Jan Rutten. Coinductive counting: bisimulation in enumerative combinatorics. *Electronic Notes in Theoretical Computer Science*, 65(1):286–304, 2002.

[Rut03a] Jan Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1-3):1–53, 2003.

[Rut03b] Jan Rutten. Coinductive counting with weighted automata. *Journal of Automata, Languages and Combinatorics*, 8(2):319–352, 2003.

[Rut05] Jan Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.

[Rut08] Jan Rutten. Rational streams coalgebraically. *Logical Methods in Computer Science*, 4(3), 2008.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[SBBR10] Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan Rutten. Generalizing the powerset construction, coalgebraically. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 272–283. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2010.

[SBBR13] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013.

[SBR10] Alexandra Silva, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Non-deterministic Kleene coalgebras. *Logical Methods in Computer Science*, 6(3), 2010.

[Sch61a] Marcel-Paul Schützenberger. On a theorem of R. Jungen. *Proceedings of the American Mathematical Society*, 13:885–889, 1961.

[Sch61b] Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[Sil10] Alexandra Silva. *Kleene Coalgebra.* PhD thesis, Radboud Universiteit Nijmegen, 2010.

[SS78] Arto Salomaa and Matti Soittola. *Automata-theoretic aspects of formal power series.* Texts and monographs in computer science. Springer, 1978.

[Ste03] Ralf Stephan. Divide-and-conquer generating functions. part I. Elementary sequences. *ArXiv Mathematics e-prints*, jul 2003.

[TP97] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291. IEEE Computer Society, 1997.

[WBR11] Joost Winter, Marcello M. Bonsangue, and Jan Rutten. Context-free languages, coalgebraically. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 359–376. Springer, 2011.

[WBR13] Joost Winter, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Coalgebraic characterizations of context-free languages. *Logical Methods in Computer Science*, 9 (3:14), 2013.

[WBR14] Joost Winter, Marcello M. Bonsangue, and Jan Rutten. Context-free coalgebras, 2014. Accepted for publication in the Journal of Computer and System Sciences.

[Wil06] Herbert S. Wilf. *Generatingfunctionology.* A. K. Peters, Ltd., 2006.

[Win13] Joost Winter. QStream: a suite of streams. In Heckel and Milius [HM13], pages 353–358.

[Wor09] James Worthington. *Automata, Representations, and Proofs*. PhD thesis, Cornell University, 2009.

[ZE11] Hans Zantema and Jörg Endrullis. Proving equality of streams automatically. In Manfred Schmidt-Schauß, editor, *RTA*, volume 10 of *LIPIcs*, pages 393–408. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2011.

Titles in the IPA Dissertation Series since 2008

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras.* Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit.* Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks.* Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments.* Faculty of

Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design.* Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems.* Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation.* Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation.* Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery*

and Provisioning. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

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