

# Kleene coalgebra

Alexandra Silva

**Promotor:**

Prof. Dr. Jan Rutten

**Copromotor:**

Dr. Marcello Bonsangue

**Manuscriptcommissie:**

Dr. Luís Barbosa	Universidade do Minho, Braga, Portugal
Prof. Dr. Herman Geuvers	Radboud University Nijmegen, The Netherlands
Prof. Dr. Bart Jacobs	Radboud University Nijmegen, The Netherlands
Prof. Dr. Dexter Kozen	Cornell University, Ithaca, New York, USA
Dr. Erik de Vink	Technical University Eindhoven, The Netherlands



## CONTENTS

<b>Preface</b>	<b>iii</b>
<b>Contents</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Coalgebra . . . . .	1
1.2 Kleene . . . . .	2
1.3 . . . and followers . . . . .	2
1.4 Our aim . . . . .	3
1.5 Kleene coalgebra . . . . .	3
1.6 Thesis outline and summary of the contributions . . . . .	5
1.7 Related work . . . . .	6
<b>2 Preliminaries</b>	<b>9</b>
2.1 Sets . . . . .	9
2.2 Coalgebras . . . . .	10
<b>3 Automata as coalgebras</b>	<b>15</b>
3.1 Deterministic automata and regular expressions . . . . .	15
3.1.1 From deterministic automata to regular expressions . . . . .	21
3.1.2 From regular expressions to deterministic automata . . . . .	23
3.1.3 Non-deterministic automata and the subset construction . . . . .	27
3.1.4 Kleene algebras . . . . .	31
3.2 Automata on guarded strings and KAT expressions . . . . .	39
<b>4 Kleene meets Mealy</b>	<b>47</b>
4.1 Mealy machines . . . . .	48
4.2 Regular expressions for Mealy machines . . . . .	51
4.2.1 Expressions form a Mealy coalgebra . . . . .	52
4.2.2 A Kleene theorem for Mealy coalgebras . . . . .	54
4.3 An algebra for Mealy machines . . . . .	61
4.4 Discussion . . . . .	67
<b>5 Non-deterministic Kleene coalgebras</b>	<b>69</b>
5.1 Non-deterministic coalgebras . . . . .	70
5.2 A language of expressions for non-deterministic coalgebras . . . . .	71
5.2.1 Brzowski derivatives for non-deterministic expressions . . . . .	75

5.2.2	From coalgebras to expressions . . . . .	78
5.2.3	From expressions to coalgebras . . . . .	83
5.3	A sound and complete axiomatization . . . . .	90
5.4	Two more examples . . . . .	101
5.5	Polynomial and finitary coalgebras . . . . .	103
5.6	Discussion . . . . .	107
<b>6</b>	<b>Quantitative Kleene coalgebras</b>	<b>111</b>
6.1	The monoidal exponentiation functor . . . . .	114
6.2	A non-idempotent algebra for quantitative regular behaviors . . . . .	118
6.3	Extending the class of functors . . . . .	132
6.4	Probabilistic systems . . . . .	135
6.5	A slight variation on the functor . . . . .	138
6.6	Discussion . . . . .	139
<b>7</b>	<b>Further directions</b>	<b>141</b>
	<b>Bibliography</b>	<b>143</b>



Computer systems have widely spread since their appearance, and they now play a crucial role in many daily activities, with their deployment ranging from small home appliances to safety critical components, such as airplane or automobile control systems. Accidents caused by either hardware or software failure can have disastrous consequences, leading to the loss of human lives or causing enormous financial drawbacks. One of the greatest challenges of computer science is to cope with the fast evolution of computer systems and develop formal techniques which facilitate the construction of safe software and hardware systems.

Since the early days of computer science, many scientists have searched for suitable models of computation and for specification languages that are appropriate for reasoning about such models. When devising a model for a particular system, there is a need to encode different features which capture the inherent behavior of the system. For instance, some systems have deterministic behavior (a calculator or an elevator), whereas others have inherently non-deterministic or probabilistic behavior (think of a casino slot machine). The rapidly increasing complexity of systems demands for compositional and unifying models of computation, as well as general methods and guidelines to derive specification languages.

### 1.1 Coalgebra

In the last decades, coalgebra has arisen as a prominent candidate for a mathematical framework to specify and reason about computer systems. Coalgebraic modeling works, on the surface, as follows: the basic features of a system, such as non-determinism or probability, are collected and combined in the appropriate way, determining the type of the system. This type (formally, a functor) is then used to derive a suitable equivalence relation and a universal domain of behaviors, which allow to reason about equivalence of systems. The strength of coalgebraic modeling lies in the fact that many important notions are parametrized by the type of the system. On the one hand, the coalgebraic framework is unifying, allowing for a uniform study of different systems and making precise the connection between them. On the other

hand, it can serve as a guideline to the development of basic notions for new models of computation.

## 1.2 Kleene ...

One of the simplest models of computation is that of a deterministic finite automaton. In his seminal paper in 1956, Kleene [64] described finite deterministic automata (which he called nerve nets), together with a specification language: regular expressions. One of his most important results is the theorem which states that any finite deterministic automaton can be characterized by a regular expression and that, conversely, every regular expression can be realized by such automaton. This theorem, which is today referred to as *Kleene's theorem*, became one of the cornerstones of theoretical computer science.

## 1.3 ... and followers

In his paper, Kleene left open the question of whether there would exist a finite, sound and complete, axiomatization of the equivalence of regular expressions, which would enable algebraic reasoning. The first answer to this question was given in 1966 by Salomaa [101], who presented two complete axiomatizations. The 1971 monograph of Conway [39] presents an extended overview of results on regular expressions and their axiomatizations. Later, in 1990, Kozen [66] showed that Salomaa's axiomatization is non-algebraic, in the sense that it is unsound under substitution of alphabet symbols by arbitrary regular expressions, and presented an algebraic axiomatization: Kleene algebras.

McNaughton and Yamada [80] gave algorithms to build a non-deterministic automaton from a regular expression and back, and introduced a notion of extended regular expression with intersection and complementation operators. This enrichment of the language of regular expressions was relevant in the context of the most important application of regular expressions at that time, in the design of digital circuits, allowing a more easy conversion of a natural language specification of problems into a regular expression.

Brzozowski [28,29] introduced the notion of derivative of regular expressions, which allowed him to prove Kleene's theorem for the extended set of regular expressions without having to recur to non-deterministic automata.

Efficient algorithms to compile regular expressions to deterministic and non-deterministic automata became crucial when regular expressions started to be widely used for pattern matching. One of the fastest (and most beautiful) algorithms to translate regular expressions into automata was devised by Berry and Sethi [19]. This algorithm became the basis of one of the first compilers of the language Esterel [18], a synchronous programming language, based on regular expressions, dedicated to embedded systems. Esterel is one of the most successful follow ups of Kleene's work: it



is used as a specification language of control-intensive applications, such as the ones running in the central units of cars or airplanes. In such systems guaranteeing important safety properties is of the uttermost importance and, hence, formal models of computation play a central role.

In 1981, Milner adapted Kleene and Salomaa's results to labeled transition systems: a model of computation in which non-determinism is allowed [84]. He introduced a language for finitely presented behaviors, which can be seen as a fragment of the calculus of communicating systems (CCS) [83], and a sound and complete axiomatization with respect to bisimilarity. The paper of Milner served as inspiration for many researchers in the concurrency community. Probabilistic extensions of CCS, together with sound and complete axiomatizations (with respect to the appropriate notion of equivalence) have been presented for instance in [41, 42, 106].

In addition to the models already mentioned, other important models of computation include Mealy machines [81] (automata with input and output), automata on guarded strings [69] and weighted automata [104]. These three models have applications, for instance, in digital circuit design, compiler optimization and image recognition, respectively.

## 1.4 Our aim

The aim of this thesis is to make use of the coalgebraic view on systems to devise a framework where languages of specification and axiomatizations can be uniformly derived for a large class of systems, which include all the models mentioned above. As a sanity check, it should be possible to derive from the general framework known results. More importantly, we should be able to derive new languages and axiomatizations.

## 1.5 Kleene coalgebra

In this thesis, we combine the work of Kleene with coalgebra. The theory of universal coalgebra [96] provides a standard equivalence and a universal domain of behaviors, uniquely based on the type of the system, given by a functor  $\mathcal{F}$ . It is our main aim to show how the type of the system also allows for a uniform derivation of both a set of expressions describing the system's behavior and a corresponding axiomatization, sound and complete with respect to the equivalence induced by  $\mathcal{F}$ , which enables algebraic reasoning on the specifications. Furthermore, we want to show the correspondence of the behaviors denoted by the expressions in the language and the systems under study, formulating the coalgebraic analogue of Kleene's theorem.

The class of systems we study (or in other words, the class of functors  $\mathcal{F}$ ) is large enough to cover finite deterministic automata and labeled transition systems. It includes also other models, such as Mealy machines, automata on guarded strings, weighted automata and several types of probabilistic automata, such as Segala, stratified and Pnueli-Zuck systems. From this general framework we will recover known

languages and axiomatizations but, more interestingly, we will also derive new ones, for the so-called stratified and Pnueli-Zuck systems.

To give the reader a feeling of the type of expressions and axiomatizations we will derive, we show in Figure 1.1 a few examples of systems, together with their type and examples of valid expressions and axioms.

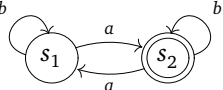
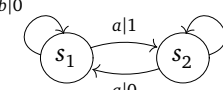
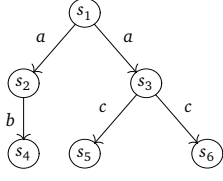
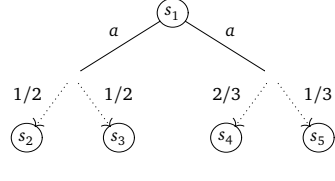
Deterministic automata	Mealy machines
	
$\mathcal{F} = 2 \times \text{Id}^A$	$\mathcal{F} = (2 \times \text{Id})^A$
$\mu x.b(x) \oplus a(\mu y.b(y) \oplus a(x) \oplus 1)$	$\mu x.b(x) \oplus a(\mu y.a(x) \oplus b(y)) \oplus a \downarrow 1$
$\emptyset \oplus \varepsilon \equiv \varepsilon$	$\mu x.\varepsilon \equiv \varepsilon[\mu x.\varepsilon/x]$
Labeled transition systems	Segala systems
	
$\mathcal{F} = (\mathcal{P}_\omega \text{Id})^A$	$\mathcal{F} = (\mathcal{P}_\omega(\mathcal{D}_\omega \text{Id}))^A$
$a(\{b(\emptyset)\} \oplus \{c(\{\emptyset\} \oplus \{\emptyset\})\})$	$a(\{1/2 \cdot \emptyset \oplus 1/2 \cdot \emptyset\}) \boxplus a(\{1/3 \cdot \emptyset \oplus 2/3 \cdot \emptyset\})$
$\varepsilon \oplus \varepsilon \equiv \varepsilon$	$p \cdot \varepsilon \oplus p' \cdot \varepsilon \equiv (p + p') \cdot \varepsilon$

Figure 1.1: For each of the systems we show a concrete example, the corresponding functor type, an expression describing the behavior of  $s_1$ , and an example of a valid axiom.

## 1.6 Thesis outline and summary of the contributions

We summarize below the content and main contributions of each chapter.

**Chapter 2** introduces basic preliminaries on coalgebra.

**Chapter 3** contains material already existing in the literature, collecting the main results on regular expressions and Kleene algebras, mainly due to Kleene, Brzozowski, Kozen and Rutten. These results serve as basis for the generalizations presented in the subsequent chapters. Furthermore, we recall the basics of Kleene algebra with tests (KAT), an extension, due to Kozen, of the algebra of regular expressions with Boolean tests. KAT has many applications in compiler optimization, program transformation and dataflow analysis.

We also recall the Berry-Sethi construction, from regular expressions to non-deterministic automata, mentioned previously in the introduction, and we show how to generalize it to KAT expressions. This generalization constitutes the original contribution of this chapter.

**Chapter 4** introduces a language and axiomatization for Mealy machines. The main contributions are summarized in the table below.

Kleene meets Mealy	
Specification language for Mealy machines	Definition 4.2.1
An analogue of Kleene's theorem	Theorems 4.2.7 and 4.2.8
Sound and complete axiomatization	Section 4.3

The results in this chapter are subsumed by the ones in the subsequent chapter, but they can be seen as a concrete example of the coalgebraic approach, illustrating many of the general principles. This chapter paves the way for the construction of the general framework which we will present in the subsequent chapters.

This chapter is based on the following paper:

*Marcello M. Bonsangue, Jan J. M. M. Rutten, Alexandra Silva: Coalgebraic Logic and Synthesis of Mealy Machines. FoSSaCS 2008:231-245.*

**Chapter 5** contains the development of a general framework, parametrized by the type of the system, a functor, where languages and axiomatizations can be derived uniformly. The class of functors considered, which we call *non-deterministic* functors, covers a number of models, such as deterministic automata, labeled transition systems and automata on guarded strings. We summarize the main contributions of the chapter in the table below.

Non-deterministic Kleene coalgebras	
Specification language for non-deterministic coalgebras	Definition 5.2.1
An analogue of Brzozowski derivatives	Definition 5.2.11
An analogue of Kleene's theorem	Theorems 5.2.12 and 5.2.14
Sound and complete axiomatization	Section 5.3

This chapter is based on the following papers:

*Marcello M. Bonsangue, Jan J. M. M. Rutten, Alexandra Silva: A Kleene Theorem for Polynomial Coalgebras. FOSSACS 2009:122-136.*

*Marcello M. Bonsangue, Jan J. M. M. Rutten, Alexandra Silva: An Algebra for Kripke Polynomial Coalgebras. LICS 2009:49-58.*

*Alexandra Silva, Marcello M. Bonsangue, Jan J. M. M. Rutten: Non-deterministic Kleene coalgebras. Accepted for publication in Logical Methods in Computer Science.*

**Chapter 6** extends the framework of the previous chapter to be able to deal with *quantitative systems* such as weighted or probabilistic automata. The main technical challenge is that quantitative systems have an inherently non-idempotent behavior and thus the proof of Kleene's theorem and the axiomatization require extra care. In this chapter, the generality of our approach pays off: we were able to derive a sound and complete axiomatization for stratified systems, a type of probabilistic automata for which a language existed but no axiomatization, and a language and axiomatization for Pnueli-Zuck systems, another type of probabilistic automata for which only a modal logic like language existed. The main contributions of the chapter are summarized in the following table.

Quantitative Kleene coalgebras	
Specification language for quantitative systems	Definition 6.2.3
An analogue of Kleene's theorem	Theorem 6.2.9
Sound and complete axiomatization	Theorems 6.2.11 and 6.2.17
Examples of application to probabilistic systems	Section 6.4

This chapter based on the following papers:

*Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, Alexandra Silva: Deriving Syntax and Axioms for Quantitative Regular Behaviours. CONCUR 2009:146-162.*

*Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, Alexandra Silva: Quantitative Kleene coalgebras. Accepted for publication in Information and Computation.*

## 1.7 Related work

The connection between Kleene's regular expressions, deterministic automata and coalgebra was first explored in [95, 97]. Rutten explored the coalgebraic structure

of the set of regular expressions, given by Brzozowski derivatives [29], in order to show that the coalgebraic semantics coincides with the standard inductive semantics of regular expressions. He proved the usefulness of the approach by proving equalities by coinduction. The coinductive proofs turned out to be, in many cases, more concise and intuitive than the alternative algebraic proof using the axioms of Kleene algebra. Later, Jacobs [61] presented a bialgebraic review on deterministic automata and regular expressions, which allowed him to present an alternative coalgebraic proof of Kozen's result on the completeness of Kleene algebras for language equivalence [66, 69]. We took inspiration from all of these papers: the work of Brzozowski and Rutten led us to the definition of the coalgebraic structure on the set of expressions, whereas the work by Jacobs and Kozen served as a guideline to the proof of soundness and completeness of the axiomatization we will introduce for the set of generalized regular expressions.

In the last few years several proposals for specification languages for coalgebras appeared [24, 25, 36, 50, 60, 75, 86, 94, 103]. The languages presented in this thesis are similar in spirit to that of Rössiger [94], Jacobs [60], Pattinson and Schröder [103] in that we use the ingredients of a functor for typing expressions. They differ from the logics presented in [60, 94] because we do not need an explicit "next-state" operator, as we can deduce it from the type information.

Apart from the logics introduced by Kupke and Venema in [75], the languages mentioned above do not include fixed point operators. Our language of generalized regular expressions is similar to a fragment of the logic presented in [75] and can be seen as an extension of the coalgebraic logic of [24] with fixed point operators, as well as the multi-sorted logics of [103]. However, our goal is rather different: we want (1) a finitary language that characterizes exactly all *locally finite* coalgebras; (2) a Kleene like theorem for the language or, in other words, a *map* (and not a relation) from expressions to coalgebras and vice-versa. Similar to many of the works above, we also derive a modular axiomatization, sound and complete with respect to the equivalence induced by the functor.

The languages studied in this thesis allow for recursive specifications and therefore formalize potentially infinite computations. This type of computations were studied also in the context of iterative theories, which have been introduced by Elgot [45]. The main example of an iterative theory is the theory of regular trees, that is trees which have finitely many distinct subtrees. Adámek, Milius and Velebil have presented Elgot's work from a coalgebraic perspective [5, 6], simplified some of his original proofs, and generalized the notion of free iterative theory to any finitary endofunctor of every locally presentable category. The language associated with each functor, which we introduce in this thesis, modulo the axioms is closely related to the work above: it is an initial iterative algebra. This also shows the connection of our work with the work by Bloom and Ésik on iterative algebras/theories [20].

Kleene's theorem has been extended in various ways. Büchi [32] extended it to infinite words and  $\omega$ -automata, introducing an  $\omega$  operator on languages. Ochmanski [87] introduced a concurrent version of the Kleene star operator, which lead him to define a notion of co-rational languages, obtained as the rational ones by simply

replacing the star by the concurrent iteration. He then generalized Kleene's theorem showing that the recognizable trace languages are exactly the co-rational languages. Gastin, Petit and Zielonka [46, 47] extended Ochmanski's results to infinite trace languages. For weighted automata, Schützenberger [104] has shown that the set of recognizable formal power series (corresponding to the behavior of weighted automata) coincides with the set of rational formal power series. For timed automata, there were several proposals, including the papers by Bouyer and Petit [27], Asarin, Caspi and Maler [11], and Asarin and Dima [12]. Recently, the results of Bouyer and Petit as well as those of Schützenberger have been extended to the class of weighted timed automata by Droste and Quaas [44]. Furthermore, Kozen has extended Kleene's language with Boolean tests as a finitary representation of regular sets of guarded strings and proved an analogue of Kleene's theorem for automata on guarded strings [69]. From the aforementioned extensions, only the weighted automata example of Schützenberger and the automata on guarded strings of Kozen would fit in the framework we will present in this thesis. The language we will derive is, however, different from the ones they proposed. Schützenberger's and Kozen's languages had full sequential composition and star in their syntax, instead of the action prefixing and unique fixed point operators that we will use in our language.

We give the basic definitions on functors and coalgebras and introduce the notion of bisimulation. We assume the reader is familiar with basic categorical concepts such as category, functor and natural transformation.

### 2.1 Sets

First we fix notation on sets and operations on them. Let **Set** be the category of sets and functions. Sets are denoted by capital letters  $X, Y, \dots$  and functions by lower case  $f, g, \dots$ . We write  $\emptyset$  for the empty set and the collection of all *finite* subsets of a set  $X$  is defined as  $\mathcal{P}_\omega(X) = \{Y \subseteq X \mid Y \text{ finite}\}$ . The collection of functions from a set  $X$  to a set  $Y$  is denoted by  $Y^X$ . We write  $id_X$  for the identity function on set  $X$ . Given functions  $f: X \rightarrow Y$  and  $g: Y \rightarrow Z$  we write their composition as  $g \circ f$ . The product of two sets  $X, Y$  is written as  $X \times Y$ , with projection functions  $X \times Y \xrightarrow{\pi_1} X \xrightarrow{\pi_2} Y$ . The set  $1$  is a singleton set typically written as  $1 = \{*\}$  and it can be regarded as the empty product. We define

$$X \diamond Y = (X \uplus Y) \cup \{\perp, \top\}$$

where  $\uplus$  is the disjoint union of sets, with injections  $X \xrightarrow{\kappa_1} X \uplus Y \xleftarrow{\kappa_2} Y$ , and  $\perp$  and  $\top$  are distinct from the elements of  $X \uplus Y$ . Note that the set  $X \diamond Y$  is different from the classical coproduct of  $X$  and  $Y$  (which we shall denote by  $X + Y$ ), because of the two extra elements  $\perp$  and  $\top$ . These extra elements will later be used to represent, respectively, underspecification and inconsistency in the specification of systems. For each of the operations defined above on sets, there are analogous ones on func-

tions. Let  $f : X \rightarrow Y$ ,  $f_1 : X \rightarrow Y$  and  $f_2 : Z \rightarrow W$ . We define the following operations:

$$\begin{aligned} f_1 \times f_2 : X \times Z &\rightarrow Y \times W & f_1 \oplus f_2 : X \oplus Z &\rightarrow Y \oplus W \\ (f_1 \times f_2)((x, z)) &= (f_1(x), f_2(z)) & (f_1 \oplus f_2)(c) &= c, c \in \{\perp, \top\} \\ & & (f_1 \oplus f_2)(\kappa_i(x)) &= \kappa_i(f_i(x)), i \in \{1, 2\} \end{aligned}$$

$$\begin{aligned} f^A : X^A &\rightarrow Y^A & \mathcal{P}_\omega(f) : \mathcal{P}_\omega(X) &\rightarrow \mathcal{P}_\omega(Y) \\ f^A(g) &= f \circ g & \mathcal{P}_\omega(f)(S) &= \{f(x) \mid x \in S\} \end{aligned}$$

Note that here we are using the same symbols that we defined above for the operations on sets. It will always be clear from the context which operation is being used. In our definition of non-deterministic functors we will use constant sets equipped with an information order. In particular, we will use join-semilattices. A (bounded) join-semilattice is a set  $B$  equipped with a binary operation  $\vee_B$  and a constant  $\perp_B \in B$ , such that  $\vee_B$  is commutative, associative and idempotent. The element  $\perp_B$  is neutral with respect to  $\vee_B$ . As usual,  $\vee_B$  gives rise to a partial ordering  $\leq_B$  on the elements of  $B$ :  $b_1 \leq_B b_2 \Leftrightarrow b_1 \vee_B b_2 = b_2$ . Every set  $S$  gives rise to a join-semilattice by taking  $B$  to be the set of all finite subsets of  $S$  with union as join.

## 2.2 Coalgebras

An  $\mathcal{F}$ -coalgebra is a pair  $(S, f : S \rightarrow \mathcal{F}(S))$ , where  $S$  is a set of states and  $\mathcal{F} : \mathbf{Set} \rightarrow \mathbf{Set}$  is a functor. The functor  $\mathcal{F}$ , together with the function  $f$ , determines the *transition structure* (or dynamics) of the  $\mathcal{F}$ -coalgebra [96].

An  $\mathcal{F}$ -homomorphism  $h : (S, f) \rightarrow (T, g)$ , from a  $\mathcal{F}$ -coalgebra  $(S, f)$  to a  $\mathcal{F}$ -coalgebra  $(T, g)$ , is a function  $h : S \rightarrow T$  preserving the transition structure, *i.e.*, such that the following diagram commute:

$$\begin{array}{ccc} S & \xrightarrow{h} & T \\ f \downarrow & & \downarrow g \\ \mathcal{F}(S) & \xrightarrow{\mathcal{F}(h)} & \mathcal{F}(T) \end{array} \quad g \circ h = \mathcal{F}(h) \circ f$$

**2.2.1 DEFINITION** (Final coalgebra). An  $\mathcal{F}$ -coalgebra  $(\Omega, \omega)$  is said to be *final* if for any  $\mathcal{F}$ -coalgebra  $(S, f)$  there exists a unique  $\mathcal{F}$ -homomorphism  $\mathbf{beh}_S : (S, f) \rightarrow (\Omega, \omega)$ :

$$\begin{array}{ccc} S & \xrightarrow{\mathbf{beh}_S} & \Omega \\ f \downarrow & & \downarrow \omega \\ \mathcal{F}(S) & \xrightarrow{\mathcal{F}(\mathbf{beh}_S)} & \mathcal{F}(\Omega) \end{array} \quad \omega \circ \mathbf{beh}_S = \mathcal{F}(\mathbf{beh}_S) \circ f$$





The notion of finality will play a key role later in providing semantics to the expressions we will associate with functors in subsequent chapters. For that reason, it is important to characterize a class of functors for which final coalgebras exist.

A functor is said to be bounded [53, Theorem 4.7] if there exists a natural surjection  $\eta$  from a functor  $B \times (-)^A$  to  $\mathcal{F}$ , for some sets  $B$  and  $A$ . For every bounded functor there exists a final  $\mathcal{F}$ -coalgebra  $(\Omega_{\mathcal{F}}, \omega_{\mathcal{F}})$  [53, 96].

**2.2.2 DEFINITION (Subcoalgebra).** Given an  $\mathcal{F}$ -coalgebra  $(S, f)$  and a subset  $V$  of  $S$  with inclusion map  $i: V \rightarrow S$  we say that  $V$  is a *subcoalgebra* of  $S$  if there exists  $g: V \rightarrow \mathcal{F}(V)$  such that  $i$  is a  $\mathcal{F}$ -homomorphism:

$$\begin{array}{ccc}
 V & \xrightarrow{i} & S \\
 \downarrow g & & \downarrow f \\
 \mathcal{F}(V) & \xrightarrow{\mathcal{F}(i)} & \mathcal{F}(S)
 \end{array}
 \qquad f \circ i = \mathcal{F}(i) \circ g$$



The transition structure  $t$  is unique (this is a consequence of the fact that all **Set** functors preserve monos):

**2.2.3 THEOREM ([96, Proposition 6.1]).** *Let  $(S, f)$  be an  $\mathcal{F}$ -coalgebra and let  $V$  be a subset of  $S$  with inclusion map  $i: V \rightarrow S$ . If  $k, l: V \rightarrow \mathcal{F}(V)$  are such that  $i$  is an  $\mathcal{F}$ -homomorphism both from  $(V, k)$  to  $(S, f)$  and from  $(V, l)$  to  $(S, f)$ , then  $k = l$ .*

Given  $s \in S$ ,  $\langle s \rangle = (T, t)$  denotes the smallest subcoalgebra generated by  $s$ , with  $T$  given by

$$T = \bigcap \{V \mid V \text{ is a subcoalgebra of } S \text{ and } s \in V\} \quad (2.1)$$

If the functor  $\mathcal{F}$  preserves arbitrary intersections, then the subcoalgebra  $\langle s \rangle$  exists. This will be the case for every functor considered in this thesis.

We will write  $\text{Coalg}(\mathcal{F})$  for the category of  $\mathcal{F}$ -coalgebras together with coalgebra homomorphisms. We also write  $\text{Coalg}_{\text{LF}}(\mathcal{F})$  for the category of  $\mathcal{F}$ -coalgebras that are *locally finite*:  $\mathcal{F}$ -coalgebras  $(S, f)$  such that for each state  $s \in S$  the state space of the generated subcoalgebra  $\langle s \rangle$  is finite.

**2.2.4 DEFINITION (Bisimulation).** Let  $(S, f)$  and  $(T, g)$  be two  $\mathcal{F}$ -coalgebras. A relation  $R \subseteq S \times T$  is called a *bisimulation* [3] if there exists a map  $e: R \rightarrow \mathcal{F}(R)$  such that the projections  $\pi_1$  and  $\pi_2$  are coalgebra homomorphisms, *i.e.* the following diagram

commutes.

$$\begin{array}{ccccc}
 S & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & T \\
 f \downarrow & & \exists e \downarrow & & \downarrow g \\
 \mathcal{F}(S) & \xleftarrow{\mathcal{F}(\pi_2)} & \mathcal{F}(R) & \xrightarrow{\mathcal{F}(\pi_1)} & \mathcal{F}(T)
 \end{array}$$

This is equivalent to the following alternative definition, which is sometimes more convenient in proofs. A relation  $R \subseteq S \times T$  a *bisimulation* [58] iff

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

where  $\overline{\mathcal{F}(R)}$  is defined as  $\overline{\mathcal{F}(R)} = \{(\mathcal{F}(\pi_1)(x), \mathcal{F}(\pi_2)(x)) \mid x \in \mathcal{F}(R)\}$ .  $\clubsuit$

We write  $s \sim_{\mathcal{F}} t$  whenever there exists a bisimulation relation containing  $(s, t)$  and we call  $\sim_{\mathcal{F}}$  the bisimilarity relation. We shall drop the subscript  $\mathcal{F}$  whenever the functor  $\mathcal{F}$  is clear from the context.

**2.2.5 DEFINITION** (Behavioral equivalence). Let  $(S, f)$  and  $(T, g)$  be  $\mathcal{F}$ -coalgebras. We say that the states  $s \in S$  and  $t \in T$  are *behaviorally equivalent*, written  $s \sim_b t$ , if and only if they are mapped into the same element in the final coalgebra, that is  $\mathbf{beh}_S(s) = \mathbf{beh}_T(t)$ .  $\clubsuit$

If two states are bisimilar then they are behaviorally equivalent ( $s \sim t \Rightarrow s \sim_b t$ ). The converse implication is only true for certain classes of functors. For instance, if the functor  $\mathcal{F}$  preserves weak-pullbacks then we also have  $s \sim_b t \Rightarrow s \sim t$ .

All the functors considered in this thesis, except the ones in Chapter 6, satisfy the above property.

The main use of bisimulations is their application as a proof principle.

**2.2.6 THEOREM** (Coinduction). *If  $(\Omega, \omega)$  is a final coalgebra and  $a, b \in \Omega$ , then*

$$a \sim b \Rightarrow a = b$$

PROOF. Let  $\mathcal{F}$  be a functor with final coalgebra  $(\Omega, \omega)$  and let  $a, b \in \Omega$  satisfying  $a \sim b$ . There exists a bisimulation  $R$ , with  $\langle a, b \rangle \in R$  such that the following diagram commutes.

$$\begin{array}{ccccc}
 \Omega & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & \Omega \\
 \omega \downarrow & & \exists e \downarrow & & \downarrow \omega \\
 \mathcal{F}(\Omega) & \xleftarrow{\mathcal{F}(\pi_2)} & \mathcal{F}(R) & \xrightarrow{\mathcal{F}(\pi_1)} & \mathcal{F}(\Omega)
 \end{array}$$

By the properties of the final coalgebra there exist unique  $\mathcal{F}$ -homomorphisms  $\mathbf{beh}_\Omega$  and  $\mathbf{beh}_R$  such that:

$$\begin{array}{ccccc}
 & & (\Omega, \omega) & & \\
 & \mathbf{beh}_\Omega \dashrightarrow & \uparrow & \dashleftarrow & \mathbf{beh}_\Omega \\
 & & \mathbf{beh}_R & & \\
 & & \downarrow & & \\
 (\Omega, \omega) & \xleftarrow{\pi_1} & (R, e) & \xrightarrow{\pi_2} & (\Omega, \omega)
 \end{array}$$

Thus,  $\mathbf{beh}_\Omega \circ \pi_1 = \mathbf{beh}_R = \mathbf{beh}_\Omega \circ \pi_2$ , and since  $\mathbf{beh}_\Omega = id_\Omega$ , we can conclude that  $a = b$ .  $\square$

This theorem implies that to prove that two states  $a$  and  $b$  in a final coalgebra are equal it is enough to exhibit a bisimulation  $R$  containing the pair  $\langle a, b \rangle$ . The following two theorems will be useful in the proofs of soundness and completeness we shall present later.

**2.2.7 THEOREM** ([96, Proposition 5.8]). *Let  $S$  be a set and  $R$  an equivalence relation on  $S$ . Then the natural quotient map  $[-]: S \rightarrow S/R$  is the co-equalizer of the projection morphisms  $\pi_1, \pi_2: R \rightarrow S$ . That is, for any other set  $T$  and map  $q: S \rightarrow T$  such that*

$$s_1 R s_2 \Rightarrow q(s_1) = q(s_2)$$

*there exists a unique morphism  $u: S/R \rightarrow T$  such that the following diagram commutes:*

$$\begin{array}{ccc} R & \begin{array}{c} \xrightarrow{\pi_1} \\ \xrightarrow{\pi_2} \end{array} & S \\ & & \searrow q \\ & & S/R \\ & & \downarrow u \\ & & T \end{array}$$

*Moreover if  $S$  has a coalgebra structure  $f: S \rightarrow \mathcal{F}(S)$ , then there exists a unique  $\mathcal{F}$ -coalgebra structure  $\alpha: S/R \rightarrow \mathcal{F}(S/R)$  such that  $[-]$  is a coalgebra homomorphism:*

$$\begin{array}{ccc} R & \begin{array}{c} \xrightarrow{\pi_1} \\ \xrightarrow{\pi_2} \end{array} & S \\ & & \downarrow f \\ & & \mathcal{F}(S) \\ & & \downarrow \mathcal{F}([-]) \\ & & \mathcal{F}(S/R) \end{array} \quad \begin{array}{ccc} & \xrightarrow{[-]} & S/R \\ & & \downarrow \alpha \\ & & \mathcal{F}(S/R) \end{array}$$

**2.2.8 THEOREM** (Epi-mono factorization, [96, Theorem 7.1]). *Given an  $\mathcal{F}$ -homomorphism  $h: (S, f) \rightarrow (T, g)$ , there exists a unique  $\mathcal{F}$ -coalgebra  $(\overline{T}, \overline{g})$  and homomorphisms  $e: (S, f) \rightarrow (\overline{T}, \overline{g})$  and  $m: (\overline{T}, \overline{g}) \rightarrow (T, g)$ :*

$$\begin{array}{ccccc} & & h & & \\ & & \curvearrowright & & \\ S & \xrightarrow{e} & \overline{T} & \xrightarrow{m} & T \\ & & \downarrow \overline{g} & & \downarrow g \\ \mathcal{F}(S) & \xrightarrow{\mathcal{F}(e)} & \mathcal{F}(\overline{T}) & \xrightarrow{\mathcal{F}(m)} & \mathcal{F}(T) \\ & & \downarrow \mathcal{F}(h) & & \\ & & \mathcal{F}(T) & & \end{array}$$

*such that  $m$  is a monomorphism and  $e$  is an epimorphism.*



In this chapter we will illustrate many of the general concepts introduced in the previous chapter in a concrete setting. Thus, in contrast with the universal coalgebra of the previous chapter we will now encounter concrete coalgebra.

The chapter has two parts: the first on deterministic automata, Kleene's regular expressions and Kleene algebras and the second on Kleene algebra with tests (KAT).

Many of the results presented in the first part of the chapter have appeared in the literature (e.g. in [29, 39, 61, 68, 95, 97]). For completeness, we will include many of the proofs of known results and in some cases we will present them in a different way. This will allow us to introduce results and techniques which we will generalize in the subsequent chapters. It should be said upfront that there will be a crucial difference in the generalized languages of regular expressions, which we will introduce later, and Kleene's regular expressions, which we recall in this chapter. In the latter, full sequential composition and star are used, whereas in the languages we shall introduce we opted to have action prefixing and unique fixed point operators.

The results on KAT expressions are mostly due to Kozen [69, 70]. We follow up on those results and present a new automata model for KAT expressions and a construction from expressions to automata. This part of the chapter will not play any role later on the thesis, but we include it since KAT is one of the most basic extensions of Kleene algebras, with various applications.

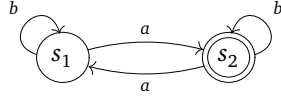
Most of the material in this chapter can be read without much knowledge of universal coalgebra.

### 3.1 Deterministic automata and regular expressions

Let  $A$  be a set of input letters (or symbols). A deterministic automaton with inputs in  $A$  is a pair  $(S, \langle o_S, t_S \rangle)$  consisting of a set of states  $S$  and a pair of functions  $\langle o_S, t_S \rangle$ , where  $o: S \rightarrow 2$  is the *output* function, which determines if a state  $s$  is final ( $o_S(s) = 1$ ) or not ( $o_S(s) = 0$ ), and  $t: S \rightarrow S^A$  is the *transition* function, which, given an input letter  $a$  determines the next state. We will frequently write  $s_a$  to denote  $t_S(s)(a)$  and refer to  $s_a$  as the derivative of  $s$  for input  $a$ . Moreover, when depicting deterministic automata

we will draw a single circle around non-final states and a double circle around final ones.

We illustrate the notation we will use in the representation of deterministic automata in the following example.



$$\begin{aligned} o_S(s_1) &= 0 & o_S(s_2) &= 1 \\ (s_1)_a &= s_2 & (s_1)_b &= s_1 \\ (s_2)_a &= s_1 & (s_2)_b &= s_2 \end{aligned}$$

Deterministic automata are coalgebras for the functor  $\mathcal{D}(X) = 2 \times X^A$ . The classical notion of automata homomorphism will instantiate precisely to the definition of coalgebra homomorphism for the functor  $\mathcal{D}$ : given two deterministic automata  $(S, \langle o_S, t_S \rangle)$  and  $(T, \langle o_T, t_T \rangle)$ , a function  $h: S \rightarrow T$  is a homomorphism if it preserves outputs and input derivatives, that is  $o_T(h(s)) = o_S(s)$  and  $h(s)_a = t_T(h(s))(a) = t_S(s)(a) = s_a$ , for any  $a \in A$ . These equations correspond to the commutativity of the following diagram.

$$\begin{array}{ccc} S & \xrightarrow{h} & T \\ \langle o_S, t_S \rangle \downarrow & & \downarrow \langle o_T, t_T \rangle \\ 2 \times S^A & \xrightarrow{id \times h^A} & 2 \times T^A \end{array}$$

The input derivative  $s_a$  of a state  $s$  for input  $a \in A$  can be extended to the word derivative  $s_w$  of a state  $s$  for input  $w \in A^*$  by defining, by induction on the length of  $w$ ,  $s_\epsilon = s$  and  $s_{aw'} = (s_a)_{w'}$ , where  $\epsilon$  denotes the empty word and  $aw'$  the word obtained by prefixing  $w'$  with the letter  $a$ . This enables an easy definition of the semantics of a state  $s$  of a deterministic automaton: the language  $L(s) \in 2^{A^*}$  recognized by  $s$ , that is the language containing all words  $w$  for which  $L(s)(w) = 1$ , is given by:

$$L(s)(w) = o_S(s_w) \tag{3.1}$$

For instance, the language recognized by state  $s_1$  of the automaton above is the set of all words with an odd number of  $a$ 's. It is easy to check that, for example,  $L(s_1)(bab) = o_S((s_1)_{bab}) = o_S(s_2) = 1$  and  $L(s_1)(aab) = o_S((s_1)_{aab}) = o_S(s_1) = 0$ .

Given two deterministic automata  $(S, \langle o_S, t_S \rangle)$  and  $(T, \langle o_T, t_T \rangle)$  a relation  $R \subseteq S \times T$  is a bisimulation if  $\langle s, t \rangle \in R$  implies

$$o_S(s) = o_T(t) \quad \text{and} \quad \langle s_a, t_a \rangle \in R \text{ for all } a \in A$$

We will write  $s \sim t$  whenever there exists a bisimulation  $R$  containing  $\langle s, t \rangle$ . This concrete definition of bisimulation can be recovered as a special case of the general definition of bisimulation for  $\mathcal{F}$ -coalgebras (Definition 2.2.4) by instantiating the functor  $\mathcal{F}$  to  $\mathcal{D}(X) = 2 \times X^A$ . The following theorem guarantees that the definition above is a valid proof principle for language equivalence of deterministic automata.

**3.1.1 THEOREM (Coinduction).** Given two deterministic automata  $(S, \langle o_S, t_S \rangle)$  and  $(T, \langle o_T, t_T \rangle)$ ,  $s \in S$  and  $t \in T$ :

$$s \sim t \Rightarrow L(s) = L(t)$$

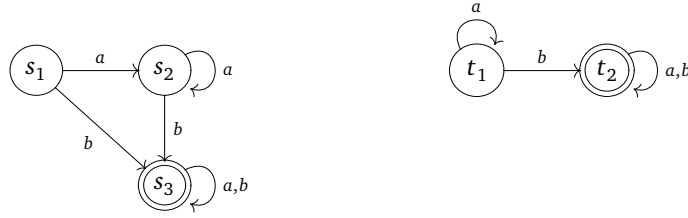
PROOF. Let  $R$  be a bisimulation relation containing  $\langle s, t \rangle$ . We prove, by induction on the length of words  $w \in A^*$ , that, for all  $\langle s, t \rangle \in R$ ,  $w \in L(s) \Leftrightarrow w \in L(t)$ .

$$\begin{aligned} L(s)(\epsilon) &= o_S(s) \stackrel{s \sim t}{=} o_T(t) = L(t) \\ L(s)(aw') &= o_S(s_{aw'}) = o_S((s_a)_{w'}) = o_T((t_a)_{w'}) = L(t) \end{aligned}$$

The one but last step follows using the induction hypothesis and the fact that  $\langle s_a, t_a \rangle \in R$ .  $\square$

To determine whether two states  $s$  and  $t$  of two deterministic automata  $(S, \langle o_S, t_S \rangle)$  and  $(T, \langle o_T, t_T \rangle)$  (over the same alphabet) recognize the same language we can now use coinduction: it is enough to construct a bisimulation containing  $\langle s, t \rangle$ .

**3.1.2 EXAMPLE.** Let  $(S, \langle o_S, t_S \rangle)$  and  $(T, \langle o_T, t_T \rangle)$  be the deterministic automata over the alphabet  $\{a, b\}$  given by



The relation  $R = \{\langle s_1, t_1 \rangle, \langle s_2, t_1 \rangle, \langle s_3, t_2 \rangle\}$  is a bisimulation:

$$\begin{aligned} o_S(s_1) &= o_S(s_2) = 0 = o_T(t_1) & o_S(s_2) &= 1 = o_T(t_2) \\ (s_1)_a &= s_2 \ R \ t_1 = (t_1)_a & (s_2)_a &= s_2 \ R \ t_1 = (t_1)_a & (s_3)_a &= s_3 \ R \ t_2 = (t_2)_a \\ (s_1)_b &= s_3 \ R \ t_2 = (t_1)_b & (s_2)_b &= s_2 \ R \ t_2 = (t_1)_b & (s_3)_b &= s_3 \ R \ t_2 = (t_2)_b \end{aligned}$$

Thus,  $L(s_1) = L(t_1) = L(s_2)$  and  $L(s_3) = L(t_2)$ .  $\spadesuit$

The language recognized by a state  $s$  is the behavior (or semantics) of  $s$ . Thus, the set of languages  $2^{A^*}$  over  $A$  can be thought of as the universe of all possible behaviors for deterministic automata. We now turn  $2^{A^*}$  into a deterministic automaton (with inputs in  $A$ ) and then show that such automaton has the universal property of being *final*, which will connect the coalgebraic semantics induced by the functor with the classical language semantics we have just presented.

For an input letter  $a \in A$ , the input derivative  $l_a$  of a language  $l \in 2^{A^*}$  on input  $a$  is defined by  $l_a(w) = l(aw)$ . The output of  $l$  is defined by  $l(\epsilon)$ . These notions determine a deterministic automaton  $(2^{A^*}, \langle o_L, t_L \rangle)$  defined, for  $l \in 2^{A^*}$  and  $a \in A$ , by  $o_L(l) = l(\epsilon)$  and  $t_L(l)(a) = l_a$ .

**3.1.3 THEOREM.** *The automaton  $(2^{A^*}, \langle o_L, t_L \rangle)$  is final. That is, for any deterministic automaton  $(S, \langle o_S, t_S \rangle)$ , there is a unique homomorphism  $L: S \rightarrow 2^{A^*}$  which makes the following diagram commute.*

$$\begin{array}{ccc} S & \xrightarrow{\quad L \quad} & 2^{A^*} \\ \langle o_S, t_S \rangle \downarrow & & \downarrow \langle o_L, t_L \rangle \\ 2 \times S^A & \xrightarrow{\quad id \times L^A \quad} & 2 \times (2^{A^*})^A \end{array}$$

Given a state  $s$ , the language  $L(s)$  is precisely the language recognized by  $s$  (as defined in equation (3.1)).

PROOF. We have to prove that the diagram commutes and that  $L$  is unique. First the commutativity:

$$\begin{aligned} o_L(L(s)) &= L(s)(\epsilon) = o_S(s) \\ t_L(L(s))(a) &= L(s)_a = \lambda w. L(s)(aw) = \lambda w. L(s_a)(w) = L(s_a) \end{aligned}$$

For the one but last step, note that, by definition of  $L$  (equation (3.1))

$$L(s)(aw) = o_S(s_{aw}) = o_S((s_a)_w) = L(s_a)(w)$$

For the uniqueness, suppose there is another morphism  $h: S \rightarrow 2^{A^*}$  such that, for every  $s \in S$  and  $a \in A$ ,  $o_L(h(s)) = o_S(s)$  and  $h(s)_a = h(s_a)$ .

We prove by induction on the length of words  $w \in A^*$  that  $h(s) = L(s)$ .

$$\begin{aligned} h(s)(\epsilon) &= o_L(h(s)) = o_S(s) = L(s)(\epsilon) \\ h(s)(aw) &= (h(s)_a)(w) = h(s_a)(w) \stackrel{(IH)}{=} L(s_a)(w) = L(s)(aw) \end{aligned}$$

□

The semantics induced by the unique map  $L$  into the final coalgebra coincides, in the case of deterministic automata, with the bisimulation semantics we defined above, that is  $s \sim t \Leftrightarrow L(s) = L(t)$  (the implication in Theorem 3.1.1 is actually an equivalence). This phenomenon is an instance of the general fact we presented in the last chapter that behavioral equivalence coincides with bisimilarity for many functors, including the deterministic automata functor  $\mathcal{D} = 2 \times \text{Id}^A$ . This fact will actually be true for most functors we consider in this thesis, apart from some of the ones considered in Chapter 6.

We will now recall the basic definitions and results on regular expressions. The set  $\mathcal{R}(A)$  of regular expressions over a finite input alphabet  $A$  is given by the following syntax:

$$r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r_1^*$$



The semantics of regular expressions is given in terms of languages<sup>1</sup> and it is defined by induction on the syntax as follows:

$$\begin{aligned} L(\underline{1}) &= \{\epsilon\} & L(\underline{0}) &= \emptyset & L(a) &= \{a\} \\ L(r_1 + r_2) &= L(r_1) \cup L(r_2) & L(r_1 r_2) &= L(r_1) \cdot L(r_2) & L(r_1^*) &= L(r_1)^* \end{aligned} \quad (3.2)$$

where, given languages  $l, l_1$  and  $l_2$ ,  $l_1 \cdot l_2 = \{w_1 w_2 \mid w_1 \in l_1 \text{ and } w_2 \in l_2\}$ ;  $l^* = \bigcup_{n \in \mathbb{N}} l^n$ , and, for  $n \in \mathbb{N}$ ,  $l^n$  is inductively defined by  $l^0 = \{\epsilon\}$  and  $l^{n+1} = l \cdot l^n$ .

Here, we have intentionally used  $L(r)$  to represent the language denoted by a regular expression  $r \in \mathcal{R}(A)$  (recall that we had used  $L(s)$  to represent the language recognized by a state  $s$  of a deterministic automaton). This is because we shall prove later that the languages recognized by deterministic automata are precisely the ones denoted by regular expressions.

We now equip the set  $\mathcal{R}(A)$  with a deterministic automaton structure. This definition was first proposed by Brzozowski in his paper *Derivatives of regular expressions* [29] and, for that reason, it is sometimes referred to as Brzozowski derivatives. We define the output  $o_{\mathcal{R}}(r)$  of a regular expression  $r$  by

$$\begin{aligned} o_{\mathcal{R}}(\underline{0}) &= 0 & o_{\mathcal{R}}(r_1 + r_2) &= o_{\mathcal{R}}(r_1) \vee o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(\underline{1}) &= 1 & o_{\mathcal{R}}(r_1 r_2) &= o_{\mathcal{R}}(r_1) \wedge o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(a) &= 0 & o_{\mathcal{R}}(r^*) &= 1 \end{aligned}$$

and the input derivative  $t_{\mathcal{R}}(r)(a) = r_a$  by

$$\begin{aligned} (\underline{0})_a &= \underline{0} & (r_1 + r_2)_a &= (r_1)_a + (r_2)_a \\ (\underline{1})_a &= \underline{0} & (r_1 r_2)_a &= \begin{cases} (r_1)_a r_2 & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ (r_1)_a r_2 + (r_2)_a & \text{otherwise} \end{cases} \\ (a)_{a'} &= \begin{cases} \underline{1} & \text{if } a = a' \\ \underline{0} & \text{if } a \neq a' \end{cases} & (r^*)_a &= r_a r^* \end{aligned}$$

In the definition of  $o_{\mathcal{R}}$  we use the fact that  $2 = \{0, 1\}$  can be given a lattice structure  $(\{0, 1\}, \vee, \wedge, 0, 1)$  (0 is neutral with respect to  $\vee$  and 1 with respect to  $\wedge$ ).

Similarly to what happened in deterministic automata, the input derivative  $r_a$  of a regular expression  $r$  for input  $a$  can be extended to the word derivative  $r_w$  of  $r$  for input  $w \in A^*$  by defining  $r_\epsilon = r$  and  $r_{aw} = (r_a)_w$ .

---

<sup>1</sup>Here, we represent languages as subsets of  $A^*$ , rather than functions  $2^{A^*}$ . Although we prefer the latter view on languages, the traditional semantics of regular expressions was presented as sets of words and we recall it here unchanged. We will only use the set view on languages when referring to the classical semantics of regular expressions.

We have now defined a deterministic automaton  $(\mathcal{R}(A), \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle)$  and thus, by Theorem 3.1.3, we have a unique map  $L$  which makes the following diagram commute.

$$\begin{array}{ccc} \mathcal{R}(A) & \xrightarrow{\quad L \quad} & 2^{A^*} \\ \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle \downarrow & & \downarrow \langle o_L, t_L \rangle \\ 2 \times (\mathcal{R}(A))^A & \xrightarrow{\quad id \times L^A \quad} & 2 \times (2^{A^*})^A \end{array} \quad (3.3)$$

We now prove that, for any  $r \in \mathcal{R}(A)$ , the semantics defined inductively in (3.2) is the same as the one given by the unique map into the final coalgebra  $L: \mathcal{R}(A) \rightarrow 2^{A^*}$ . More precisely, we prove, by induction on the structure of  $r$ , that

$$w \in L(r) \Leftrightarrow o_{\mathcal{R}}(r_w) = 1$$

where  $L(r)$  is the inductively defined semantics from equation (3.2).

$$L(\mathbf{0}) = \emptyset$$

$$L(\mathbf{1}) = \{\epsilon\} \text{ and } o_{\mathcal{R}}(\mathbf{1}_\epsilon) = 1$$

$$L(a) = \{a\} \text{ and } o_{\mathcal{R}}((a)_a) = 1$$

$$w \in L(r_1 + r_2) \Leftrightarrow w \in L(r_1) \text{ or } w \in L(r_2)$$

$$\stackrel{(IH)}{\Leftrightarrow} o_{\mathcal{R}}((r_1)_w) = 1 \text{ or } o_{\mathcal{R}}((r_2)_w) = 1 \Leftrightarrow o_{\mathcal{R}}((r_1 + r_2)_w) = 1$$

$$w \in L(r_1 r_2) \Leftrightarrow w = w_1 w_2, \text{ with } w_1 \in L(r_1) \text{ and } w_2 \in L(r_2)$$

$$\stackrel{(IH)}{\Leftrightarrow} o_{\mathcal{R}}((r_1)_w) = 1 \text{ and } o_{\mathcal{R}}((r_2)_w) = 1 \Leftrightarrow o_{\mathcal{R}}((r_1 r_2)_w) = 1$$

$$w \in L(r^*) \Leftrightarrow w \in L(r)^n, \text{ for some } n \in \mathbb{N} \Leftrightarrow w = w_1 \dots w_n \text{ with } w_i \in L(r)$$

$$\stackrel{(IH)}{\Leftrightarrow} w = w_1 \dots w_n \text{ with } o_{\mathcal{R}}(r_{w_i}) = 1 \Leftrightarrow o_{\mathcal{R}}((r^*)_w) = 1$$

We have now proved that the classical semantics of both deterministic automata and regular expressions coincides with the coalgebraic semantics. In the sequel, we will say that a regular expression  $r$  and a state  $s$  of a deterministic automaton are equivalent if  $L(s) = L(r)$ .

Next, we present Kleene's theorem, which states the equivalence between the class of languages recognized by finite deterministic automata (finite here means that the set of states  $S$  and the input alphabet  $A$  are finite) and the one denoted by regular expressions.

**3.1.4 THEOREM (Kleene's Theorem).** *Let  $l \in 2^{A^*}$ . The following are equivalent.*

1.  $l = L(r)$ , for some regular expression  $r \in \mathcal{R}(A)$ .
2.  $l = L(s)$ , for state  $s \in S$  of a finite deterministic automaton  $(S, \langle o_S, t_S \rangle)$ .

The proof of this theorem amounts to constructing an equivalent regular expression from a state of a deterministic automaton and, conversely, to constructing a deterministic automaton which has a state that is equivalent to a given regular expression. We will sketch these constructions in the next two sections.

### 3.1.1 From deterministic automata to regular expressions

To present the construction of a regular expression from a state of a deterministic automaton we need to introduce some algebraic laws to simplify regular expressions. For regular expressions  $r_1, r_2$ , if  $L(r_1) = L(r_2)$  we say that  $r_1$  and  $r_2$  are equivalent and we write  $r_1 \equiv r_2$ . The relation  $\equiv \subseteq \mathcal{R}(A) \times \mathcal{R}(A)$  is an equivalence relation. Moreover, if  $r_1 \equiv r_2$ , one can substitute  $r_1$  for  $r_2$ , or vice versa, in any regular expression and the result will be equivalent to the original expression.

Below are a few laws that can be used to simplify regular expressions.

$$\begin{array}{llll}
r_1 + (r_2 + r_3) & \equiv & (r_1 + r_2) + r_3 & \\
r_1 + r_2 & \equiv & r_2 + r_1 & \\
r + r & \equiv & r & \\
r + \underline{0} & \equiv & r & \\
r_1(r_2r_3) & \equiv & (r_1r_2)r_3 & \\
(r_2 + r_3)r_1 & \equiv & r_2r_1 + r_3r_1 & \\
r\underline{1} & \equiv & r & \equiv \underline{1}r \\
r\underline{0} & \equiv & \underline{0} & \equiv \underline{0}r \\
rr^* + \underline{1} & \equiv & r^* & \\
r_1 + r_2x \leq x & \Rightarrow & r_2^*r_1 \leq x & 
\end{array}$$

where  $\leq$  refers to subset inclusion:  $r_1 \leq r_2 \Leftrightarrow L(r_1) \subseteq L(r_2) \Leftrightarrow r_1 + r_2 \equiv r_2$ . The last law above is not an equation but it will allow the derivation of equations: it states that the inequality  $r_1 + r_2x \leq x$  has  $r_2^*r_1$  as least solution. Below we will make use of this fact to solve equations of the form  $x \equiv r_1 + r_2x$ . Note that using the laws above we can prove that  $r_2^*r_1$  is also the least solution of the latter equation ( $r_2^*r_1 \equiv (r_2r_2^* + \underline{1})r_1 \equiv r_2(r_2^*r_1) + r_1$ ). If  $\epsilon \notin L(r_2)$  then  $r_2^*r_1$  is actually the unique solution [101].

Let  $(S, \langle o_S, t_S \rangle)$  be a finite deterministic automaton. We associate with each  $s \in S$  an equation

$$r_s \equiv \sum_{a \in A} ar_{s_a} + \underline{o_S}(s) \quad (3.4)$$

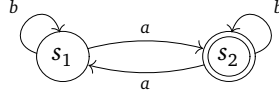
Note that the sum here is well defined because  $A$  is finite. In this way, we build a system with  $n$  equations and  $n$  variables, where  $n$  is the size of  $S$ . We can solve it by using the laws above. The fact that the resulting  $r_s$  is equivalent to  $s$  is an easy proof by coinduction. The relation

$$R = \{ \langle r, s \rangle \mid r \equiv r_s, s \in S \}$$

is a bisimulation: for any  $r \equiv r_s$ , we have that  $o_{\mathcal{R}}(r) = o_{\mathcal{R}}(r_s) = o_S(s)$ ,  $r_a \equiv (r_s)_a \equiv r_{s_a}$  and  $\langle r_{s_a}, s_a \rangle \in R$ .

We will first illustrate the construction with an example and then we will present a formal definition of the solution of a system of equations of the same form as (3.4) using matrices.

**3.1.5 EXAMPLE.** Consider the following deterministic automaton over the alphabet  $A = \{a, b\}$ :



Let  $r_1$  and  $r_2$  denote  $r_{s_1}$  and  $r_{s_2}$ , respectively. The system of the equations arising from the automaton is the following:

$$r_1 \equiv (ar_2 + br_1) + \underline{0} \quad r_2 \equiv (ar_1 + br_2) + \underline{1}$$

Using the laws above we can rewrite the second equation to  $r_2 \equiv (ar_1 + \underline{1}) + br_2$  and then solve it which yields  $r_2 \equiv b^*(ar_1 + \underline{1})$  or, simplified,  $r_2 \equiv b^*ar_1 + b^*$ . We then replace  $r_2$  in the first equation and simplify it:

$$r_1 \equiv (a(b^*ar_1 + b^*) + br_1) + \underline{0} \equiv (ab^*a + b)r_1 + ab^*$$

Solving it and then replacing it in the equation for  $r_2$  results in the following two expressions

$$r_1 \equiv (ab^*a + b)^*ab^* \quad r_2 \equiv b^*a(ab^*a + b)^*ab^* + b^*$$

which satisfy  $L(r_1) = L(s_1)$  and  $L(r_2) = L(s_2)$ . ♠

We will conclude this section by recalling from [67] a formalization of a system of  $n$  equations of the form

$$x_i \equiv a_{i1}x_1 + \dots + a_{in}x_n + o_i \quad 1 \leq i \leq n$$

Constructing an  $n \times n$  matrix  $T$  with all the  $a_{ij}$ , an  $n \times 1$  vector  $O$  with all the  $o_i$  and an  $n \times 1$  vector  $X$  with all the  $x_i$ , we obtain the matrix-vector equation

$$X = TX + O \tag{3.5}$$

Moreover, we define, for an  $n \times n$  matrix  $R$ , with entries  $r \in \mathcal{R}(A)$ , a matrix  $R^*$ , by induction on  $n$ .

If  $n = 1$ , then the matrix  $R$  is reduced to a single entry  $r$  and  $R^* = r^*$ .

If  $n \geq 1$ , we divide the matrix into four submatrices

$$R = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

with  $A$  and  $D$  are square with dimensions, respectively,  $m \times m$  (for some  $m < n$ ) and  $(n - m) \times (n - m)$ . By induction hypothesis, we can use  $A^*$ ,  $B^*$ ,  $C^*$  and  $D^*$ . This allows us to define

$$R^* = \begin{pmatrix} (A + BD^*C)^* & (A + BD^*C)^*BD^* \\ (D + CA^*B)^*CA^* & (D + CA^*B)^* \end{pmatrix}$$

Now, one can show that the vector  $T^*O$  is the least (with respect to the pointwise order  $\leq$  of regular expressions) solution to the system in (3.5) [67, Theorem A.3]. Let us revisit the example above using this method. First, we construct the matrices  $X$ ,  $T$  and  $O$ :

$$X = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \quad T = \begin{pmatrix} b & a \\ a & b \end{pmatrix} \quad O = \begin{pmatrix} \underline{0} \\ \underline{1} \end{pmatrix}$$

Now we compute  $T^*$

$$T^* = \begin{pmatrix} (b + ab^*a)^* & (b + ab^*a)^*ab^* \\ (b + ab^*a)^*ab^* & (b + ab^*a)^* \end{pmatrix}$$

and  $T^*O$

$$T^*O = \begin{pmatrix} ((b + ab^*a)^*\underline{0} + ((b + ab^*a)^*ab^*)\underline{1}) \\ ((b + ab^*a)^*ab^*\underline{0} + ((b + ab^*a)^*)\underline{1}) \end{pmatrix} \equiv \begin{pmatrix} (b + ab^*a)^*ab^* \\ (b + ab^*a)^* \end{pmatrix}$$

In the last step we used some of the equations to simplify regular expressions. Now note that the expression computed for  $r_1$  is (almost) the same as the one we obtained in the example above, whereas the one for  $r_2$  is quite different (syntactically), but still equivalent. We will show the proof of this equivalence later in Example 3.1.17.

### 3.1.2 From regular expressions to deterministic automata

Given a regular expression  $r$  we want to construct a finite deterministic automaton with an equivalent (that is, bisimilar) state to  $r$ . We have shown that the set  $\mathcal{R}(A)$  of regular expressions over  $A$  carries a deterministic automata structure given by  $\langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle$ . Hence, an obvious way of constructing an automaton with a state equivalent to a regular expression  $r$  is to consider the subcoalgebra  $\langle r \rangle$  generated by  $r$  in  $\langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle$  (this idea goes back to Brzozowski [29], the state  $r$  of the constructed automaton will be equivalent to the regular expression  $r$ ). However, even for very simple expressions  $\langle r \rangle$  might not be finite. Take, for example, the expression  $(a^*)^*$ . Computing the input derivative for  $a$  yields:

$$\begin{aligned} ((a^*)^*)_a &= (\underline{1}a^*)(a^*)^* \\ ((\underline{1}a^*)(a^*)^*)_a &= (\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^* \\ ((\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*)_a &= ((\underline{0}a^* + \underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*) + ((\underline{1}a^*)(a^*)^*)_a \\ &\vdots \end{aligned}$$

Now note that all the derivatives are above equivalent. However, they are syntactically different and therefore they will not be identified as denoting the same state, which results in  $\langle (a^*)^* \rangle$  having an infinite state space. In order to achieve finiteness, it is enough to remove double occurrences of expressions  $r$  in sums of the form  $\dots + r + \dots + r + \dots$ . This uses the laws for associativity, commutativity and idempotency of  $+$  (ACI), but note that it does not amount to take the quotient of  $\mathcal{R}(A)$  with respect to the equivalence induced by the laws ACI, which would also guarantee finiteness [29, 39] but would identify more expressions. For instance, the expressions  $a + b$  and  $b + a$  for  $a, b \in A$  with  $a \neq b$  will not be identified in our procedure.

The following proposition provides a syntactic characterization of the derivative  $r_w$ , for a word  $w \in A^+$  and  $r \in \mathcal{R}(A)$ .

**3.1.6 PROPOSITION.** *Let  $w \in A^*$  and  $r, r_1, r_2 \in \mathcal{R}(A)$ . Then, the word derivatives of  $(r_1 + r_2)$ ,  $r_1 r_2$  and  $r^*$  are of the form:*

$$\begin{aligned} (r_1 + r_2)_w &= (r_1)_w + (r_2)_w \\ (r_1 r_2)_w &= (r_1)_{w_1} r_2 + \cdots + (r_1)_{w_k} r_2 + (r_2)_{w'_1} + \cdots + (r_2)_{w'_l} \\ (r^*)_w &= r_{w_1} r^* + \cdots + r_{w_m} r^* \end{aligned}$$

for  $k, l, m \geq 0$ ,  $w_i, w'_i \in A^*$ .

PROOF. By induction on the length of  $w \in A^*$ . For  $w = \epsilon$ , the equalities hold trivially. For  $w = aw'$ , we have:

$$\begin{aligned} (r_1 + r_2)_{aw'} &= ((r_1)_a + (r_2)_a)_{w'} \stackrel{(IH)}{=} (r_1)_{aw'} + (r_2)_{aw'} \\ (r_1 r_2)_{aw'} &= \begin{cases} ((r_1)_a r_2)_{w'} & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ ((r_1)_a r_2)_{w'} + (r_2)_{aw'} & \text{otherwise} \end{cases} \\ &\stackrel{(IH)}{=} \begin{cases} (r_1)_{aw_1} r_2 + \cdots + (r_1)_{aw_k} r_2 + (r_2)_{w'_1} + \cdots + (r_2)_{w'_l} & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ (r_1)_{aw_1} r_2 + \cdots + (r_1)_{aw_k} r_2 + (r_2)_{w'_1} + \cdots + (r_2)_{w'_l} + (r_2)_{aw'} & \text{otherwise} \end{cases} \\ (r^*)_{aw'} &= (r_a r^*)_{w'} = r_{aw_1} r^* + \cdots + r_{aw_k} r^* + (r^*)_{w'_1} + \cdots + (r^*)_{w'_l} \\ &\stackrel{(IH)}{=} r_{u_1} r^* + \cdots + r_{u_m} r^* \end{aligned}$$

□

Using Proposition 3.1.6, we can now prove that the number of word derivatives of a regular expression is finite, if double occurrences of expressions  $r_1$  in sums of the form  $\dots + r_1 + \dots + r_1 + \dots$  are removed.

**3.1.7 THEOREM.** *Let  $r \in \mathcal{R}(A)$ . The number of word derivatives  $r_w$  is finite, as long as double occurrences of expressions  $r_1$  in sums of the form  $\dots + r_1 + \dots + r_1 + \dots$  are removed.*

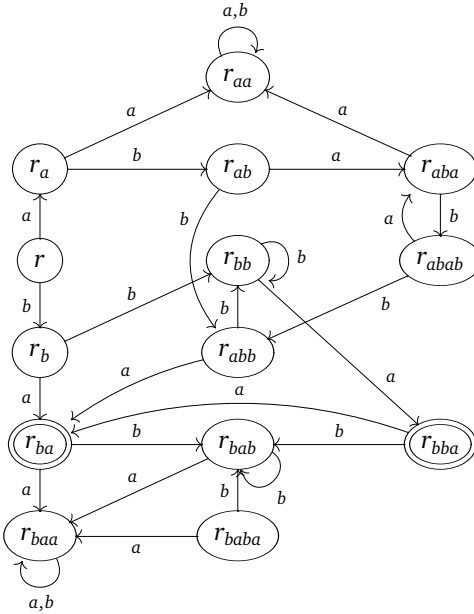
PROOF. By induction on the structure of the expression. The base cases are trivial: they have either one, two or three distinct word derivatives. The sum  $r_1 + r_2$  has as upper-bound for the number of derivatives, once we remove the double occurrences in the sum,  $N \times M$ ; the concatenation  $r_1 r_2$  has as bound  $2^{N+M}$ ; and the star  $(r_1)^*$  has  $2^N$ , where  $N$  and  $M$  are the inductive bounds for  $r_1$  and  $r_2$ . □

We now illustrate the construction of an automaton using the procedure described above.

**3.1.8 EXAMPLE.** Let  $A = \{a, b\}$  and  $r = (ab + b)^*ba$ . We compute derivatives incrementally, marking with  $\checkmark$  the derivatives that are not new and numbering the new ones:

$$\begin{aligned}
r_\epsilon &= (ab + b)^*ba && \textcircled{1} \\
r_a &= (\underline{1}b + \underline{0})(ab + b)^*ba + \underline{0}a && \textcircled{2} \\
r_b &= (\underline{0}b + \underline{1})(ab + b)^*ba + \underline{1}a && \textcircled{3} \\
r_{aa} &= (\underline{0}b + \underline{0})(ab + b)^*ba + \underline{0}a && \textcircled{4} \\
r_{ab} &= (\underline{0}b + \underline{1} + \underline{0})(ab + b)^*ba + \underline{0}a && \textcircled{5} \\
r_{ba} &= (\underline{0}b + \underline{0})(ab + b)^*ba + (\underline{1}b + \underline{0})(ab + b)^*ba + \underline{0}a + \underline{0}a + \underline{1} = r_{aa} + r_a + \underline{1} && \textcircled{6} \\
r_{bb} &= (\underline{0}b + \underline{0})(ab + b)^*ba + (\underline{0}b + \underline{1})(ab + b)^*ba + \underline{1}a + \underline{0}a + \underline{0} = r_{aa} + r_b + \underline{0} && \textcircled{7} \\
r_{aaa} &= (\underline{0}b + \underline{0})(ab + b)^*ba + \underline{0}a = r_{aa} && \checkmark \\
r_{aab} &= (\underline{0}b + \underline{0})(ab + b)^*ba + \underline{0}a = r_{aa} && \checkmark \\
r_{aba} &= (\underline{0}b + \underline{0})(ab + b)^*ba + (\underline{1}b + \underline{0})(ab + b)^*ba + \underline{0}a + \underline{0}a = r_{aa} + r_a && \textcircled{8} \\
r_{abb} &= (\underline{0}b + \underline{0})(ab + b)^*ba + (\underline{0}b + \underline{1})(ab + b)^*ba + \underline{1}a + \underline{0}a = r_{aa} + r_b && \textcircled{9} \\
r_{baa} &= r_{aa} + \underline{0} && \textcircled{10} \\
r_{bab} &= r_{aa} + r_{ab} + \underline{0} && \textcircled{11} \\
r_{bba} &= r_{aa} + r_{ba} + \underline{0} = r_{aa} + r_a + \underline{1} + \underline{0} && \textcircled{12} \\
r_{bbb} &= r_{aa} + r_{bb} + \underline{0} = r_{bb} && \checkmark \\
r_{abaa} &= r_{aa} && \checkmark \\
r_{abab} &= r_{aa} + r_{ab} && \textcircled{13} \\
r_{abba} &= r_{aa} + r_{ba} = r_{ba} && \checkmark \\
r_{abbb} &= r_{aa} + r_{bb} = r_{bb} && \checkmark \\
r_{baaa} &= r_{baa} && \checkmark \\
r_{baab} &= r_{baa} && \checkmark \\
r_{baba} &= r_{aa} + r_{aba} + \underline{0} = r_{aa} + r_a + \underline{0} && \textcircled{14} \\
r_{babbb} &= r_{aa} + r_{abb} + \underline{0} = r_{aa} + r_b + \underline{0} = r_{bb} && \checkmark \\
r_{bbaa} &= r_{aa} + \underline{0} = r_{baa} && \checkmark \\
r_{bbab} &= r_{aa} + r_{ab} + \underline{0} = r_{bab} && \checkmark \\
r_{ababa} &= r_{aa} + r_{aba} = r_{aba} && \checkmark \\
r_{ababb} &= r_{aa} + r_{abb} = r_{abb} && \checkmark \\
r_{babaa} &= r_{aa} + \underline{0} = r_{baa} && \checkmark \\
r_{babab} &= r_{aa} + r_{ab} + \underline{0} = r_{bab} && \checkmark
\end{aligned}$$

The resulting automaton will have 14 states (note also that only  $r_{ba}$  and  $r_{bba}$  have output value 1):



In the above calculations, we strictly followed the procedure described above. The goal was to construct a finite deterministic automaton with a state equivalent to the regular expression  $r$  and we did not worry about its size. It is however obvious that by allowing the simplification of certain expressions the resulting automaton would be much smaller. For instance, let us add to the procedure the following three rules:  $\underline{0}r$  can be replaced by  $\underline{0}$ ;  $\underline{1}r$  can be replaced by  $r$  and  $\underline{0}$  can be eliminated from sums. The calculations for  $r$  would then be much more simplified (and easier to read):

$$r_\epsilon = (ab + b)^*ba \quad \textcircled{1}$$

$$r_a = b(ab + b)^*ba \quad \textcircled{2}$$

$$r_b = (ab + b)^*ba + a \quad \textcircled{3}$$

$$r_{aa} = \underline{0} \quad \textcircled{4}$$

$$r_{ab} = (ab + b)^*ba = r \quad \checkmark$$

$$r_{ba} = b(ab + b)^*ba + \underline{1} \quad \textcircled{5}$$

$$r_{bb} = (ab + b)^*ba + a = r_b \quad \checkmark$$

$$r_{aaa} = \underline{0} = r_{aa} \quad \checkmark$$

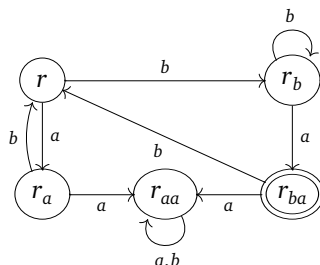
$$r_{aab} = r_{aa} \quad \checkmark$$

$$r_{baa} = \underline{0} = r_{aa} \quad \checkmark$$

$$r_{bab} = (ab + b)^*ba = r \quad \checkmark$$



The resulting automaton would then have 5 states:



In fact, this is the minimal automaton recognizing the language denoted by  $r$ . Note that the axioms we considered to simplify  $r$  are not always enough to get the minimal automaton (for that one would have to consider the complete set of axioms we will present later in this chapter). However, in many cases the automata computed is minimal: an empirical study about this phenomenon appears in [88].

### 3.1.3 Non-deterministic automata and the subset construction

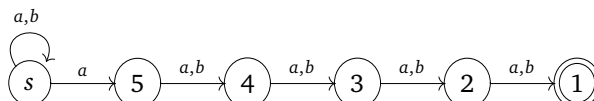
A non-deterministic automaton (NDA) is similar to a deterministic automaton but the transition function gives a set of next-states for each input letter instead of a single state. NDA's often provide compacter representations of regular languages than deterministic automata. For that, they are computationally very interesting and much research has been devoted to constructions compiling a regular expression into an NDA [9, 19, 34, 49, 80, 111] (we will show an example of such construction below). Surprisingly, in what concerns language acceptance NDA's are not more powerful than deterministic automata. For every NDA there exists a deterministic automaton with a state equivalent to a given state of the NDA. Such deterministic automaton can be obtained from a given NDA by the so-called subset (or powerset) construction [93], which we will show below coalgebraically.

Formally, an NDA over the input alphabet  $A$  is a pair  $(S, \langle o, \delta \rangle)$ , where  $S$  is a set of states and  $\langle o, \delta \rangle : S \rightarrow 2 \times (\mathcal{P}_\omega(S))^A$  is a pair of functions with  $o$  as before and where  $\delta$  determines for each input letter  $a$  a set of possible next states.

As an example of the compactness of NDA's, consider the following regular language (taken from [67]):

$$\{w \in \{a, b\}^* \mid \text{the fifth symbol from the right is } a\}$$

One can intuitively construct a NDA with a state  $s$  recognizes this language (which could be, for instance, denoted by the regular expression  $(a + b)^* a(a + b)(a + b)(a + b)(a + b)$ ):



A deterministic automaton recognizing the same language will have at least  $2^5 = 32$  states.

In order to formally compute the language recognized by a state  $x$  of an NDA  $\mathcal{A}$ , it is usual to first determinize it, constructing a deterministic automaton  $\mathbf{det}(\mathcal{A})$  where the state space is  $\mathcal{P}_\omega(S)$ , and then compute the language recognized by the state  $\{x\}$  of  $\mathbf{det}(\mathcal{A})$ . Next, we describe in coalgebraic terms how to construct the automaton  $\mathbf{det}(\mathcal{A})$  [96].

Given an NDA  $\mathcal{A} = (S, \langle o, \delta \rangle)$ , we construct  $\mathbf{det}(\mathcal{A}) = (\mathcal{P}_\omega(S), \langle \bar{o}, \bar{\delta} \rangle)$ , where, for all  $Y \in \mathcal{P}_\omega(S)$ ,  $a \in A$ , the functions  $\bar{o}: \mathcal{P}_\omega(S) \rightarrow 2$  and  $\bar{\delta}: \mathcal{P}_\omega(S) \rightarrow \mathcal{P}_\omega(S)^A$  are

$$\bar{o}(Y) = \begin{cases} 1 & \text{if } \exists_{y \in Y} o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \quad \bar{\delta}(Y)(a) = \bigcup_{y \in Y} \delta(y)(a).$$

The automaton  $\mathbf{det}(\mathcal{A})$  is such that the language  $L(\{x\})$  recognized by  $\{x\}$  is the same as the one recognized by  $x$  in the original NDA  $\mathcal{A}$  (more generally, the language recognized by state  $X \in \mathcal{P}_\omega(S)$  of  $\mathbf{det}(\mathcal{A})$  is the union of the languages recognized by each state  $x \in X$  of  $\mathcal{A}$ ).

We summarize the situation above with the following commuting diagram:

$$\begin{array}{ccccc} S & \xrightarrow{\{\cdot\}} & \mathcal{P}_\omega(S) & \xrightarrow{L} & 2^{A^*} \\ \langle o, \delta \rangle \downarrow & & \swarrow \langle \bar{o}, \bar{\delta} \rangle & & \downarrow \langle o_L, t_L \rangle \\ 2 \times \mathcal{P}_\omega(S)^A & \dashrightarrow & & \dashrightarrow & 2 \times (2^{A^*})^A \end{array}$$

We note that the language semantics of NDA's, presented in the above diagram, can alternatively be achieved by using  $\lambda$ -coinduction [16, 61].

### From regular expressions to non-deterministic automata: the Berry-Sethi construction

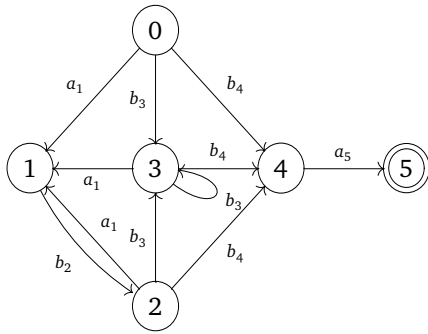
There are several algorithms to construct a non-deterministic automaton from a regular expression. We will show here the one presented in [19] by Berry and Sethi. We shall generalize this algorithm in the next section in order to deal with the expressions of Kleene algebra with tests. The basic idea behind the algorithm is that of marking: all input letters in a regular expression are marked (with subscripts) in order to make them distinct. As an example, a marked version of  $(ab + b)^*ba$  is  $(a_1b_2 + b_3)^*b_4a_5$ , where  $a_1$  and  $a_5$  are considered different letters. The choice we made for the subscripts are the positions of the letters in the expression. For that reason the Berry-Sethi construction is often referred to as position automata.

We will explain the algorithm with an example (taken from [19]) and then state the results that justify its correctness.

**3.1.9 EXAMPLE.** Let  $r = (ab + b)^*ba$  and let  $\bar{r} = (a_1b_2 + b_3)^*b_4a_5$  be its marked version. We define  $c_i = (\bar{r})_{a_1 \dots a_i}$ , and call it the continuation  $i$  of  $\bar{r}$ . We then construct an automaton from  $\bar{r}$  in the following way:

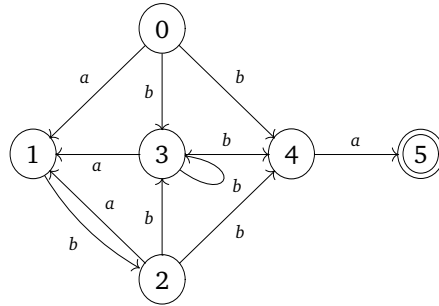
1. The automaton will have a state  $i \in \{1, 2, 3, 4, 5\}$  for each distinct symbol in  $\bar{r}$  plus an extra state 0 which will be equivalent to  $\bar{r}$ .
2. A state  $i$  has a transition to state  $j$ , labeled by  $a_j$ , if  $(c_i)_{a_j} = c_j$ . A state  $i$  is final if  $o_{\mathcal{R}}(c_i) = 1$ .

The automaton resulting from  $\bar{r} = (a_1 b_2 + b_3)^* b_4 a_5$  is the following



$$\begin{aligned}
 c_0 &= (\bar{r}) = (a_1 b_2 + b_3)^* b_4 a_5 \\
 c_1 &= (\bar{r})_{a_1} = b_2 (a_1 b_2 + b_3)^* b_4 a_5 \\
 c_2 &= (\bar{r})_{a_1 b_2} = (c_1)_{b_2} = (a_1 b_2 + b_3)^* b_4 a_5 \\
 c_3 &= (\bar{r})_{a_1 b_2 b_3} = (c_2)_{b_3} = (a_1 b_2 + b_3)^* b_4 a_5 \\
 c_4 &= (\bar{r})_{a_1 b_2 b_3 b_4} = (c_3)_{b_4} = a_5 \\
 c_5 &= \bar{1}
 \end{aligned}$$

Note that to compute the transition structure we had to compute all input derivatives for each  $c_i$ . This can be overcome by using some of the properties of derivatives of expressions with distinct symbols (more below). Now, note that by deleting all the marks in the labels of the automaton above the state 0 of the resulting NDA accepts precisely the language denoted by  $(ab + b)^* ba$  (all words that finish with  $ba$  and all other occurrences of  $a$  are followed by one or more  $b$ 's).



(3.6)



The algorithm above works as expected due to the properties of derivatives of expressions with distinct letters. We summarize the crucial properties for the correctness of the algorithm.

**3.1.10 THEOREM** ([19, Proposition 3.2 and Theorem 3.4]). *Let  $\bar{r}$  be the regular expression obtained from  $r$  by marking all symbols to make them distinct. Then, the following holds:*

1. If  $\mathcal{A}'$  is an automaton with a state  $s$  such that  $L(s) = L(\bar{r})$ , then the state  $s$  of the automaton  $\mathcal{A}$ , obtained from  $\mathcal{A}'$  by unmarking all the labels, is such that  $L(s) = L(r)$ .
2. Given any symbol  $a$  and word  $w$ , the derivative  $(\bar{r})_{aw}$  is either  $\underline{0}$  or unique modulo associativity, commutativity and idempotency.

Starting from a regular expression  $r \in \mathcal{R}(A)$ , we can then obtain a non-deterministic automaton by first marking the symbols, then apply the algorithm above and finally unmarking the labels. If wanted, a deterministic automaton can then be obtained via the subset construction (the complexity of this construction for position automata was studied in [33]).

In [19], the authors presented also a more efficient way of computing the position automaton, based on the fact that each continuation is uniquely determined by an input symbol. We briefly recall it here, since this is precisely the version we will later generalize for KAT expressions. Let  $pos(r)$  denote the positions (distinct symbols) in the regular expression  $r$ . For any regular expression  $r$  and  $i \in pos(r)$  we define:

$$\begin{aligned} first(r) &= \{i \mid p_i w \in L(\bar{r})\} \\ follow(r, i) &= \{j \mid up_i p_j v \in L(\bar{r})\} \\ last(r) &= \{i \mid wp_i \in L(\bar{r})\} \end{aligned}$$

This sets can be computed efficiently from the expression: we recall [19, Proposition 4.3].

**3.1.11 PROPOSITION** ([19, Proposition 4.3]). *Let  $r$  be a regular expression with distinct symbols.  $\mathbf{F}$ , defined by the rules below, is such that  $\mathbf{F}(r, \{!\})$  yields a set of pairs of the form  $\langle a_i, follow(r!, a_i) \rangle$ , where  $!$  is a symbol distinct from all symbols in  $r$ . The rules are:*

$$\begin{aligned} \mathbf{F}(r_1 + r_2, S) &= \mathbf{F}(r_1, S) \cup \mathbf{F}(r_2, S) \\ \mathbf{F}(r_1 r_2, S) &= \mathbf{F}(r_1, first(r_2)) \cup o_{\mathcal{R}}(r_2).S \cup \mathbf{F}(r_2, S) \\ \mathbf{F}(r_1^*, S) &= \mathbf{F}(r_1, first(r_1)) \cup S \\ \mathbf{F}(a, S) &= \{\langle a, S \rangle\} \\ \mathbf{F}(\underline{1}, S) &= \mathbf{F}(\underline{0}, S) = \emptyset \end{aligned}$$

Here, for a set  $S$ ,  $\underline{1}.S = S$  and  $\underline{0}.S = \emptyset$ . Note that in  $\mathbf{F}$  also the set  $last(r)$  is computed:  $i \in last(r) \Leftrightarrow ! \in follow(r!, i)$ .

The position automaton corresponding to a given regular expression  $r \in \mathcal{R}(A)$  is then given by

$$\mathcal{A}_{pos}(r) = (\{0\} \cup pos(\bar{r}), A, \langle o, \delta \rangle)$$

where  $\bar{r}$  is the marked version of  $r$  and  $o$  and  $\delta$  are defined as follows:

$$\begin{aligned} o_S(0) &= o_{\mathcal{R}}(r) & \delta(0)(a) &= \{j \mid j \in first(\bar{r}), unmark(a_j) = a\} \\ o_S(i) &= \begin{cases} 1 & \text{if } i \in last(\bar{r}) \\ 0 & \text{otherwise} \end{cases} & \delta(i)(a) &= \{j \mid j \in follow(\bar{r}, i), unmark(a_j) = a\} \quad i \neq 0 \end{aligned}$$

We show an example of the algorithm above. We consider again  $r = (ab + b)^*ba$  and its marked version  $\bar{r} = (a_1b_2 + b_3)^*b_4a_5$ .

$$\begin{aligned}
\text{first}(\bar{r}) &= \{1, 3, 4\} & \text{first}(a_1b_2 + b_3) &= \{1, 3\} & \text{first}(a_1b_2) &= \{1\} & \text{first}(b_4a_5) &= \{4\} \\
\mathbf{F}(\bar{r}, \{!\}) &= \mathbf{F}((a_1b_2 + b_3)^*, \{4\}) \cup \mathbf{F}(b_4a_5, \{!\}) \\
&= \mathbf{F}(a_1b_2 + b_3, \{1, 3, 4\}) \cup \mathbf{F}(b_4, \{5\}) \cup \mathbf{F}(a_5, \{!\}) \\
&= \mathbf{F}(a_1b_2, \{1, 3, 4\}) \cup \mathbf{F}(b_3, \{1, 3, 4\}) \cup \{\langle b_4, \{5\} \rangle, \langle a_5, \{!\} \rangle\} \\
&= \mathbf{F}(a_1, \{2\}) \cup \mathbf{F}(b_2, \{1, 3, 4\}) \cup \{\langle b_3, \{1, 3, 4\} \rangle, \langle b_4, \{5\} \rangle, \langle a_5, \{!\} \rangle\} \\
&= \{\langle a_1, \{2\} \rangle, \langle b_2, \{1, 3, 4\} \rangle, \langle b_3, \{1, 3, 4\} \rangle, \langle b_4, \{5\} \rangle, \langle a_5, \{!\} \rangle\}
\end{aligned}$$

The position automaton  $\mathcal{A}_{\text{pos}}(r)$  constructed is the same as the one presented above in (3.6).

It should be remarked that the construction of the position automaton from a regular expression does not always extend to additional operators, such as intersection or complement.

### 3.1.4 Kleene algebras

In Section 3.1.1, we showed a set of algebraic laws that allow for a sound simplification of regular expressions with respect to language equivalence. In this section, we will show a set of laws (which includes the aforementioned ones) that is sound and complete: all true equations between regular expressions can be proved purely algebraically using only the equations included in this set.

**3.1.12 DEFINITION** (Kleene algebra). A Kleene algebra  $\mathcal{K} = (\Sigma, +, \cdot, (-)^*, 0, 1)$  consists of a nonempty set  $\Sigma$  with two distinguished elements 0 and 1, two binary operations  $+$  and  $\cdot$  (usually omitted when writing the expressions) and a unary operation  $(-)^*$  satisfying the following axioms, for  $e, e_1, e_2, e_3 \in \Sigma$ :

$$\begin{array}{lll}
e_1 + (e_2 + e_3) & \equiv & (e_1 + e_2) + e_3 & \text{(associativity of } +) \\
e_1 + e_2 & \equiv & e_2 + e_1 & \text{(commutativity of } +) \\
e + e & \equiv & e & \text{(idempotency of } +) \\
e + \underline{0} & \equiv & e & \text{(0 is an identity of } +) \\
\\
e_1(e_2e_3) & \equiv & (e_1e_2)e_3 & \text{(associativity of } \cdot) \\
e\underline{1} & \equiv & e & \equiv \underline{1}e & \text{(1 is an identity of } \cdot) \\
e\underline{0} & \equiv & \underline{0} & \equiv \underline{0}e & \text{(0 is an annihilator of } \cdot) \\
\\
(e_2 + e_3)e_1 & \equiv & e_2e_1 + e_3e_1 & \text{(right distributivity)} \\
e_1(e_2 + e_3) & \equiv & e_1e_2 + e_1e_3 & \text{(left distributivity)} \\
e^*e + \underline{1} & \equiv & e^* & \\
ee^* + \underline{1} & \equiv & e^* & \\
e_1 + e_2x \leq x & \Rightarrow & e_2^*e_1 \leq x & \\
e_1 + xe_2 \leq x & \Rightarrow & e_1e_2^* \leq x &
\end{array}$$

where  $\leq$  is defined by  $e \leq f \Leftrightarrow e + f \equiv f$ .



The first group of rules, which summarizes the properties of  $+$ , essentially says that  $\mathcal{K}$  is a join-semilattice. Together with the second group, which contains the properties of  $\cdot$  and its interaction with  $+$ , it states that  $\mathcal{K}$  is an idempotent semiring. The last group of rules axiomatizes the star operator. It follows quite easily from the axioms that  $\leq$  is a partial order. Moreover, all the operators are monotone with respect to  $\leq$ : if  $e_1 \leq e_2$  then  $e_1 e \leq e_2 e$ ,  $e e_1 \leq e e_2$ ,  $e_1 + e \leq e_2 + e$  and  $e_1^* \leq e_2^*$ .

Typical examples of Kleene algebras are [67]:

- The set  $2^{A^*}$  of languages over an input alphabet  $A$  with constants  $\emptyset$  and  $\{\epsilon\}$  and operations  $\cup$ ,  $\cdot$  and  $*$ . In  $2^{A^*}$ ,  $\leq$  is just set inclusion.
- The family of all binary relations over a set  $X$  with the empty relation for  $0$ , the identity relation for  $1$ , union for  $+$ , relational composition for  $\cdot$  and the reflexive transitive closure operator for  $(-)^*$ .
- The family of  $n \times n$  matrices over a Kleene algebra  $\mathcal{K} = (\Sigma, +, \cdot, (-)^*, 0, 1)$ . We show the operations and identity elements for the case  $n = 2$ . The identity elements for  $+$  and  $\cdot$  are

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

respectively, and the operations  $+$ ,  $\cdot$  and  $(-)^*$  are given by

$$\begin{pmatrix} e_1 & e_2 \\ e_3 & e_4 \end{pmatrix} + \begin{pmatrix} e_5 & e_6 \\ e_7 & e_8 \end{pmatrix} = \begin{pmatrix} e_1 + e_5 & e_2 + e_6 \\ e_3 + e_7 & e_4 + e_8 \end{pmatrix},$$

$$\begin{pmatrix} e_1 & e_2 \\ e_3 & e_4 \end{pmatrix} \cdot \begin{pmatrix} e_5 & e_6 \\ e_7 & e_8 \end{pmatrix} = \begin{pmatrix} e_1 e_5 + e_2 e_7 & e_1 e_6 + e_2 e_8 \\ e_3 e_5 + e_4 e_7 & e_3 e_6 + e_4 e_8 \end{pmatrix},$$

and

$$\begin{pmatrix} e_1 & e_2 \\ e_3 & e_4 \end{pmatrix}^* = \begin{pmatrix} (e_1 + e_2 e_4^* e_3)^* & (e_1 + e_2 e_4^* e_3)^* e_2 e_4^* \\ (e_4 + e_3 e_1^* e_2)^* e_3 e_1^* & (e_4 + e_3 e_1^* e_2)^* \end{pmatrix},$$

respectively. The sum and product operations are just standard matrix sum and multiplication.

Two regular expressions  $r_1$  and  $r_2$  are equivalent (that is, they denote the same language) if and only if  $r_1 \equiv r_2$  is derivable from the axioms of Kleene algebra. We will present below a coalgebraic proof of this result (which states the soundness and completeness of the Kleene algebra axioms for regular expressions). But before that let us show a few typical equalities derivable using the axioms above.

**3.1.13 EXAMPLE.** The following equalities hold in Kleene algebras:

$$\begin{array}{ll} e^* & \equiv e^* + \underline{1} \\ e^* e^* & \equiv e^* \\ e^{**} & \equiv e^* \\ (e + f)^* & \equiv (e^* f)^* e^* \quad \text{denesting rule} \\ e(fe)^* & \equiv (ef)^* e \quad \text{shifting rule} \end{array}$$

The first equality follows easily:

$$e^* \equiv \underline{1} + ee^* \equiv \underline{1} + \underline{1} + ee^* \equiv \underline{1} + e^*$$

For the second equality, we have:

$$e^*e^* \equiv (1 + ee^*)e^* \equiv e^* + ee^*e^* \geq e^*$$

and, using the fact that  $e^*e^*$  is the least solution of  $ex + e^* \leq x$

$$ee^* + e^* \leq \underline{1} + ee^* + e^* \equiv e^* + e^* \equiv e^* \Rightarrow e^*e^* \leq e^*$$

For the third equality we use first that  $e^{**}$  is the least solution of  $e^*x + \underline{1} \leq x$ :

$$e^* \equiv e^* + \underline{1} \equiv e^*e^{**} + \underline{1} \Rightarrow e^{**} \leq e^*$$

and then that  $e^*$  is the least solution of  $ex + \underline{1} \leq x$ :

$$e^{**} \equiv e^*e^{**} + \underline{1} \equiv (1 + ee^*)e^{**} + \underline{1} \equiv e^{**} + ee^*e^{**} + \underline{1} \leq ee^{**} + \underline{1} \Rightarrow e^* \leq e^{**}$$

For the denesting rule, first observe that

$$\begin{aligned} (e^*f)^*e^* &\leq ((e+f)^*(e+f))^*(e+f)^* && \text{(by monotonicity)} \\ &\leq ((e+f)^*)^*(e+f)^* && ((e+f)^*(e+f) \leq (e+f)^*) \\ &\equiv (e+f)^*(e+f)^* \\ &\equiv (e+f)^* \end{aligned}$$

Then we use the fact that  $(e+f)^*$  is the least solution of  $x(e+f) + \underline{1} \leq x$ :

$$\begin{aligned} &(e^*f)^*e^*(e+f) + \underline{1} \\ \equiv &(e^*f)^*e^*e + (e^*f)^*e^*f + \underline{1} \\ \leq &(e^*f)^*e^* + (e^*f)^*e^* + \underline{1} && (e^*e \leq e^* \text{ end } (e^*f)^*e^*f \leq (e^*f)^*) \\ \leq &(e^*f)^*e^* && (\underline{1} \leq (e^*f)^*e^*) \\ &\Downarrow \\ &(e+f)^* \leq (e^*f)^*e^* \end{aligned}$$

For the shifting rule we use the facts that  $e(fe)^*$  and  $(ef)^*e$  are the least solutions of, respectively,  $x(fe) + e \leq x$  and  $(ef)x + e \leq x$ :

$$\begin{aligned} (ef)^*efe + e &\equiv ((ef)^*ef + \underline{1})e \equiv (ef)^*e \Rightarrow e(fe)^* \leq (ef)^*e \\ efe(fe)^* + e &\equiv e(fe(fe)^* + \underline{1}) \equiv e(fe)^* \Rightarrow (ef)^*e \leq e(fe)^*(ef)^*e \end{aligned}$$



We next present an alternative proof of a result originally due to Brzozowski [29, Theorem 6.4].

**3.1.14 THEOREM** (Fundamental theorem for regular expressions). *Every regular expression  $r \in \mathcal{R}(A)$  satisfies the following equality*

$$r \equiv \underline{o_{\mathcal{R}}(r)} + \sum_{a \in A} ar_a$$

This theorem is closely related to the proof of Kleene's theorem, namely the construction of a regular expression from a deterministic automaton (cf. equation (3.4)). In the subsequent chapters we will generalize (and make extensive use of) this equality. Note that the intended reading of the equality above is syntactic: the left side of the equation can be derived from the right side using the axioms of Kleene algebra. If the goal would be to prove equivalence, that is  $L(r) = L(\underline{o_{\mathcal{R}}(r)} + \sum_{a \in A} ar_a)$ , then the proof would be straightforward using coinduction.

The name *Fundamental Theorem* is borrowed from [97], where an analogue theorem is stated for formal power series (functions  $f : A^* \rightarrow k$ , for a semiring  $k$ ). As it is explained there, the name is chosen in analogy to analysis. Viewing prefixing with the letter  $a$  as a kind of integration, the theorem tells us that regular expression derivation and integration are inverse operations: the equality above gives a way of obtaining  $r$  from the  $a$ -derivatives  $r_a$  (and the initial value  $\underline{o_{\mathcal{R}}(r)}$ ).

PROOF (Theorem 3.1.14). By induction on the structure of  $r$ .

The base cases are direct consequences of the fact that  $\underline{1}$  is the identity for  $\cdot$  and  $\underline{0}$  is an annihilator for  $\cdot$  and the identity for  $+$ .

$$\begin{aligned} \underline{1} &\equiv \underline{1} + \underline{0} \equiv \underline{1} + \sum_{a \in A} a\underline{0} \\ \underline{0} &\equiv \underline{0} + \sum_{a \in A} a\underline{0} \\ a &\equiv \underline{0} + a\underline{1} + \sum_{a' \in A \setminus \{a\}} a'a \end{aligned}$$

The cases  $r_1 + r_2$  and  $r_1 r_2$  follow by induction and using several of the semiring laws (we remark that in the second case the induction hypotheses are applied subsequently and not at the same time):

$$\begin{aligned} r_1 + r_2 &\stackrel{IH}{\equiv} \left( \underline{o_{\mathcal{R}}(r_1)} + \sum_{a \in A} a(r_1)_a \right) + \left( \underline{o_{\mathcal{R}}(r_2)} + \sum_{a \in A} a(r_2)_a \right) \\ &\equiv \underline{o_{\mathcal{R}}(r_1)} + \underline{o_{\mathcal{R}}(r_2)} + \sum_{a \in A} a((r_1)_a + (r_2)_a) \\ &\equiv \underline{o_{\mathcal{R}}(r_1 + r_2)} + \sum_{a \in A} a((r_1 + r_2)_a) \\ \\ r_1 r_2 &\stackrel{IH}{\equiv} \left( \underline{o_{\mathcal{R}}(r_1)} + \sum_{a \in A} a(r_1)_a \right) r_2 \\ &\equiv \underline{o_{\mathcal{R}}(r_1)} r_2 + \sum_{a \in A} a(r_1)_a r_2 \end{aligned}$$



$$\begin{aligned}
&\stackrel{IH}{\equiv} \underline{o_{\mathcal{R}}(r_1)} \left( \underline{o_{\mathcal{R}}(r_2)} + \sum_{a \in A} a(r_2)_a \right) + \sum_{a \in A} a(r_1)_a r_2 \\
&\equiv \underline{o_{\mathcal{R}}(r_1 r_2)} + \sum_{a \in A} a((r_1)_a r_2 + \underline{o_{\mathcal{R}}(r_1)}(r_2)_a) \\
&\equiv \underline{o_{\mathcal{R}}(r_1 r_2)} + \sum_{a \in A} a(r_1 r_2)_a
\end{aligned}$$

For  $r^*$  we distinguish between the cases  $o_{\mathcal{R}}(r) = 0$  and  $o_{\mathcal{R}}(r) = 1$ . For  $o_{\mathcal{R}}(r) = 0$ , it follows easily by induction:

$$r^* \equiv \underline{1} + r r^* \stackrel{IH}{\equiv} \underline{1} + \left( \underline{o_{\mathcal{R}}(r)} + \sum_{a \in A} a r_a \right) r^* \equiv \underline{1} + \sum_{a \in A} a(r_a r^*)$$

For  $o_{\mathcal{R}}(r) = 1$ , a bit more of work is needed:

$$r^* \equiv \underline{1} + r r^* \stackrel{IH}{\equiv} \underline{1} + \left( \underline{o_{\mathcal{R}}(r)} + \sum_{a \in A} a r_a \right) r^* \equiv r^* + \left( \underline{1} + \sum_{a \in A} a(r_a r^*) \right)$$

From this equation, we can conclude that  $r^* \geq \underline{1} + \sum_{a \in A} a(r_a r^*)$ . It remains to prove that  $r^* \leq \underline{1} + \sum_{a \in A} a(r_a r^*)$ . We do that by showing that  $x \equiv \underline{1} + x r$ , where  $x = \underline{1} + \sum_{a \in A} a(r_a r^*)$ . Using the second star inequality rule this implies that  $\underline{1} r \leq x$  which is precisely the inequality we want to prove.

$$\begin{aligned}
x = \underline{1} + \sum_{a \in A} a(r_a r^*) &\equiv \underline{1} + \sum_{a \in A} a r_a (1 + r^* r) \\
&\equiv \underline{1} + \underline{1} + \sum_{a \in A} a r_a + \sum_{a \in A} a r_a r^* r \\
&\stackrel{IH}{\equiv} \underline{1} + r + \sum_{a \in A} a r_a r^* r \\
&\equiv \underline{1} + (\underline{1} + \sum_{a \in A} a r_a r^*) r = \underline{1} + x r
\end{aligned}$$

□

### Soundness and completeness

It is the goal of this section to prove that the axiomatization is sound and complete with respect to language equivalence (recall that  $L(r_1) = L(r_2) \Leftrightarrow r_1 \sim r_2$ ), that is

$$r_1 \sim r_2 \Leftrightarrow r_1 \equiv r_2$$

The original proof of soundness and completeness of Kleene algebras is due to Kozen [66], who later presented an alternative proof [68]. A coalgebraic proof of soundness and

completeness was presented by Jacobs [61]<sup>2</sup>. The proof we present here is a slight variation on [61, 68].

For soundness, the right to left implication, it is enough to prove that  $\equiv$  is a bisimulation relation.

**3.1.15 THEOREM (Soundness).** *The equivalence relation  $\equiv$  is a bisimulation, that is, for every  $r_1, r_2 \in \mathcal{R}(A)$ , if  $r_1 \equiv r_2$  then*

$$o_{\mathcal{R}}(r_1) = o_{\mathcal{R}}(r_2) \text{ and } (r_1)_a \equiv (r_2)_a$$

*In other words, the Kleene algebra axiomatization of regular expressions is sound: for all  $r_1, r_2 \in \mathcal{R}(A)$ , if  $r_1 \equiv r_2$  then  $r_1 \sim r_2$ .*

PROOF. By induction on the length of derivations for  $\equiv$ .

For the cases of length zero everything follows from unwinding the definitions of  $o_{\mathcal{R}}$  and  $t_{\mathcal{R}}$ . We show the proof for the equations  $r_1 + r_2 \equiv r_2 + r_1$  and  $rr^* + \underline{1} \equiv r^*$ .

$$\begin{aligned} o_{\mathcal{R}}(r_1 + r_2) &= o_{\mathcal{R}}(r_1) \vee o_{\mathcal{R}}(r_2) = o_{\mathcal{R}}(r_2 + r_1) \\ o_{\mathcal{R}}(rr^* + \underline{1}) &= 1 = o_{\mathcal{R}}(r^*) \\ (r_1 + r_2)_a &= (r_1)_a + (r_2)_a \equiv (r_2)_a + (r_1)_a = (r_2 + r_1)_a \\ (rr^* + \underline{1})_a &\equiv r_a r^* + \underline{o_{\mathcal{R}}(r)}(r^*)_a + \underline{0} = \begin{cases} (r^*)_a + \underline{0}(r^*)_a + \underline{0} & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ (r^*)_a + \underline{1}(r^*)_a + \underline{0} & \text{otherwise} \end{cases} \\ &\equiv (r^*)_a \end{aligned}$$

For the inductive cases, we illustrate the case  $r_1 + r_2 x \leq x \Rightarrow r_2^* r_1 \leq x$ . The other case is proved precisely in the same way. Suppose we have just derived  $r_2^* r_1 \leq x$ , using  $r_1 + r_2 x \leq x$  as a premise. We want to prove that  $o_{\mathcal{R}}(r_2^* r_1) \leq o_{\mathcal{R}}(x)$  and  $(r_2^* r_1)_a \leq x_a$ . The first, which simplifies to  $o_{\mathcal{R}}(r_1) \leq o_{\mathcal{R}}(x)$ , follows from the induction hypothesis  $o_{\mathcal{R}}(r_1) + o_{\mathcal{R}}(r_2 x) \leq o_{\mathcal{R}}(x)$ . For the second inequality, we calculate:

$$\begin{aligned} (r_2^* r_1)_a &= (r_2)_a r_2^* r_1 + (r_1)_a \\ &\equiv (r_2)_a x + (r_1)_a && (r_2^* r_1 \leq x) \\ &\leq (r_2)_a x + \underline{o_{\mathcal{R}}(r_2)} x_a + (r_1)_a \\ &\equiv (r_2 x)_a + (r_1)_a \\ &\equiv (r_1 + r_2 x)_a \\ &\leq x_a && \text{(induction hypothesis)} \end{aligned}$$

□

Let  $\mathcal{R}(A)/_{\equiv}$  denote the set of expressions modulo  $\equiv$ . The equivalence relation  $\equiv$  induces the equivalence map  $[-]: \mathcal{R}(A) \rightarrow \mathcal{R}(A)/_{\equiv}$  given by  $[r] = \{r' \mid r \equiv r'\}$ .

<sup>2</sup>The paper by Jacobs was of great inspiration for us to formulate the proof of soundness and completeness of the generalized language of regular expressions we will present in the subsequent chapters.

Theorem 3.1.15 guarantees the existence of a unique map (by Theorem 2.2.7)

$$\langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle: \mathcal{R}(A)/\equiv \rightarrow 2 \times (\mathcal{R}(A)/\equiv)^A$$

which makes  $[-]$  a coalgebra homomorphism:

$$\begin{array}{ccc} \mathcal{R}(A) & \xrightarrow{[-]} & \mathcal{R}(A)/\equiv & \overline{o_{\mathcal{R}}}([r]) = o_{\mathcal{R}}(r) \text{ and } [r]_a = [r_a] \\ \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle \downarrow & & \downarrow \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle & \\ 2 \times (\mathcal{R}(A))^A & \longrightarrow & 2 \times (\mathcal{R}(A)/\equiv)^A & \end{array}$$

In order to prove completeness (that is,  $r_1 \sim r_2 \Rightarrow r_1 \equiv r_2$ ), we recall the main steps of the coalgebraic proof of completeness of Kleene algebras, given by Jacobs [61], which can be seen as a coalgebraic review of Kozen's proof [68].

① First, the unique map into the final coalgebra is factorized into an epimorphism followed by a monomorphism.

$$\begin{array}{ccccc} & & L & & \\ & & \curvearrowright & & \\ \mathcal{R}(A)/\equiv & \xrightarrow{e} & I & \xrightarrow{m} & 2^{A^*} \\ \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle \downarrow & & \downarrow \langle o_I, t_I \rangle & & \downarrow \\ 2 \times (\mathcal{R}(A))^A & \longrightarrow & 2 \times I^A & \longrightarrow & 2 \times (2^{A^*})^A \end{array}$$

② The map  $e$  is proved to be an isomorphism. The key idea behind the proof is that both coalgebras  $(\mathcal{R}(A)/\equiv, \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle)$  and  $(I, \langle o_I, t_I \rangle)$  are final among the locally finite coalgebras: coalgebras for which the smallest subcoalgebra generated by a point is always finite.

To prove that  $(\mathcal{R}(A)/\equiv, \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle)$  is final, the following facts are needed:

- (i) For any locally finite automaton  $(S, \langle o_S, t_S \rangle)$ , there exists a coalgebra homomorphism  $[-]_S: (S, \langle o_S, t_S \rangle) \rightarrow (\mathcal{R}(A)/\equiv, \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle)$ .
- (ii) For any homomorphism  $f: (S, \langle o_S, t_S \rangle) \rightarrow (T, \langle o_T, t_T \rangle)$ , it holds that  $[-]_T \circ f = [-]_S$ .
- (iii) The homomorphism  $[-]_{\mathcal{R}(A)/\equiv}$  is the identity.

Points (i)-(iii) above imply that for any finite automaton  $(S, \langle o_S, t_S \rangle)$ , there exists a *unique* coalgebra homomorphism  $[-]_S: (S, \langle o_S, t_S \rangle) \rightarrow (\mathcal{R}(A)/\equiv, \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle)$ . Hence, the coalgebra  $(\mathcal{R}(A)/\equiv, \langle \overline{o_{\mathcal{R}}}, \overline{t_{\mathcal{R}}} \rangle)$  is final among the locally finite ones.

For the finality of  $(I, \langle o_I, t_I \rangle)$ , we need to observe that:

- (i) For any locally finite automaton  $(S, \langle o_S, t_S \rangle)$ , there exists a coalgebra homomorphism  $e \circ [-]_S: (S, \langle o_S, t_S \rangle) \rightarrow (I, \langle o_I, t_I \rangle)$ .
- (ii) If there exist two homomorphisms  $f, g: (S, \langle o_S, t_S \rangle) \rightarrow (I, \langle o_I, t_I \rangle)$ , then  $f = g$ : by finality, we have that  $m \circ f = m \circ g$  and, since  $m$  is a monomorphism,  $f = g$ .

③ From ① and ②, it follows that  $L: \mathcal{R}(A)/\equiv \rightarrow 2^A$  is injective, which is the key to prove completeness.

**3.1.16 THEOREM (Completeness).** *For all  $r_1, r_2 \in \mathcal{R}(A)$ , if  $r_1 \sim r_2$  then  $r_1 \equiv r_2$ .*

PROOF. For all  $r_1, r_2 \in \mathcal{R}(A)$ , if  $r_1 \sim r_2$  then we have that they are mapped into the same element in the final coalgebra:  $L(r_1) = L(r_2)$ . Since  $[-]$  is a homomorphism, this implies that  $L([r_1]) = L([r_2])$  (here we are using  $L$  to denote the map into the final coalgebra both from  $\mathcal{R}(A)$  and  $\mathcal{R}(A)/\equiv$ ). By ③,  $L$  is injective and thus we can conclude that  $[r_1] = [r_2]$ , that is  $r_1 \equiv r_2$ .  $\square$

Completeness now enables the use of coinduction to prove that expressions are provably equivalent using the axioms of Kleene algebra.

**3.1.17 EXAMPLE.** Let us illustrate the proof of the denesting rule  $(a + b)^* \equiv (a^*b)^*a^*$  – which we proved using the axioms in Example 3.1.13.

We construct the relation

$$R = \{((a + b)^*, (a^*b)^*a^*), ((a + b)^*, (a^*b)(a^*b)^*a^* + a^*)\}$$

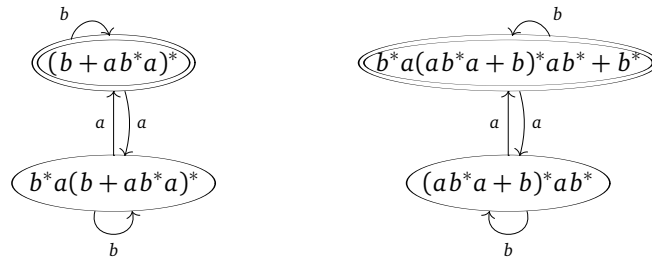
and observe that it is a bisimulation (in the calculations below we use some simplifications, which is no problem, given the fact that  $\equiv$  is a bisimulation; this means that actually the relation  $R$  above is formally a bisimulation up to a finite set of (sound) axioms):

$$\begin{aligned} ((a + b)^*)_a &= (a + b)^* & ((a + b)^*)_b &= (a + b)^* \\ ((a^*b)^*a^*)_a &= (a^*b)(a^*b)^*a^* + a^* & ((a^*b)^*a^*)_b &= (a^*b)^*a^* \\ ((a^*b)(a^*b)^*a^* + a^*)_a &= (a^*b)(a^*b)^*a^* + a^* & ((a^*b)(a^*b)^*a^* + a^*)_b &= (a^*b)^*a^* \end{aligned}$$

Thus,  $(a + b)^* \sim (a^*b)^*a^*$  which implies, by completeness,  $(a + b)^* \equiv (a^*b)^*a^*$ . For another example, take the expressions  $(b + ab^*a)^*$  and  $b^*a(ab^*a + b)^*ab^* + b^*$ . The relation

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

is a bisimulation. It is very easy to check: we show the automaton structure underlying each expression:



The algebraic proof requires a bit more of ingenuity, using the denesting and shifting rule which we proved in Example 3.1.13:

$$\begin{aligned}
(b + ab^*a)^* &\equiv (b^*ab^*a)^*b^* && \text{(denesting rule)} \\
&\equiv b^*(ab^*a^*b^*)^* && \text{(shifting rule)} \\
&\equiv b^*(ab^*a^*b^*(ab^*a^*b^*)^* + \underline{1}) && (r^* \equiv \underline{1} + rr^*) \\
&\equiv b^*ab^*a^*b^*(ab^*a^*b^*)^* + b^* && \text{(left distributivity and } r\underline{1} = r) \\
&\equiv b^*a(b^*a^*b^*a)^*b^*a^*b^* + b^* && \text{(shifting rule)} \\
&\equiv b^*a(b + ab^*a)^*a^*b^* + b^* && \text{(denesting rule)} \\
&\equiv b^*a(ab^*a + b)^*a^*b^* + b^* && \text{(commutativity of } +)
\end{aligned}$$



## 3.2 Automata on guarded strings and KAT expressions

Kleene algebra with tests (KAT) is an equational system that combines Kleene and Boolean algebra. One can model basic programming constructs and assertions in KAT, which allowed for its application in compiler optimization, program transformation or dataflow analysis [8, 70, 72]. In this section, we will recall the basic definitions on KAT and we will show how to generalize the Berry-Sethi construction (Section 3.1.3) in order to (efficiently) obtain an automaton from a KAT expression.

**3.2.1 DEFINITION** (Kleene algebra with tests). A Kleene algebra with tests is a two-sorted structure  $(\Sigma, B, +, \cdot, (-)^*, \bar{\phantom{x}}, 0, 1)$  where

- $(\Sigma, +, \cdot, (-)^*, 0, 1)$  is a Kleene algebra,
- $(B, +, \cdot, \bar{\phantom{x}}, 0, 1)$  is a Boolean algebra, and
- $(B, +, \cdot, \bar{\phantom{x}}, 0, 1)$  is a subalgebra of  $(\Sigma, +, \cdot, (-)^*, 0, 1)$ .



Given a set  $P$  of (primitive) action symbols and a set  $B$  of (primitive) test symbols, we can define the free Kleene algebra with tests on generators  $P \cup B$  as follows. Syntactically, the set  $BExp$  of Boolean tests is given by:

$$BExp \ni b ::= b \in B \mid b_1 b_2 \mid b_1 + b_2 \mid \bar{b} \mid 0 \mid 1$$

The set  $At$  of atoms is given by  $At = 2^B$  (an atom  $\alpha \in At$  is a minimal nonzero element of the free Boolean algebra  $B$  on  $B$ ). The set of KAT expressions is given by

$$Exp \ni e, f ::= p \in P \mid b \in BExp \mid ef \mid e + f \mid e^*$$

The free Kleene algebra with tests on generators  $P \cup B$  is obtained by quotienting  $BExp$  by the axioms of Boolean algebra and  $Exp$  by the axioms of Kleene algebra.

Guarded strings were introduced in [63] as an abstract interpretation for program schemes. They are like ordinary strings over an input alphabet  $P$ , but the symbols in

$P$  alternate with the atoms of the free Boolean algebra generated by  $B$ . We define the set  $GS$  of guarded strings by

$$GS = (At \times P)^* At$$

Kozen [69] showed that the regular sets of guarded strings plays the same role in KAT as regular languages play in Kleene algebra. He showed an analogue of Kleene's theorem: *automata on guarded strings*, which are non-deterministic automata over the alphabet  $P \cup B$ , recognize precisely the regular sets of guarded strings.

**3.2.2 DEFINITION** (Regular sets of guarded strings). Each KAT expression  $e$  denotes a set  $G(e)$  of guarded strings define inductively on the structure of  $e$  as follows:

$$\begin{aligned} G(p) &= \{\alpha p \beta \mid \alpha, \beta \in At\} \\ G(b) &= \{\alpha \mid \alpha \leq b\} \\ G(e + f) &= G(e) \cup G(f) \\ G(e f) &= G(e) \diamond G(f) \\ G(e^*) &= \bigcup_{n \geq 0} G(e)^n \end{aligned}$$

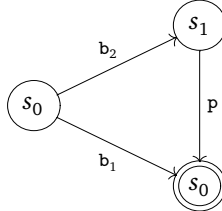
where, given two guarded strings  $x = \alpha_0 p_0 \dots p_{n-1} \alpha_n$  and  $y = \beta_0 q_1 \dots q_{n-1} \beta_n$ , we define the fusion product of  $x$  and  $y$  by  $x \diamond y = \alpha_0 p_0 \dots \alpha_n q_1 \dots q_{n-1} \beta_n$ , if  $\alpha_n = \beta_0$ , otherwise  $x \diamond y$  is undefined. Then, given  $X, Y \subseteq GS$ ,  $X \diamond Y$  is the set containing all existing fusion products  $x \diamond y$  of  $x \in X$  and  $y \in Y$  and  $X^n$  is defined inductively as  $X^0 = X$  and  $X^{n+1} = X \diamond X^n$ .

A set of guarded strings is regular if it is equal to  $G(e)$  for some KAT expression  $e$ . Note that a guarded string is itself a KAT expression and  $G(x) = \{x\}$ . ♣

**3.2.3 EXAMPLE.** Consider the KAT expression  $e = b_1 + b_2 p$  over  $B = \{b_1, b_2\}$  and  $P = \{p\}$ . We compute the set  $G(e)$ :

$$\begin{aligned} G(e) &= G(b_1) \cup (G(b_2) \diamond G(p)) \\ &= \{\alpha \mid \alpha \leq b_1\} \cup (\{\alpha \mid \alpha \leq b_2\} \diamond \{\alpha p \beta \mid \alpha, \beta \in At\}) \\ &= \{\alpha \mid \alpha \leq b_1\} \cup \{\alpha p \beta \mid \alpha \leq b_2, \beta \in At\} \end{aligned}$$

We will now show an example of an automaton on guarded strings. As mentioned above such automaton is just a non-deterministic automaton over the alphabet  $A = P \cup B$ , that is  $(S, \langle o_S, t_S \rangle)$  with  $o: S \rightarrow 2$  and  $t: S \rightarrow \mathcal{P}_\omega(S)^A$ . State  $s_0$  of the following automaton would *recognize* (we shall explain the precise meaning of this below) the same set of guarded string as  $e$ :



Let us now explain how to compute  $G(s)$ , the set of guarded strings accepted by a state  $s$  of an automaton  $\mathcal{A}$  on guarded strings. A guarded string  $x$  is accepted by  $\mathcal{A}$  if  $x \in GS(e)$  for some  $e \in L(s)$ , where  $L(s) \subseteq (P \cup B)^*$  is just the language accepted by  $s$ , as defined in 3.1.3. In the example above, we have  $L(s) = \{b_1, b_2p\}$  and thus  $G(s) = G(b_1) \cup G(b_2p) = G(b_1 + b_2p)$ . ♠

Later in [71], Kozen showed that the deterministic version of automata on guarded strings (already defined in [69]) fits neatly in the coalgebraic framework: two expressions are bisimilar if and only if they recognize the same set of guarded strings.

A deterministic automaton on guarded strings is a pair  $(S, \langle o_S, t_S \rangle)$  where  $o: S \rightarrow \mathcal{B}$  (recall that  $\mathcal{B}$  is the free Boolean algebra on  $B$ , satisfying  $\mathcal{B} \cong 2^{At}$ ) and  $t: S \rightarrow S^{At \times P}$ . We can obtain a deterministic automaton by using the following generalization of Brzozowski derivatives for KAT expressions (modulo ACI, as before).

**3.2.4 DEFINITION** (Brzozowski derivatives for KAT expressions). Given a KAT expression  $e \in \text{Exp}$ , we define  $E: \text{Exp} \rightarrow \mathcal{B} \cong 2^{At}$  and  $D: \text{Exp} \rightarrow \text{Exp}^{At \times P}$  by induction on the structure of  $e$ . First,  $E(e)$  is given by:

$$E(p) = \emptyset \quad E(b) = \{\alpha \in At \mid \alpha \leq b\} \quad E(e f) = E(e) \cap E(f) \quad E(e + f) = E(e) \cup E(f) \quad E(e^*) = At$$

Next, we define  $e_{\alpha q} = D(e)(\langle \alpha, q \rangle)$  by

$$p_{\alpha q} = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases} \quad b_{\alpha q} = 0 \quad (e f)_{\alpha q} = \begin{cases} e_{\alpha q} f + f_{\alpha q} & \text{if } \alpha \in E(e) \\ e_{\alpha q} f & \text{if } \alpha \notin E(e) \end{cases}$$

$$(e + f)_{\alpha q} = e_{\alpha q} + f_{\alpha q} \quad (e^*)_{\alpha q} = e_{\alpha q} e^*$$

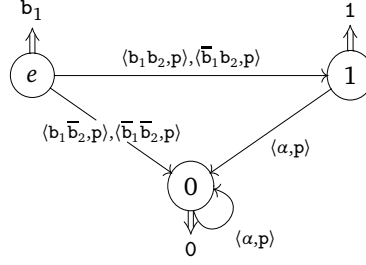
♣

The functions  $\langle E, D \rangle$  provide  $\text{Exp}$  with a deterministic automata structure, which leads, by finality, to the existence of a unique homomorphism

$$\begin{array}{ccc} \text{Exp} & \xrightarrow{\quad G \quad} & (2^{At})^{(At \times P)^*} \cong 2^{GS} \\ \langle E, D \rangle \downarrow & & \downarrow \\ 2^{At} \times \text{Exp}^{At \times P} & \xrightarrow{\quad} & 2^{At} \times (2^{GS})^{At \times P} \end{array}$$

which assigns to each expression the language of guarded strings that it denotes.

**3.2.5 EXAMPLE.** The deterministic automaton of  $e = b_1 + b_2p$ , which is the deterministic counterpart of the automaton in Example 3.2.3, would be



since, for  $B = \{b_1, b_2\}$ ,  $At = \{b_1 b_2, b_1 \bar{b}_2, \bar{b}_1 b_2, \bar{b}_1 \bar{b}_2\}$  and

$$\begin{aligned} e_{b_1 b_2, p} &= 0 + (b_2 p)_{b_1 b_2, p} = p_{b_1 b_2, p} = 1 & e_{\bar{b}_1 b_2, p} &= 0 + (b_2 p)_{\bar{b}_1 b_2, p} = p_{\bar{b}_1 b_2, p} = 1 \\ e_{b_1 \bar{b}_2, p} &= 0 + (b_2 p)_{b_1 \bar{b}_2, p} = 0 & e_{\bar{b}_1 \bar{b}_2, p} &= 0 \end{aligned}$$

$$E(e) = \{\alpha \mid \alpha \leq b_1\} = \{b_1 b_2, b_1 \bar{b}_2\} \quad E(0) = \emptyset \quad E(1) = At$$

Above we represent the output  $o_s(s)$  of a state by  $\Rightarrow b$  where  $b \in \mathcal{B}$  is the element corresponding to the set  $o_s(s)$  coming from the isomorphism  $2^{At} \cong \mathcal{B}$ . ♠

In short, there are two types of automata recognizing regular sets of guarded strings:

$$S \rightarrow 2 \times (\mathcal{P}_\omega(S))^{P \cup B} \quad S \rightarrow \mathcal{B} \times S^{At \times P}$$

The non-deterministic version has the advantage that it is very close to the expression, that is, one can easily compute the automaton from a given KAT expression and back, but its semantics is not coalgebraic. The deterministic version fits neatly into the coalgebraic framework, but it has the disadvantage that constructing the automaton from an expression inherits the same problems as in the Brzozowski construction: the number of equivalences that need to be decided increase exponentially. We propose here yet another type of automaton to recognize guarded strings: the construction from an expression to an automaton will be inspired by the Berry-Sethi construction presented in Section 3.1.3 and it is linear in the size of the expression.

Note that since KAT expressions can be interpreted as regular expressions over the extended alphabet  $B \cup P$ , the Berry-Sethi construction could be applied directly.

**3.2.6 THEOREM.** Let  $e$  be a KAT expression and  $A_{pos}(e)$  be the corresponding position automaton. Then,  $G(e) = G(A_{pos}(e))$ .

PROOF. We know that  $L(A_{pos}(e)) = L(e)$ . Now the result follows by using Kozen's observation in [69] that given a guarded string  $e$  and an automaton  $\mathcal{A}$  such that  $L(\mathcal{A}) = L(e)$ , one has  $G(e) = G(\mathcal{A})$ .  $\square$



The resulting automaton would have precisely the same type as the non-deterministic version of automata on guarded strings. However, there would be one state for each input symbol in  $P \cup B$ . The construction we will show next includes only states for each atomic action in  $P$ , yielding smaller automata. From a given KAT expression  $e$ , we will construct an automaton  $(S, t)$  where  $t : S \rightarrow \mathcal{B} \times \mathcal{P}_\omega(S)^{\mathcal{B} \times P}$ . This automaton type can be regarded as a compromise between the non-deterministic and deterministic versions of Kozen's automata.

We will start by generalizing the sets *first*, *follow* and *last*.

$$\begin{aligned} \text{first}(e) &= \{\langle b, p \rangle \mid b_1 b_2 \dots b_n p x \in L(e), b = \bigvee (b_1 \wedge b_2 \wedge \dots \wedge b_n)\} \\ \text{follow}(e, p) &= \{\langle b, q \rangle \mid x p b_1 b_2 \dots b_n p q y \in L(e), b = \bigvee (b_1 \wedge b_2 \wedge \dots \wedge b_n)\} \\ \text{last}(e) &= \{\langle b, p \rangle \mid x p b_1 b_2 \dots b_n \in L(e), b = \bigvee (b_1 \wedge b_2 \wedge \dots \wedge b_n)\} \end{aligned}$$

Note that the empty disjunction is 1 (and the empty conjunction is 0). Below, we will use expressions of the form  $e!$ , where  $!$  is a special end-marker, to avoid the computation of the last symbols that can be generated in  $e$ :  $\langle b, p \rangle \in \text{last}(e) \Leftrightarrow \langle b, ! \rangle \in \text{follow}(e!, p)$ .

Given a KAT expression  $e$  with all action symbols distinct we construct the automaton  $\mathcal{A}_e = (\text{Pos}(e) \cup \{0\}, \langle o_S, t_S \rangle)$  where  $\text{Pos}(e)$  is the number of distinct action symbols in  $e$  and

$$o_S(i) = \begin{cases} E(e) & \text{if } i = 0 \\ b & \text{if } i > 0 \text{ and } \langle b, ! \rangle \in \text{follow}(e!, p_i) \\ 0 & \text{otherwise} \end{cases}$$

and  $t$  is given by the following rules

$$\begin{array}{ccc} \textcircled{0} \xrightarrow{\langle b, p_i \rangle} \textcircled{j} & \text{iff} & \langle b, p_j \rangle \in \text{first}(e) \\ \textcircled{i} \xrightarrow{\langle b, p_j \rangle} \textcircled{j} & \text{iff} & \langle b, p_j \rangle \in \text{follow}(e!, p_i) \end{array}$$

The way the automaton is defined, state  $i$  will only have incoming transitions labeled by  $\langle b, p_i \rangle$ . Moreover, the fact that  $e$  has distinct symbols implies that the constructed automaton is deterministic, that is,  $t : S \rightarrow \mathcal{B} \times S^{\mathcal{B} \times P}$ . Only after unmarking the labels  $p_i$  non-determinism will be introduced, as we will observe in an example below.

The guarded strings recognized by a state  $s \in S$  of the automaton  $(S, t)$  where  $t : S \rightarrow \mathcal{B} \times \mathcal{P}_\omega(S)^{\mathcal{B} \times P}$  are now defined by the following rule

$$\begin{aligned} x \in G(s) &\Leftrightarrow x = \alpha \text{ with } \alpha \leq E(s) \\ &\text{or } x = \alpha p x' \text{ with } x' \in G(s') \text{ for some } s' \in t_S(s)(\langle b, p \rangle) \\ &\text{and for some } b \text{ s.t. } \alpha \leq b \end{aligned}$$

**3.2.7 THEOREM.** *Let  $e$  be a guarded string, with all action symbols distinct, and let  $\mathcal{A}_e = (\text{Pos}(e) \cup \{0\}, \langle o_S, t_S \rangle)$  be the corresponding automaton constructed as above. Then,  $G(e) = G(0)$ .*

PROOF. By induction on the structure of  $e$ .

If  $e = \mathbf{b}$  then  $GS(\mathbf{b}) = \{\alpha \mid \alpha \leq \mathbf{b}\}$  and  $\mathcal{A}_e$  is a one state automaton with no transitions. Thus,  $G(0) = \{\alpha \mid \alpha \leq E(\mathbf{b})\} = \{\alpha \mid \alpha \leq \mathbf{b}\} = G(\mathbf{b})$ .

If  $e = \mathbf{p}$  then  $GS(\mathbf{p}) = \{\alpha\mathbf{p}\beta \mid \alpha, \beta \in \text{At}\}$  and  $\mathcal{A}_e$  is a two state automaton with only one transition from state 0 (with output  $E(\mathbf{p}) = 0$ ) to state 1 (with output 1) labeled by  $\langle 1, \mathbf{p} \rangle$ . Thus,

$$G(0) = \{\alpha \mid \alpha \leq E(\mathbf{p})\} \cup \{\alpha\mathbf{p}\beta \mid \alpha \leq 1, \beta \leq 1\} = \{\alpha\mathbf{p}\beta \mid \alpha, \beta \in \text{At}\} = G(\mathbf{p})$$

For  $e = e_1 + e_2$ , we have

$$\begin{aligned} & G(0) \\ &= \{\alpha \mid \alpha \leq E(e_1 + e_2)\} \cup \{\alpha\mathbf{p}x' \mid x' \in G(t_S(0)(\langle \mathbf{b}, \mathbf{p} \rangle)), \alpha \leq \mathbf{b}\} \\ &= \{\alpha \mid \alpha \leq E(e_1)\} \cup \{\alpha \mid \alpha \leq E(e_2)\} \cup \\ &\quad \{\alpha\mathbf{p}x' \mid x' \in t_S(0)(\langle \mathbf{b}, \mathbf{p} \rangle), \alpha \leq \mathbf{b}, \mathbf{b}_1\mathbf{b}_2 \dots \mathbf{b}_n\mathbf{p}x \in L(e_1 + e_2), \mathbf{b} = \bigvee(\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \dots \wedge \mathbf{b}_n)\} \\ &= \{\alpha \mid \alpha \leq E(e_1)\} \cup \{\alpha \mid \alpha \leq E(e_2)\} \cup \\ &\quad \{\alpha\mathbf{p}x' \mid x' \in t_S(0)(\langle \mathbf{b}, \mathbf{p} \rangle), \alpha \leq \mathbf{b}, \mathbf{b}_1\mathbf{b}_2 \dots \mathbf{b}_n\mathbf{p}x \in L(e_1), \mathbf{b} = \bigvee(\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \dots \wedge \mathbf{b}_n)\} \cup \\ &\quad \{\alpha\mathbf{p}x' \mid x' \in t_S(0)(\langle \mathbf{b}, \mathbf{p} \rangle), \alpha \leq \mathbf{b}, \mathbf{b}_1\mathbf{b}_2 \dots \mathbf{b}_n\mathbf{p}x \in L(e_2), \mathbf{b} = \bigvee(\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \dots \wedge \mathbf{b}_n)\} \\ &\stackrel{IH}{=} G(e_1) \cup G(e_2) \\ &= G(e_1 + e_2) \end{aligned}$$

Note that  $\mathbf{b}_1\mathbf{b}_2 \dots \mathbf{b}_n\mathbf{p}x \in L(e_i)$ , for  $i = 1, 2$ , if and only if the state 0 of the automaton  $\mathcal{A}_{e_i}$  has a transition labeled by  $\langle \mathbf{b}, \mathbf{p} \rangle$  into some state.

For  $e = e_1e_2$ , things get slightly more complicated. Let us start by the easy bit:

$$\alpha \in G(0) \Leftrightarrow \alpha \leq E(e_1e_2) \Leftrightarrow \alpha \leq E(e_1) \text{ and } \alpha \leq E(e_2) \Leftrightarrow \alpha \in G(e_1e_2)$$

Now take  $\alpha_1\mathbf{p}_1 \dots \mathbf{p}_{n-1}\alpha_n \in G(0)$ . This means that there exists a sequence of transitions:

$$\begin{array}{ccccccc} 0 & \xrightarrow{\langle \mathbf{b}_1, \mathbf{p}_1 \rangle} & \bullet & \xrightarrow{\langle \mathbf{b}_2, \mathbf{p}_2 \rangle} & \dots & \xrightarrow{\langle \mathbf{b}_{n-1}, \mathbf{p}_{n-1} \rangle} & \bullet \\ & & & & & & \downarrow \\ & & & & & & \mathbf{b}_n \end{array}$$

such that  $\alpha_i \leq \mathbf{b}_i$ , for all  $i = 1, \dots, n$ . Because all the symbols in  $e_1e_2$  are distinct we can divide the above sequence of transitions as follows. Let  $p_k$  be the last action symbol in belonging to  $e_1$ . We have

$$\begin{array}{ccccccc} 0 & \xrightarrow{\langle \mathbf{b}_1, \mathbf{p}_1 \rangle} & \bullet & \xrightarrow{\langle \mathbf{b}_2, \mathbf{p}_2 \rangle} & \dots & \xrightarrow{\langle \mathbf{b}_k, \mathbf{p}_k \rangle} & \bullet \\ & & & & & \swarrow \langle \mathbf{b}_{k+1}, \mathbf{p}_{k+1} \rangle & \\ \bullet & \xleftarrow{\langle \mathbf{b}_{k+2}, \mathbf{p}_{k+2} \rangle} & \bullet & \xrightarrow{\langle \mathbf{b}_{k+3}, \mathbf{p}_{k+3} \rangle} & \dots & \xrightarrow{\langle \mathbf{b}_{n-1}, \mathbf{p}_{n-1} \rangle} & \bullet \\ & & & & & & \downarrow \\ & & & & & & \mathbf{b}_n \end{array}$$

and we observe that  $\mathbf{b}_{k+1}$  is such that  $x\mathbf{p}_k\mathbf{b}_{k+1}\mathbf{p}_{k+1}\mathbf{y} \in L(e_1e_2)$ . Thus,  $\mathbf{b}_{k+1} = \mathbf{b}_{k+1}^1\mathbf{b}_{k+1}^2$  such that  $x\mathbf{p}_k\mathbf{b}_{k+1}^1 \in L(e_1)$  and  $\mathbf{b}_{k+1}^2\mathbf{p}_{k+1}\mathbf{y} \in L(e_2)$  and, as a consequence,

$$\langle \mathbf{b}_{k+1}^1, \mathbf{p}_k \rangle \in \text{last}(e_1) \text{ and } \langle \mathbf{b}_{k+1}^2, \mathbf{p}_{k+1} \rangle \in \text{first}(e_2)$$

Now we can conclude using the induction hypothesis since  $\alpha_1\mathbf{p}_1 \dots \alpha_k\mathbf{p}_k\alpha_{k+1} \in G(0_1)$ , where  $0_1$  is the state 0 of  $\mathcal{A}_{e_1}$ , and  $\alpha_{k+1}\mathbf{p}_{k+1} \dots \alpha_{n-1}\mathbf{p}_{n-1}\alpha_n \in G(0_2)$ , where  $0_2$  denotes the state 0 of  $\mathcal{A}_{e_2}$ , and therefore:

$$\begin{aligned} & \alpha_1\mathbf{p}_1 \dots \alpha_k\mathbf{p}_k\alpha_{k+1} \in G(e_1) \text{ and } \alpha_{k+1}\mathbf{p}_{k+1} \dots \alpha_{n-1}\mathbf{p}_{n-1}\alpha_n \in G(e_2) \\ \Leftrightarrow & \alpha_1\mathbf{p}_1 \dots \alpha_k\mathbf{p}_k\alpha_{k+1}\mathbf{p}_{k+1} \dots \alpha_{n-1}\mathbf{p}_{n-1}\alpha_n \in G(e_1e_2) \end{aligned}$$

The case  $e^*$  follows a similar reasoning as in  $e_1e_2$  and is left to the reader.  $\square$

This theorem refers to marked expressions. Note however that unmarking the labels of the automaton only changes the action symbols and it will also yield  $G(\overline{0}) = \overline{G(0)}$ , where  $G(\overline{0})$  denotes the set of guarded strings recognized by state 0 of the unmarked automaton and  $\overline{G(0)}$  the unmarking of the set of guarded strings recognized by state 0 of the marked automaton.

Next, we present an algorithm to compute the sets *first*, *follow* and *last*.

**3.2.8 PROPOSITION.** *Let  $e$  be a KAT expression with distinct symbols.  $\mathbf{F}$ , defined by the rules below, is such that  $\mathbf{F}(e, \{\{1, !\}\})$  yields a set of pairs of the form  $\langle p_i, \text{follow}(e!, p_i) \rangle$ . The rules are:*

$$\begin{aligned} \mathbf{F}(e_1 + e_2, S) &= \mathbf{F}(e_1, S) \cup \mathbf{F}(e_2, S) \\ \mathbf{F}(e_1.e_2, S) &= \mathbf{F}(e_1, \text{first}(e_2) \cup E(e_2).S) \cup \mathbf{F}(e_2, S) \\ \mathbf{F}(e_1^*, S) &= \mathbf{F}(e_1, \text{first}(e_1) \cup S) \\ \mathbf{F}(\mathbf{p}, S) &= \{\langle \mathbf{p}, S \rangle\} \\ \mathbf{F}(\mathbf{b}, S) &= \emptyset \end{aligned}$$

where

$$\begin{aligned} \text{first}(e_1 + e_2) &= \text{first}(e_1) \cup \text{first}(e_2) \\ \text{first}(e_1.e_2) &= \text{first}(e_1) \cup E(e_1).\text{first}(e_2) \\ \text{first}(e_1^*) &= \text{first}(e_1) \\ \text{first}(\mathbf{p}) &= \{\langle 1, \mathbf{p} \rangle\} \\ \text{first}(\mathbf{b}) &= \emptyset \end{aligned}$$

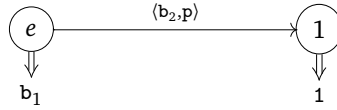
Note the similarities between Propositions 3.2.8 and 3.1.11. The fact that the Boolean algebra  $\mathcal{B}$  generalizes the two element Boolean algebra of classical regular expressions is reflected in the clause for the concatenation in the following way. The test for empty word  $o_{\mathcal{R}}$  is replaced by the Boolean value of a KAT expression  $e$  and the multiplication is now redefined to propagate the tests:

$$\mathbf{b}.S = \begin{cases} \emptyset & \text{if } \mathbf{b} = 0 \\ \{\langle \mathbf{b}\mathbf{b}', \mathbf{p} \rangle \mid \langle \mathbf{b}', \mathbf{p} \rangle \in S\} & \text{otherwise} \end{cases}$$

**3.2.9 EXAMPLE.** We show now two examples of the algorithm above. We start by applying to the expression  $e = b_1 + b_2p$ , which we already used in Examples 3.2.3 and 3.2.5. This expression already has all action symbols distinct so no marking is needed. First, we compute  $F(e, \{ \langle 1, ! \rangle \})$ :

$$F(b_1 + b_2p, \{ \langle 1, ! \rangle \}) = F(b_1, \{ \langle 1, ! \rangle \}) \cup F(b_2p, \{ \langle 1, ! \rangle \}) = F(p, \{ \langle 1, ! \rangle \}) = \{ \langle p, \{ \langle 1, ! \rangle \} \rangle \}$$

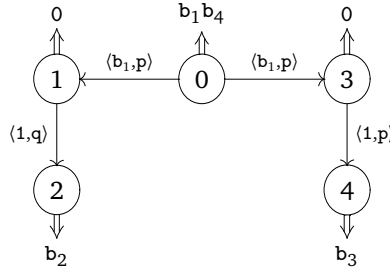
Thus, because  $first(e) = \{ \langle b_2, p \rangle \}$  and  $E(e) = b_1$ , we have that  $\mathcal{A}_e$  is given by



Next, we consider the expression  $e_1 = b_1(pq_2b_2 + pp_4b_3 + b_4)$ . We have  $E(e_1) = b_1b_4$ ,  $\overline{e_1} = b_1(p_1q_2b_2 + p_3p_4b_3 + b_4)$  and

$$\begin{aligned} & first(\overline{e_1}) \\ &= first(b_1) \cup E(b_1).first((p_1q_2b_2 + p_3p_4b_3 + b_4)) \\ &= b_1 \cdot \{ \langle 1, p_1 \rangle, \langle 1, p_3 \rangle \} = \{ \langle b_1, p_1 \rangle, \langle b_1, p_3 \rangle \} \\ & F(\overline{e_1}, \{ \langle 1, ! \rangle \}) \\ &= F(p_1q_2b_2 + p_3p_4b_3 + b_4, \{ \langle 1, ! \rangle \}) \\ &= F(p_1q_2b_2, \{ \langle 1, ! \rangle \}) \cup F(p_3p_4b_3, \{ \langle 1, ! \rangle \}) \\ &= F(p_1, \{ \langle q_2, 1 \rangle \}) \cup F(q_2, E(b_2) \cdot \{ \langle 1, ! \rangle \}) \cup F(p_3, \{ \langle 1, p_4 \rangle \}) \cup F(p_4, E(b_3) \cdot \{ \langle 1, ! \rangle \}) \\ &= \{ \langle p_1, \{ \langle 1, q_2 \rangle \} \rangle, \langle q_2, \{ \langle b_2, ! \rangle \} \rangle, \langle p_3, \{ \langle 1, p_4 \rangle \} \rangle, \langle p_4, \{ \langle b_3, ! \rangle \} \rangle \} \end{aligned}$$

The automaton  $\mathcal{A}_{e_1}$ , after unmarking, is then given by:



The non-deterministic version of Kozen's automata on guarded strings would have 7 states and 8 transitions, whereas the (minimal) deterministic version would have 5 states (same as the automaton above), but  $8 \times 8 = 64$  transitions since for  $B = \{b_1, b_2, b_3\}$  the set  $\text{At}$  has 8 elements. ♠

Regular expressions were first introduced by Kleene [64] to study the properties of neural networks. They are an algebraic description of languages, offering a declarative way of specifying the strings to be recognized and they define exactly the same class of languages accepted by deterministic (and non-deterministic) finite state automata: the regular languages. The fundamental correspondence between regular expressions and deterministic automata has been formalized in Kleene's theorem: each regular expression denotes a language that can be recognized by a deterministic automaton and, vice-versa, the language accepted by a deterministic automaton can be specified by a regular expression. Languages denoted by regular expressions are called *regular*. Two regular expressions are called (language) equivalent if they denote the same regular language. Algebraic reasoning on equivalence of regular expressions was made possible by the sound and complete axiomatization introduced by Salomaa [101], which was later refined by Kozen, who showed that Salomaa's axiomatization is non-algebraic, in the sense that it is unsound under substitution of alphabet symbols by arbitrary regular expressions, and who presented an algebraic axiomatization in [66].

A Mealy machine  $(S, \alpha)$  is a pair consisting of a set  $S$  of states and a transition function  $\alpha: S \rightarrow (B \times S)^A$  assigning to each state  $s \in S$  and input symbol  $a \in A$  a pair  $\langle b, s' \rangle$ , containing an output symbol  $b \in B$  and a next state  $s' \in S$ . Mealy machines can be seen as a slight variation of deterministic automata, where each input is now associated with an output. The semantics of Mealy machines is given by  $B^{A^+}$ , which is the counterpart of formal languages (the set  $2^{A^+}$ ). Alternatively, and equivalently, the semantics of Mealy machines is given by the set of causal functions from infinite sequences of inputs  $A^\omega$  to infinite sequences of outputs  $B^\omega$ . A function is causal if the  $n$ -th output depends only on the first  $n$  inputs.

One of the most important applications of Mealy machines is their use in the specification of sequential digital circuits. Taking binary inputs and outputs, there is a well-known correspondence between such binary Mealy machines, on the one hand, and sequential digital circuits built out of logical gates and some kind of memory elements, on the other. In present day text books on logic design [79] — on the

construction of sequential digital circuits — Mealy machines are still the most important mathematically exact means for the specification of the intended behaviour of circuits. There does not seem to exist, however, a generally accepted way of formally specifying Mealy machines themselves. They are typically “defined” in a natural language such as English. This obviously leads to ambiguities, inconsistencies and plain errors [37].

Kleene’s regular expressions were proposed in [28, 80] as unambiguous descriptions of circuits. However, since regular expressions were tailor made for deterministic automata, the conversion between regular expressions and Mealy machines is not the most natural one. Moreover, they can only be used as a specification for binary Mealy machines.

It is the aim of the present chapter to introduce a simple but expressive language for the specification of Mealy machines, in the same spirit of the language Kleene introduced for deterministic automata. We present the counterpart of Kleene’s theorem: we show that every finite Mealy machine can be represented by an expression in the language and, conversely, from every expression a (behaviorally equivalent) Mealy machine can be constructed. Furthermore, we introduce a sound and complete axiomatization of the language, allowing for algebraic reasoning on equivalence of specifications.

Our approach is coalgebraic: Mealy machines are a basic and well-understood family of coalgebras, of the **Set** functor  $\mathcal{M}(S) = (B \times S)^A$ . The functor, which determines the transition type, induces a natural semantics (the so-called final coalgebra) and equivalence. We will fit our language in the coalgebraic framework and we will profit from known coalgebraic techniques along the way in the proofs. Although in this chapter the generality of the approach might not be fully clear, we will show in the next chapter how the coalgebraic approach taken in this chapter paves the way to lift all the results and techniques described here to a much more general class of systems.

**Organization of the chapter.** Section 4.1 recalls the basic definitions on Mealy machines, including the notion of bisimilarity. We introduce a set of expressions  $\text{Exp}_{\mathcal{M}}$  to specify (behaviours of) Mealy machines and we prove the analogue of Kleene’s theorem in Section 4.2. In Section 4.3 we present a sound and complete axiomatization, with respect to bisimilarity, of  $\text{Exp}_{\mathcal{M}}$ . Section 4.4 presents concluding remarks and discusses related work.

## 4.1 Mealy machines

We give the basic definitions on Mealy machines and introduce the notion of bisimulation.

First we recall the following definition. A (bounded) join-semilattice is a set  $B$  equipped with a binary operation  $\vee_B$  and a constant  $\perp_B \in B$ , such that  $\vee_B$  is commutative, associative and idempotent. The element  $\perp_B$  is neutral with respect to  $\vee_B$ . As usual,

$\vee_B$  gives rise to a partial ordering  $\leq_B$  on the elements of  $B$ :

$$b_1 \leq_B b_2 \Leftrightarrow b_1 \vee_B b_2 = b_2$$

We use semilattices to represent data structures equipped with an information order:  $b_1 \leq_B b_2$  means that  $b_1$  is less *concrete* than  $b_2$ .

Now let  $A$  be a finite set and let  $B$  be a join-semilattice. A Mealy machine  $(S, \alpha)$  with inputs in  $A$  and outputs in  $B$  consists of a set of states  $S$  together with a function

$$\alpha: S \rightarrow (B \times S)^A$$

For a given state  $s \in S$  and an input  $a \in A$ , the function  $\alpha$  returns a pair  $\alpha(s)(a) = \langle b, s' \rangle$ , consisting of an output value  $b \in B$  and a state  $s' \in S$ . Typically we will write

$$\alpha(s)(a) = \langle s[a], s_a \rangle$$

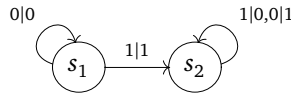
and call  $s[a]$  the (*initial*) *output on input  $a$*  and  $s_a$  the *next state on input  $a$* . We will also sometimes refer to  $s_a$  as the  *$a$ -derivative of  $s$* . We shall also use the following convention for the representation of Mealy machines:

$$\alpha(s)(a) = \langle b, s' \rangle \Leftrightarrow \begin{array}{c} \text{---} a|b \text{---} \\ \circ s \longrightarrow \circ s' \end{array}$$

The usual definition of Mealy machines takes  $B$  to be a set instead of a join-semilattice. This minor variation on the output set will play an important role in defining the semantics of the expressions that we will associate with Mealy machines. This is similar to what happens with deterministic automata and regular expressions, where the set  $\Sigma$  of outputs in the automaton is in fact considered implicitly to be a join-semilattice. This fact becomes evident looking at any of the axiomatizations of regular expressions.

Mealy machines where  $A$  is the two-element set  $\{0, 1\}$  and  $B$  is the two-element join-semilattice  $\{0, 1\}$  (with  $\perp_B = 0$ ) are called *binary*.

For an example, consider the following binary Mealy machine with  $S = \{s_1, s_2\}$  and the transition function defined by the following picture.



This machine (or more precisely, state  $s_1$ ) computes the two's complement of a given binary number.

In algebraic terms, a Mealy machine is a coalgebra of the functor  $\mathcal{M}: \mathbf{Set} \rightarrow \mathbf{Set}$  which is defined, for any set  $X$ , as  $\mathcal{M}(X) = (B \times X)^A$  and, for a function  $h: X \rightarrow Y$ ,  $\mathcal{M}(h): (B \times X)^A \rightarrow (B \times Y)^A$ , is given by

$$\mathcal{M}(h)(\psi)(a) = \langle b, h(x) \rangle \text{ where } \langle b, x \rangle = \psi(a)$$

**4.1.1 DEFINITION.** A *homomorphism* from a Mealy machine  $(S, \alpha)$  to a Mealy machine  $(T, \beta)$  is a function  $h: S \rightarrow T$  preserving initial outputs and next states:

$$\begin{array}{ccc}
 S & \xrightarrow{h} & T \\
 \alpha \downarrow & & \downarrow \beta \\
 (\mathbb{B} \times S)^A & \xrightarrow{(id \times h)^A} & (\mathbb{B} \times T)^A
 \end{array}
 \quad
 h(s)[a] = s[a] \quad \text{and} \quad h(s_a) = h(s)_a$$

Note that the output  $(-)[a]$  and next state  $(-)_a$  functions in the left and right side of the equations above refer to  $\alpha$  and  $\beta$ , respectively. ♣

Next we define the notion of bisimulation, which plays an important role in the minimization of Mealy machines and in defining a notion of equivalence for expressions.

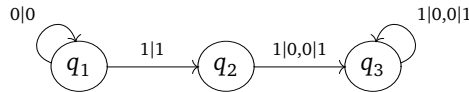
**4.1.2 DEFINITION (Bisimulation for Mealy).** Let  $(S, \alpha)$  and  $(T, \beta)$  be two Mealy machines. We call a relation  $R \subseteq S \times T$  a *bisimulation* if for all  $\langle s, t \rangle \in S \times T$  and  $a \in A$

$$s R t \Rightarrow (s[a] = t[a] \text{ and } s_a R t_a)$$

♣

We write  $s \sim t$  whenever there exists a bisimulation relation containing  $\langle s, t \rangle$ ; and we call  $\sim$  the bisimilarity relation. It is worth to remark that this notion of bisimulation is precisely the notion one gets by instantiating the more general notion of  $\mathcal{F}$ -bisimulation (Definition 2.2.4) to the functor  $\mathcal{M}(X) = (\mathbb{B} \times X)^A$  determining the transition type of Mealy machines.

As an example, consider the following binary Mealy machine:



Observe that  $q_3$  and  $q_2$  are bisimilar, since  $R = \{(q_2, q_3), (q_3, q_3)\}$  is a bisimulation. A minimal machine is obtained by identifying all bisimilar states, yielding the two's complement machine presented above.

Next we recall the construction of a *final* Mealy machine with inputs in  $A$  and outputs in  $\mathbb{B}$ . A Mealy machine  $(\Omega, \omega)$  is said to be final if for any Mealy machine  $(S, \alpha)$  there is a unique homomorphism  $[[ - ]]_S: (S, \alpha) \rightarrow (\Omega, \omega)$ . Finality plays an important role in providing semantics to the expressions as well as in the proof of soundness and completeness of the axiomatization (in Section 4.2).

Let  $A^\omega = \{\sigma \mid \sigma: \mathbb{N} \rightarrow A\}$ , the set of all infinite *streams* over  $A$ . For  $a \in A$  and  $\sigma \in A^\omega$ , we define:

$$a:\sigma = (a, \sigma(0), \sigma(1), \sigma(2), \dots) \quad \sigma' = (\sigma(1), \sigma(2), \sigma(3), \dots)$$



We call a function  $f : A^\omega \rightarrow B^\omega$  *causal* if for all  $\sigma \in A^\omega$  and  $n \geq 0$ , the  $n$ -th output value  $f(\sigma)(n)$  depends only on the first  $n$  input values  $(\sigma(0), \dots, \sigma(n-1))$ . Formally, a function  $f : A^\omega \rightarrow B^\omega$  is causal if, for all  $\sigma, \tau \in A^\omega$  and for all  $n \in \mathbb{N}$ ,

$$\text{if } \sigma(k) = \tau(k), \text{ for all } k \leq n, \text{ then } f(\sigma)(n) = f(\tau)(n).$$

Let  $\Phi = \{f : A^\omega \rightarrow B^\omega \mid f \text{ is causal}\}$ ,  $f \in \Phi$  and  $a \in A$ . The initial output of  $f$ , on input  $a$ , is defined as

$$f[a] = f(a:\sigma)(0) \text{ (where } \sigma \text{ is arbitrary).}$$

The derivative of  $f$ , on input  $a$ , is the function  $f_a$  defined, for all  $\sigma \in A^\omega$ , as

$$f_a(\sigma) = (f(a:\sigma))'.$$

The set  $\Phi$  can be turned into a Mealy machine  $(\Phi, \varphi)$  by defining  $\varphi(f)(a) = \langle f[a], f_a \rangle$ . Note that, by causality, the value of  $f(a:\sigma)(0)$  depends only on  $a$  and thus the output  $f[a]$  is well-defined. Moreover, the derivative  $f_a$  of a causal function  $f$  is again a causal function: take any  $\sigma, \tau \in A^\omega$  and  $n \in \mathbb{N}$ ; if  $\sigma(k) = \tau(k)$ , for all  $k \leq n$ , then:

$$f_a(\sigma)(n) = f(a:\sigma)(n+1) = f(a:\tau)(n+1) = f_a(\tau)(n)$$

where the one but last step follows by causality of  $f$ .

Rutten [99] showed that the coalgebra  $(\Phi, \varphi)$  is final.

**4.1.3 THEOREM** (Finality of  $(\Phi, \varphi)$  [99, Proposition 2.2]). *For every Mealy machine  $(S, \alpha)$  there exists a unique homomorphism  $[[ - ] ]_S : S \rightarrow \Phi$ .*

From the finality of  $(\Phi, \varphi)$  it follows that the subcoalgebra  $\langle f \rangle$  generated by a causal function  $f$  is a minimal Mealy coalgebra [99, Corollary 2.3].

The final coalgebra of the Mealy functor  $\mathcal{M}(X) = (B \times X)^A$  can alternatively be characterized in the following manner. Observe that  $(B \times X)^A \cong B^A \times X^A$  and thus every Mealy machine is isomorphic to a Moore automaton with inputs in  $A$  and outputs in  $B^A$ . In [97], it was proved that the final coalgebra of this automaton (type) is  $(B^A)^{A^*} \cong B^{A^+}$ . Since final coalgebras are unique up to isomorphism, the set of casual functions is then isomorphic to the set of functions from  $A^+$  to  $B$ .

## 4.2 Regular expressions for Mealy machines

We present a language for Mealy machines and define its semantics in terms of causal functions. We prove the analogue of Kleene's theorem for this language: every state of a Mealy machine can be assigned to an expression in the language denoting the same behaviour and, conversely, every expression in the language can be transformed into an equivalent Mealy machine. In Section 4.3, we shall introduce an axiomatization of the language and prove it sound and complete with respect to bisimulation.

**4.2.1 DEFINITION** (Mealy expressions). Let  $A$  be a set of input actions and let  $B$  be a join-semilattice of output actions. Furthermore, let  $X$  be a set of (recursion or) fixed

point variables. The set  $\text{Exp}$  of *expressions* is given by the following BNF syntax. For  $a \in A$ ,  $b \in B$ , and  $x \in X$ :

$$\varepsilon ::= \underline{\emptyset} \mid x \mid a(\varepsilon) \mid a \downarrow b \mid \varepsilon \oplus \varepsilon \mid \mu x. \gamma$$

where  $\gamma \in \text{Exp}_g$ , the set of *guarded expressions*, which is given by:

$$\gamma ::= \underline{\emptyset} \mid a(\varepsilon) \mid a \downarrow b \mid \gamma \oplus \gamma \mid \mu x. \gamma$$

The set  $\text{Exp}_{\mathcal{M}}$  of *Mealy expressions* contains the closed and guarded expressions  $\varepsilon$  in  $\text{Exp}$ . An expression is closed if a variable  $x$  always occurs under the scope of the binder  $\mu x$ . ♣

Intuitively,  $a(\phi)$  represents a state that for input  $a$  has a transition to the state specified by  $\phi$ , whereas  $a \downarrow b$  represents a state that outputs  $b$  for input  $a$ . Combining both expressions with  $\oplus - a(\phi) \oplus a \downarrow b -$  yields a state which is fully specified for input  $a$ . Here, we start seeing the need for the join-semilattice structure on  $B$ : if the output of a state, for a certain input, is not specified we can associate with that transition output  $\perp$ . This will become clearer below in the examples.

**Notation:** For an ordered set  $E = \{\varepsilon_1, \dots, \varepsilon_n\}$  of expressions we will define

$$\bigoplus E = \varepsilon_1 \oplus (\varepsilon_2 \oplus (\dots))$$

The ordering convention is not important, as we will soon observe (Lemma 4.2.4). Later, we will axiomatize  $\oplus$  to be an associative, commutative and idempotent operation.

### 4.2.1 Expressions form a Mealy coalgebra

We turn the set  $\text{Exp}_{\mathcal{M}}$  into a Mealy machine (coalgebra)

$$\delta : \text{Exp}_{\mathcal{M}} \rightarrow (B \times \text{Exp}_{\mathcal{M}})^A$$

by defining  $\delta$  as follows.

**4.2.2 DEFINITION.** For  $a \in A$  and  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ , we write  $\delta(\varepsilon) = \langle \varepsilon[a], \varepsilon_a \rangle$  and we define  $\varepsilon[a]$  and  $\varepsilon_a$  by

$$\begin{array}{ll} \underline{\emptyset}[a] & = \perp_B & (\underline{\emptyset})_a & = \underline{\emptyset} \\ a(\varepsilon)[a'] & = \perp_B \text{ (for any } a' \in A) & (a(\varepsilon))_{a'} & = \begin{cases} \varepsilon & \text{if } a = a' \\ \underline{\emptyset} & \text{otherwise} \end{cases} \\ (a \downarrow b)[a'] & = \begin{cases} b & \text{if } a = a' \\ \perp_B & \text{otherwise} \end{cases} & (a \downarrow b)_{a'} & = \underline{\emptyset} \text{ (for any } a' \in A) \\ (\varepsilon_1 \oplus \varepsilon_2)[a] & = \varepsilon_1[a] \vee_B \varepsilon_2[a] & (\varepsilon_1 \oplus \varepsilon_2)_a & = (\varepsilon_1)_a \oplus (\varepsilon_2)_a \\ (\mu x. \gamma)[a] & = (\gamma[\mu x. \gamma/x])[a] & (\mu x. \gamma)_a & = (\gamma[\mu x. \gamma/x])_a \end{array}$$

Here,  $\gamma[\mu x. \gamma/x]$  denotes syntactic substitution, replacing in  $\gamma$  every free occurrence of  $x$  by  $\mu x. \gamma$ . ♣

Note the similarities between this definition and the definition of Brzozowski derivatives for regular expressions, presented in the previous chapter.

The above definition uses induction on the following complexity measure, which is based on the number of nested unguarded occurrences of  $\mu$  expressions.

**4.2.3 DEFINITION.** For any  $\varepsilon \in \text{Exp}_g$ , we define, by induction on the structure of  $\varepsilon$ , the following complexity measure:

$$\begin{aligned} N(\emptyset) &= N(a \downarrow b) = N(a(\varepsilon)) = 0 \\ N(\varepsilon_1 \oplus \varepsilon_2) &= \max\{N(\varepsilon_1), N(\varepsilon_2)\} + 1 \\ N(\mu x.\gamma) &= 1 + N(\gamma) \end{aligned}$$

♣

In order to see that the definitions of  $\varepsilon[a]$  and  $\varepsilon_a$  in Definition 4.2.2 are well-formed, note that in the case of  $\mu x.\gamma$ , we have  $N(\mu x.\gamma) > N(\gamma[\mu x.\gamma/x])$ , since:

$$N(\gamma) = N(\gamma[\mu x.\gamma/x])$$

This can easily be proved by (standard) induction on the syntactic structure of  $\gamma$ , since  $\gamma$  is guarded (in  $x$ ).

The coalgebra structure on  $\text{Exp}_M$  enables us to use the notion of bisimulation for Mealy machines (Definition 4.1.2) to prove the following lemma, which shows that the definition of  $\oplus$  above yields bisimilar expressions independently of the order chosen.

**4.2.4 LEMMA.** Let  $E$  be an ordered set and  $E'$  a set obtained as a permutation of  $E$ . Then, the following holds:

$$\bigoplus E \sim \bigoplus E'$$

PROOF. Let  $E = \{\varepsilon_1, \dots, \varepsilon_n\}$  and  $E' = \{\varepsilon_{\sigma(1)}, \dots, \varepsilon_{\sigma(n)}\}$ , where  $\sigma$  is a permutation function. We prove that the relation

$$R = \{(\bigoplus E_w, \bigoplus E'_w) \mid w \in A^*\}$$

is a bisimulation. Here,  $E_w$  (respectively  $E'_w$ ) denote the pointwise application to the elements of  $E$  (respectively  $E'$ ) of the word derivative  $\varepsilon_w$  defined inductively on the length of  $w \in A^*$  by  $\varepsilon_\varepsilon = \varepsilon$  and  $\varepsilon_{aw'} = (\varepsilon_a)_{w'}$ .

To prove that  $R$  is a bisimulation, we observe that, for any  $w \in A^*$  and  $a \in A$ , we have

$$\begin{aligned} & (\bigoplus E_w)[a] \\ = & ((\varepsilon_1)_w \oplus ((\varepsilon_2)_w \oplus (\dots)))[a] && \text{(def. } \oplus) \\ = & (\varepsilon_1)_w[a] \vee_B ((\varepsilon_2)_w[a] \vee_B (\dots)) && \text{(def. } (-)[a]) \\ = & (\varepsilon_{\sigma(1)})_w[a] \vee_B ((\varepsilon_{\sigma(2)})_w[a] \vee_B (\dots)) && (\vee_B \text{ is associative and commutative)} \\ = & (\bigoplus E'_w)[a] \\ \\ & (\bigoplus E_w)_a \\ = & (\varepsilon_1)_{wa} \oplus ((\varepsilon_2)_{wa} \oplus (\dots)) && \text{(def. } \oplus \text{ and } (-)_a) \\ R & (\varepsilon_{\sigma(1)})_{wa} \oplus ((\varepsilon_{\sigma(2)})_{wa} \oplus (\dots)) && \text{(definition of } R) \\ = & (\bigoplus E'_w)_a \end{aligned}$$

□

### 4.2.2 A Kleene theorem for Mealy coalgebras

Having a Mealy coalgebra structure on  $\text{Exp}_{\mathcal{M}}$  provides us, by finality of  $\Phi$ , directly with a natural semantics because of the existence of a (unique) homomorphism:

$$\begin{array}{ccc} \text{Exp}_{\mathcal{M}} & \xrightarrow{\llbracket \cdot \rrbracket} & \Phi \\ \delta \downarrow & & \downarrow \varphi \\ (\text{B} \times \text{Exp}_{\mathcal{M}})^A & \xrightarrow{(id \times \llbracket \cdot \rrbracket)^A} & (\text{B} \times \Phi)^A \end{array} \quad \llbracket \varepsilon \rrbracket[a] = \varepsilon[a] \quad \text{and} \quad \llbracket \varepsilon \rrbracket_a = \llbracket \varepsilon_a \rrbracket$$

Here, we drop the subscript in  $\llbracket - \rrbracket$  (recall that  $\llbracket - \rrbracket_S: (S, \alpha) \rightarrow (\Phi, \varphi)$  denoted the unique homomorphism into the final coalgebra): we will only use  $\llbracket - \rrbracket$ , without subscript, to refer to  $\llbracket - \rrbracket_{\text{Exp}_{\mathcal{M}}}$ . The map  $\llbracket - \rrbracket$  assigns to every expression  $\varepsilon$  a causal stream function  $\llbracket \varepsilon \rrbracket: A^\omega \rightarrow \text{B}^\omega$ .

This is completely analogous to what happens with deterministic automata and regular expressions. The latter are provided with a deterministic automaton structure, given by the Brzozowski derivatives, and then each regular expression is mapped, via the final homomorphism, to the language that it denotes (see diagram (3.3)). Intuitively, one can think of  $\delta_{\mathcal{M}}$  as the analogue of Brzozowski derivatives [29] and causal functions as the counterpart of languages in the Mealy setting.

**4.2.5 EXAMPLE.** Let  $A = \{a, b\}$ , let  $\text{B} = \{1, 0\}$  (with  $\perp_{\text{B}} = 0$ ) and let  $\varepsilon = \mu x. a \downarrow 1 \oplus a(x)$ . The semantics of  $\varepsilon$  is given by the causal function  $f = \llbracket \varepsilon \rrbracket$ , defined by:

$$\begin{aligned} f(a:\sigma) &= 1: f(\sigma') \\ f(b:\sigma) &= (0, 0, 0, \dots) \end{aligned}$$

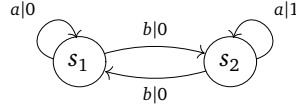
♠

We can now define when a state  $s \in S$  of a Mealy machine  $(S, \alpha)$  is equivalent to an expression  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ . We say that  $s$  is equivalent to  $\varepsilon$  if and only if  $\llbracket s \rrbracket_S = \llbracket \varepsilon \rrbracket$ . Keeping in mind the parallel with deterministic automata, this is the analogue of saying that the language recognized by a state in a deterministic automata is equal to the language denoted by a regular expression.

This is equivalent to saying that  $s \sim \varepsilon$  and, thus, proving the equivalence of  $s$  and  $\varepsilon$  amounts to the construction of a bisimulation relation  $R \subseteq S \times \text{Exp}_{\mathcal{M}}$  between the Mealy coalgebras  $(S, \alpha)$  and  $(\text{Exp}_{\mathcal{M}}, \delta)$  such that  $\langle s, \varepsilon \rangle \in R$ .

Given two Mealy coalgebras  $(S, \alpha)$  and  $(T, \beta)$ , the following fact holds, for  $s \in S$  and  $t \in T$ :  $s \sim t \Leftrightarrow \llbracket s \rrbracket_S = \llbracket t \rrbracket_T$ . As mentioned in Chapter 2, the notion of bisimulation and the equivalence induced by the final coalgebra coincide for many functors, including the Mealy one, but, unfortunately, this is not always the case. Since we are in safe ground in this chapter we will use both notions indistinctly. In Chapter 6, where we consider functors for which this fact does not hold, the subtlety behind this issue will become clearer.

**4.2.6 EXAMPLE.** Let  $A = \{a, b\}$ , let  $B = \{1, 0\}$  (with  $\perp_B = 0$ ) and let  $(S, \alpha)$  be the Mealy coalgebra depicted in the following picture



The expression  $\varepsilon_1 = \mu x.a(x) \oplus b(\mu y.a(y) \oplus a\downarrow 1 \oplus b(x))$  is equivalent to  $s_1$ . To prove this, first define  $\varepsilon_2 = \mu y.a(y) \oplus a\downarrow 1 \oplus b(\mu x.a(x) \oplus b(y))$ ,  $\varepsilon_3 = \mu y.a(y) \oplus a\downarrow 1 \oplus b(\varepsilon_1)$  and  $\varepsilon_4 = \mu x.a(x) \oplus b(\varepsilon_2)$ . Next, note that

$$\begin{array}{llll} \varepsilon_1[a] = 0 & \varepsilon_1[b] = 0 & (\varepsilon_1)_a = \varepsilon_1 & (\varepsilon_1)_b = \varepsilon_3 \\ \varepsilon_2[a] = 1 & \varepsilon_2[b] = 0 & (\varepsilon_2)_a = \varepsilon_2 & (\varepsilon_2)_b = \varepsilon_4 \\ \varepsilon_3[a] = 1 & \varepsilon_3[b] = 0 & (\varepsilon_3)_a = \varepsilon_3 & (\varepsilon_3)_b = \varepsilon_1 \\ \varepsilon_4[a] = 0 & \varepsilon_4[b] = 0 & (\varepsilon_4)_a = \varepsilon_4 & (\varepsilon_4)_b = \varepsilon_2 \end{array}$$

and, thus, the relation

$$R = \{\langle s_1, \varepsilon_1 \rangle, \langle s_2, \varepsilon_2 \rangle, \langle s_2, \varepsilon_3 \rangle, \langle s_1, \varepsilon_4 \rangle\}$$

is a bisimulation, which yields  $s_1 \sim \varepsilon_1$ . ♠

At this point we are ready to state half of the analogue of Kleene's theorem: for every state  $s$  of a finite Mealy machine we can construct an equivalent expression  $\varepsilon_s$ , that is an  $\varepsilon_s$  is such that  $s \sim \varepsilon_s$ .

**4.2.7 THEOREM** (Kleene's theorem for Mealy machines (part I)). *Let  $(S, \alpha)$  be a Mealy machine. If  $S$  is finite then there exists for any  $s \in S$  an expression  $\varepsilon_s \in \text{Exp}_{\mathcal{M}}$  such that  $s \sim \varepsilon_s$ .*

**PROOF.** Let  $S = \{s_1, \dots, s_n\}$ . We construct for a given  $s \in S$  an expression  $\varepsilon_s$  with  $s \sim \varepsilon_s$ . To this end, we associate with every state  $s_i \in S$  a variable  $x_i \in X$  and an expression  $A_i = \mu x_i.\psi_i$ , with  $\psi_i$  defined by

$$\psi_i = \bigoplus_{a \in A} (a(x_{(s_i)_a}) \oplus a\downarrow s_i[a])$$

Then we define  $A_i^1 = A_i$ ,  $A_i^{k+1} = A_i^k \{A_{k+1}^k / x_{k+1}\}$  and we set  $\varepsilon_i = A_i^n$ . Here,  $A\{A'/x\}$  denotes syntactic replacement (that is, substitution without renaming of bound variables in  $A$  which are free in  $A'$ ). This seemingly complicated definition is the analogue of computing the regular expression denoting the language recognized by a state of a deterministic automaton from a system of equations (Section 3.1.1). Below, in an example, the similarities between the system of equations we solve here (using fixed points) in the Mealy case and the one for deterministic automata will become more evident. It should be remarked that above we are implicitly considering the argument set of  $\bigoplus$  to be ordered. As remarked above (Lemma 4.2.4) the ordering is not important.

Note that the term  $A_i^n = (\mu x_i. \psi_i) \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\}$  is closed, due to the fact that, for every  $j = 1, \dots, n$ , the term  $A_j^{i-1}$  contains at most  $n - j$  free variables in the set  $\{x_{j+1}, \dots, x_n\}$ . Moreover,  $A_i^n[a] = s_i[a]$  and  $(A_i^n)_a = A_j^n$  where  $s_j = (s_i)_a$ . Both equations are proved in a similar fashion. We prove the second:

$$\begin{aligned}
& (A_i^n)_a \\
&= ((\mu x_i. \psi_i) \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\})_a \\
&= (\mu x_i. \psi_i \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\})_a \\
&= (\psi_i \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} [A_i^n/x_i])_a \quad (\text{def. of } (-)_a) \\
&= (\psi_i \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} \{A_i^n/x_i\})_a \quad ([A_i^n/x_i] = \{A_i^n/x_i\}) \\
&= (\psi_i \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_i^n/x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\})_a \quad (\dagger) \\
&= \left( \bigoplus_{a \in A} (a(x_{(s_i)_a}) \oplus a \downarrow s_i[a]) \right)_a \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_i^n/x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} \\
&= x_j \{A_j^{j-1}/x_j\} \dots \{A_n^{n-1}/x_n\} \\
&= A_j^n
\end{aligned}$$

Here, note that  $[A_i^n/x_i] = \{A_i^n/x_i\}$ , because  $A_i^n$  has no free variables. The step marked by  $(\dagger)$  follows because  $x_i$  is not free in  $A_{i+1}^i, \dots, A_n^{n-1}$ . The one but last step is a consequence of the definition of  $(-)_a$  and:

$$\begin{aligned}
& \{A_i^n/x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} \\
&= \{A_i^{i-1} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} / x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} \\
&= \{A_i^{i-1}/x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}
\end{aligned} \tag{4.1}$$

Equation (4.1) uses the syntactic identity

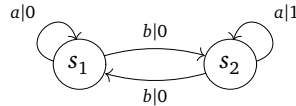
$$A\{B\{C/y\}/x\}\{C/y\} = A\{B/x\}\{C/y\} \tag{4.2}$$

As a consequence of  $A_i^n[a] = s_i[a]$  and  $(A_i^n)_a = A_j^n$  we have that the relation

$$R = \{ \langle s, \varepsilon_s \rangle \mid s \in S \}$$

is a bisimulation and thus  $s \sim \varepsilon_s$ , for all  $s \in S$ .  $\square$

Let us illustrate the construction above. Recall the Mealy machine presented before in Example 4.2.6:



We associate with  $s_1$  and  $s_2$  the variables  $x_1$  and  $x_2$ , respectively, and we define the expressions  $A_1 = \mu x_1.\psi_1$  and  $A_2 = \mu x_2.\psi_2$ , where  $\psi_1$  and  $\psi_2$  are given by

$$\psi_1 = a(x_1) \oplus a \downarrow 0 \oplus b(x_2) \oplus b \downarrow 0 \quad \psi_2 = a(x_2) \oplus a \downarrow 1 \oplus b(x_1) \oplus b \downarrow 0$$

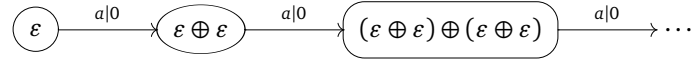
Then, we compute  $A_1^1 = A_1^0 = A_1$ ,  $A_1^2 = A_1^1\{A_2^1/x_2\}$ ,  $A_2^1 = A_2^0\{A_1^0/x_1\} = A_2\{A_1/x_1\}$  and  $A_2^2 = A_2^1$ . This yields the expressions

$$\varepsilon_1 = A_1^2 = \mu x_1.a(x_1) \oplus 0 \downarrow 0 \oplus 1(\varepsilon_2) \oplus 1 \downarrow 1$$

$$\varepsilon_2 = A_2^2 = \mu x_2.a(x_2) \oplus a \downarrow 1 \oplus b(\mu x_1.a(x_1) \oplus a \downarrow 0 \oplus b(x_2) \oplus b \downarrow 0) \oplus b \downarrow 0$$

By construction we have  $s_1 \sim \varepsilon_1$  and  $s_2 \sim \varepsilon_2$ .

The coalgebra structure  $(\text{Exp}_{\mathcal{M}}, \delta)$  also provides us with a way of constructing a Mealy machine from an expression  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ , by considering the subcoalgebra  $\langle \varepsilon \rangle$  (recall that  $\langle \varepsilon \rangle$  denotes the smallest subcoalgebra generated by  $\varepsilon$  as defined in (equation (2.1))). The synthesis of a Mealy machine from an expression  $\varepsilon \in \text{Exp}_{\mathcal{M}}$  is the first step to be able to state and prove the second half of Kleene's theorem for Mealy machines. Note however that  $\langle \varepsilon \rangle$  will in general be infinite, similarly to what happened in the case of regular expressions and Brzozowski derivatives (Section 3.1.2). Consider for instance the expression  $\varepsilon = \mu x.a(x \oplus x)$  and note that  $\varepsilon_a = \varepsilon \oplus \varepsilon$ ,  $(\varepsilon_a)_a = (\varepsilon \oplus \varepsilon) \oplus (\varepsilon \oplus \varepsilon)$  and so on. This means that  $\langle \varepsilon \rangle$  will be the following infinite Mealy coalgebra:



Fortunately, this drawback can easily be solved in the same way as for regular expressions. We just need to remove double occurrences of  $\varepsilon$  in sums of the form  $\dots \oplus \varepsilon \oplus \dots \oplus \varepsilon \oplus \dots$ . In what follows, we will consider expressions modulo the axioms for associativity, commutativity and idempotency (ACI). It is worth to remark that considering expressions modulo these axioms equates more expressions than what it is in fact needed to guarantee finiteness (yielding smaller automata).

We will use the following definitions in order to consider subcoalgebras generated modulo ACI. We start by defining the relation  $\equiv_{ACI} \subseteq \text{Exp}_{\mathcal{M}} \times \text{Exp}_{\mathcal{M}}$ , written infix, as the least equivalence relation containing the identities

$$\begin{aligned} (\text{Associativity}) \quad & \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv_{ACI} (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \\ (\text{Commutativity}) \quad & \varepsilon_1 \oplus \varepsilon_2 \equiv_{ACI} \varepsilon_2 \oplus \varepsilon_1 \\ (\text{Idempotency}) \quad & \varepsilon \oplus \varepsilon \equiv_{ACI} \varepsilon \end{aligned}$$

We denote by  $\text{Exp}_{\mathcal{M}} / \equiv_{ACI}$  the set of expressions modulo ACI. The equivalence relation  $\equiv_{ACI}$  induces the equivalence map  $[-]_{ACI} : \text{Exp}_{\mathcal{M}} \rightarrow \text{Exp}_{\mathcal{M}} / \equiv_{ACI}$ . Moreover, it is easy to prove that

$$\varepsilon_1 \equiv_{ACI} \varepsilon_2 \Rightarrow \varepsilon_1[a] = \varepsilon_2[a] \text{ and } (\varepsilon_1)_a \equiv_{ACI} (\varepsilon_2)_a$$

and, hence,  $\equiv_{ACI}$  is a bisimulation. This guarantees, by Theorem 2.2.7, that there is a unique function  $\bar{\delta}: \text{Exp}_{\mathcal{M}}/\equiv_{ACI} \rightarrow (\mathbb{B} \times \text{Exp}_{\mathcal{M}}/\equiv_{ACI})^A$  which turns  $[-]_{ACI}$  into a coalgebra homomorphism (and thus  $\varepsilon \sim [\varepsilon]_{ACI}$ ):

$$\begin{array}{ccc} \text{Exp}_{\mathcal{M}} & \xrightarrow{[-]_{ACI}} & \text{Exp}_{\mathcal{M}}/\equiv_{ACI} \\ \delta \downarrow & & \downarrow \bar{\delta} \\ (\mathbb{B} \times \text{Exp}_{\mathcal{M}})^A & \xrightarrow{(id \times [-]_{ACI})^A} & (\mathbb{B} \times \text{Exp}_{\mathcal{M}}/\equiv_{ACI})^A \end{array}$$

**4.2.8 THEOREM** (Kleene's theorem for Mealy machines (part II)). *For every  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ , there exists a finite Mealy coalgebra  $(S, \alpha)$  and  $s \in S$  such that  $s \sim \varepsilon$ .*

PROOF. Let  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ . We define  $(S, \alpha) = \langle [\varepsilon]_{ACI} \rangle$ , where  $\langle [\varepsilon]_{ACI} \rangle$  denotes the smallest subcoalgebra (equation (2.1)) generated by  $[\varepsilon]_{ACI}$ . We just need to prove that  $S$  is finite, since we already know that  $\varepsilon \sim [\varepsilon]_{ACI}$ . In what follows, we denote  $[-]_{ACI}$  by  $[-]$ . We prove that  $S$  is contained in a finite set, namely:

$$C = \{[\varepsilon_1 \oplus \dots \oplus \varepsilon_k] \mid \varepsilon_1, \dots, \varepsilon_k \in cl(\varepsilon) \text{ and } \varepsilon_1, \dots, \varepsilon_k \text{ all distinct}\}$$

In the above, we take the empty sum to be equal to  $\emptyset$  and the closure  $cl(\varepsilon)$  of  $\varepsilon$  is the set containing all subexpressions and unfoldings of  $\varepsilon$ , which is defined as the smallest set satisfying

$$\begin{aligned} cl(\emptyset) &= \{\emptyset\} \\ cl(\varepsilon_1 \oplus \varepsilon_2) &= \{\varepsilon_1 \oplus \varepsilon_2\} \cup cl(\varepsilon_1) \cup cl(\varepsilon_2) \\ cl(a(\varepsilon)) &= \{a(\varepsilon)\} \cup cl(\varepsilon) \\ cl(a \downarrow b) &= \{a \downarrow b\} \\ cl(\mu x. \varepsilon) &= \{\mu x. \varepsilon\} \cup cl(\varepsilon[\mu x. \varepsilon/x]) \end{aligned}$$

The set  $cl(\varepsilon)$  is finite, because the guardedness of the expressions guarantees that the number of different unfoldings of  $\mu$ -expressions is finite. The closure set has the properties  $\varepsilon \in cl(\varepsilon)$  and

$$\varepsilon' \in cl(\varepsilon) \Rightarrow cl(\varepsilon') \subseteq cl(\varepsilon) \quad (4.3)$$

In order to prove that  $S \subseteq C$ , we observe that  $(C, \bar{\delta})$  (here  $\bar{\delta}$  is actually the restriction of  $\bar{\delta}$  to  $C$ ) is a subcoalgebra of  $(\text{Exp}_{\mathcal{M}}/\equiv_{ACI}, \bar{\delta})$  with  $[\varepsilon] \in C$  (since  $\varepsilon \in cl(\varepsilon)$ ). Thus,  $S \subseteq C$ , since  $S$  is the state space of the smallest subcoalgebra generated by  $[\varepsilon]$ .

We only need to prove that for any  $[\varepsilon_1 \oplus \dots \oplus \varepsilon_k]$ , with  $\varepsilon_1, \dots, \varepsilon_k \in cl(\varepsilon)$  and  $\varepsilon_1, \dots, \varepsilon_k$  all distinct, and  $a \in A$

$$[\varepsilon_1 \oplus \dots \oplus \varepsilon_k]_a \in C$$

We observe that, for  $a \in A$ ,  $[(\varepsilon_i)_a] \in C$ , for any  $\varepsilon_i \in cl(\varepsilon)$  (easy proof by induction on  $\varepsilon_1$ , using equation (4.3)). Hence,

$$[\varepsilon_1 \oplus \dots \oplus \varepsilon_k]_a = [(\varepsilon_1)_a \oplus \dots \oplus (\varepsilon_k)_a] \in C$$

because, using the axioms (*Associativity*), (*Commutativity*) and (*Idempotency*), we can rewrite any sum  $u_1 \oplus \dots \oplus u_n$ , with all  $u_i \in cl(\varepsilon)$  as  $\varepsilon'_1 \oplus \dots \oplus \varepsilon'_k$ , with all  $\varepsilon'_1 \oplus \dots \oplus \varepsilon'_k \in cl(\varepsilon)$  distinct.  $\square$



Let us illustrate the construction of the theorem above.

**4.2.9 EXAMPLE.** Let  $A = \{a, b\}$  and  $B = \{0, 1\}$  (with  $\perp_B = 0$ ). In what follows, we will frequently denote equivalence classes  $[\varepsilon]$  by their representative  $\varepsilon$ , without any risk of confusion.

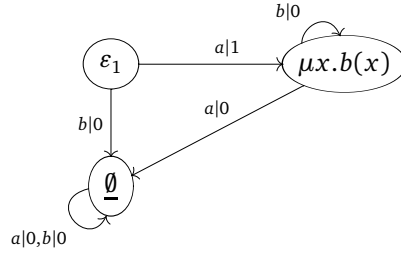
For the expression  $\varepsilon_1 = a(\mu x.b(x)) \oplus a\downarrow 1$  presented above, we have  $\langle [\varepsilon_1] \rangle = (S, \alpha)$ , where  $S = \{\varepsilon_1, \underline{\emptyset}, \mu x.b(x)\}$  and:

$$\begin{aligned} \varepsilon_1[a] &= 1 \text{ and } (\varepsilon_1)_a = \mu x.b(x) \\ \varepsilon_1[b] &= 0 \text{ and } (\varepsilon_1)_b = \underline{\emptyset} \end{aligned}$$

We now repeat the process for  $\mu x.b(x)$  and  $\underline{\emptyset}$ , which yields

$$\begin{aligned} (\mu x.b(x))[a] &= 0 \text{ and } (\mu x.b(x))_a = \underline{\emptyset} \\ (\mu x.b(x))[b] &= 0 \text{ and } (\mu x.b(x))_b = \mu x.b(x) \end{aligned}$$

Thus, the transition function  $\alpha$  is given by



Note that the Mealy machine above is not minimal: it is easy to see that the states  $\underline{\emptyset}$  and  $\mu x.b(x)$  are bisimilar and could therefore be identified.

The (special) output value  $\perp_B = 0$  allows us to define *underspecified* machines: if a given expression  $\varepsilon$  does not contain information about the output value for a given input  $a$ , then  $\varepsilon[a] = \perp_B$ . Being able to deal with underspecification is particularly important in the main context of application of Mealy machines: the design of digital circuits. If information about a particular input is not present, then a special mark should be output in order to signal the missing information. In [26], we also showed how to deal with *overspecification*, that is specifications with inconsistent information.

There, the set of output values had, in addition to  $\perp_B$ , also a special value  $\top_B$ .

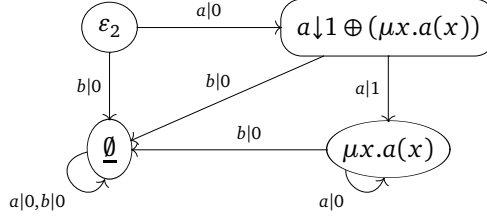
For another example, consider  $\varepsilon_2 = a(a\downarrow 1) \oplus (\mu x.a(x))$ . We have:

$$\begin{aligned} \varepsilon_2[a] &= 0 \text{ and } (\varepsilon_2)_a = a\downarrow 1 \oplus (\mu x.a(x)) \\ \varepsilon_2[b] &= 0 \text{ and } (\varepsilon_2)_b = \underline{\emptyset} \end{aligned}$$

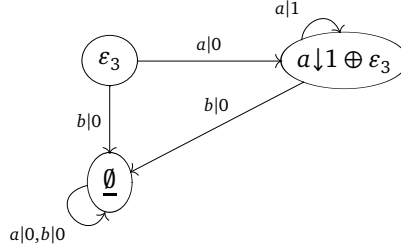
We now repeat the process for  $a\downarrow 1 \oplus (\mu x.a(x))$ , which yields

$$\begin{aligned} (\varepsilon_2)_a[a] &= 1 \text{ and } ((\varepsilon_2)_a)_a = \mu x.a(x) \\ (\varepsilon_2)_a[b] &= 0 \text{ and } ((\varepsilon_2)_a)_b = \underline{\emptyset} \end{aligned}$$

The complete Mealy machine is represented in the following diagram:



Now, take  $\varepsilon_3 = \mu x.a(a\downarrow 1) \oplus a(x)$ . Because  $a(a\downarrow 1)$  has no  $x$ 's one could be tempted to assume that the automaton for  $\varepsilon_3$  would be equivalent to the one for  $\varepsilon_2$ . However, that is not the case. The subcoalgebra generated by  $\varepsilon_3$  (modulo ACI) is quite different than the one presented above for  $\varepsilon_2$ :



In the state  $\varepsilon_2$  of the automaton above, for any input sequence, the output 1 will occur at most once, whereas in the state  $\varepsilon_3$  there will be  $n - 1$  occurrences of 1 for every input sequence starting with  $n$   $a$ 's.

As a last example, let  $\varepsilon_4 = \mu x.a(x \oplus (\mu y.a(y) \oplus a\downarrow 1))$ . We have:

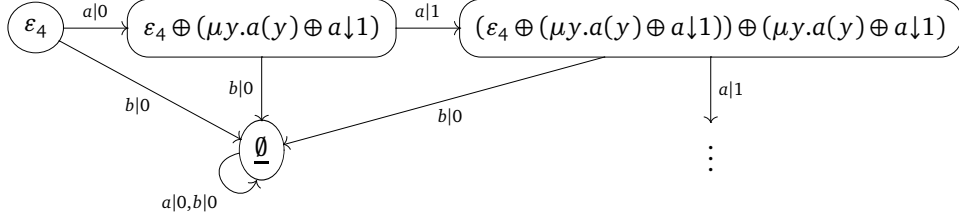
$$\begin{aligned} \varepsilon_4[a] &= 0 \text{ and } (\varepsilon_4)_a = \varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1) \\ (\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1))[a] &= 1 \\ \text{and } (\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1))_a &= (\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)) \oplus \mu y.a(y) \end{aligned}$$

and

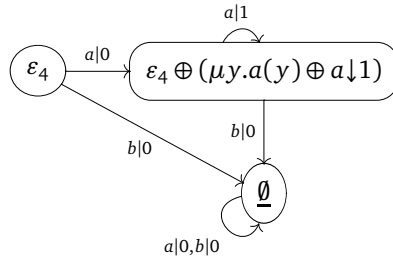
$$[\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)]_{ACI} = [(\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)) \oplus \mu y.a(y)]_{ACI}$$

Note that without ACI, the resulting state  $(\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)) \oplus \mu y.a(y)$  would be regarded as a new state, even though it is equivalent to (the already existing state)  $\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)$ . Moreover, the derivative of this state (for input  $a$ ) would yield again an equivalent but (syntactically) different state, namely  $((\varepsilon_4 \oplus (\mu y.a(y) \oplus a\downarrow 1)) \oplus \mu y.a(y)) \oplus \mu y.a(y)$ . This illustrates that without ACI the subcoalgebra generated from an expression is in general infinite. In this particular example the subcoalgebra

generated by  $\varepsilon_4$  without ACI (that is, in the coalgebra  $(\text{Exp}_{\mathcal{M}}, \delta)$ ) would be infinite:



whereas the one generated with ACI has only 3 states:



### 4.3 An algebra for Mealy machines

In this section, we introduce an equational axiomatization of the language  $\text{Exp}_{\mathcal{M}}$  for specifying Mealy machines, introduced in Section 4.2. We then prove it sound and complete with respect to bisimulation.

We define the relation  $\equiv \subseteq \text{Exp}_{\mathcal{M}} \times \text{Exp}_{\mathcal{M}}$ , written infix, as the least equivalence relation containing the following identities:

1.  $(\text{Exp}_{\mathcal{M}}, \oplus, \emptyset)$  is a join-semilattice.

$$\begin{array}{ll} \varepsilon \oplus \varepsilon \equiv \varepsilon & (\text{Idempotency}) \\ \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1 & (\text{Commutativity}) \\ \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 & (\text{Associativity}) \\ \emptyset \oplus \varepsilon \equiv \varepsilon & (\text{Empty}) \end{array}$$

2.  $\mu$  is the unique fixed point.

$$\begin{array}{ll} \gamma[\mu x.\gamma/x] \equiv \mu x.\gamma & (\text{FP}) \\ \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon & (\text{Unique}) \end{array}$$

3. The join-semilattice structure propagates through the expressions.

$$\begin{array}{lll} \underline{\emptyset} \equiv a \downarrow \perp_B & (\mathbf{B} - \underline{\emptyset}) & a \downarrow b_1 \oplus a \downarrow b_2 \equiv a \downarrow (b_1 \vee_B b_2) \quad (\mathbf{B} - \oplus) \\ a(\underline{\emptyset}) \equiv \underline{\emptyset} & (-^A - \underline{\emptyset}) & a(\varepsilon_1 \oplus \varepsilon_2) \equiv a(\varepsilon_1) \oplus a(\varepsilon_2) \quad (-^A - \oplus) \end{array}$$

4.  $\equiv$  is a congruence.

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x] \quad \text{if } x \text{ is free in } \varepsilon \quad (\text{Cong})$$

5.  $\alpha$ -equivalence

$$\mu x. \gamma \equiv \mu y. \gamma[y/x] \quad \text{if } y \text{ is not free in } \gamma \quad (\alpha\text{-equiv})$$

We denote by  $\text{Exp}_{\mathcal{M}}/\equiv$  the set of expressions modulo  $\equiv$ . The equivalence relation  $\equiv$  induces the equivalence map  $[-]: \text{Exp}_{\mathcal{M}} \rightarrow \text{Exp}_{\mathcal{M}}/\equiv$  given by  $[\varepsilon] = \{\varepsilon' \mid \varepsilon \equiv \varepsilon'\}$ .

**4.3.1 EXAMPLE.** Consider the following two expressions:

$$\varepsilon = \mu x. b(x) \oplus a(a(x) \oplus b(x)) \quad \varepsilon' = \mu y. a(x) \oplus b(x)$$

To get an intuition for their semantics note that they would be bisimilar to the states  $q_1$  and  $s_1$  below ( $A = \{a, b\}$  and  $B = \{0, 1\}$  with  $\perp_B = 0$ ).



We now show that they are provably equivalent:

$$\begin{array}{ll} \varepsilon \equiv \varepsilon' & \\ \Leftarrow b(\varepsilon') \oplus a(a(\varepsilon') \oplus b(\varepsilon')) \equiv \varepsilon' & (\text{axiom (Unique)}) \\ \Leftrightarrow b(\varepsilon') \oplus a(\varepsilon') \equiv \varepsilon' & (\text{axioms (FP) and (Cong)}) \\ \Leftrightarrow a(\varepsilon') \oplus b(\varepsilon') \equiv \varepsilon' & (\text{axioms (Commutativity)}) \\ \Leftrightarrow \varepsilon' \equiv \varepsilon' & (\text{axiom (FP)}) \end{array}$$

♠

Also as an example of applicability of the axioms above we prove the following theorem (the analogue of [29, Theorem 6.4], also presented as Theorem 3.1.14 in the previous chapter), which will be useful later in the proof of completeness.

**4.3.2 THEOREM** (Fundamental theorem for Mealy expressions). *For all  $\varepsilon \in \text{Exp}_{\mathcal{M}}$ ,*

$$\varepsilon \equiv \bigoplus_{a \in A} a(\varepsilon_a) \oplus a \downarrow \varepsilon[a] \quad (4.4)$$

PROOF. By induction on the complexity measure  $N(\varepsilon)$  (Definition 4.2.3).

For the base case, when  $N(\varepsilon) = 0$ , we need to consider  $\varepsilon \in \{\underline{\emptyset}, a \downarrow b, a(\varepsilon)\}$ .

For  $\varepsilon = \underline{\emptyset}$ , one has  $\varepsilon_a = \underline{\emptyset}$  and  $\varepsilon[a] = \perp_B$ , for any  $a \in A$ . Using the axioms  $(B - \underline{\emptyset})$  and  $(-^A - \underline{\emptyset})$  then yields  $\underline{\emptyset} \equiv \bigoplus_{a \in A} a(\underline{\emptyset}) \oplus a \downarrow \perp_B$ .

For  $\varepsilon = a \downarrow b$ ,  $\varepsilon_{a'} = \underline{\emptyset}$ , for any  $a' \in A$ ,  $\varepsilon[a'] = \perp_B \Leftrightarrow a' \neq a$  and  $\varepsilon[a] = b$ . Thus, again using  $(B - \underline{\emptyset})$  and  $(-^A - \underline{\emptyset})$ , one has  $a \downarrow b \equiv a \downarrow b \oplus a(\underline{\emptyset}) \oplus \left( \bigoplus_{a' \neq a} a'(\underline{\emptyset}) \oplus a' \downarrow \perp_B \right)$ .

For  $a(\varepsilon)$ , we have that  $a(\varepsilon)[a'] = \perp_B$ , for any  $a' \in A$ ,  $a(\varepsilon)_a = \varepsilon$  and, for any other  $a' \in A$  with  $a' \neq a$ ,  $a(\varepsilon)_{a'} = \underline{\emptyset}$ . The result then follows as for  $a \downarrow b$ .

For the inductive case, when  $N(\varepsilon) \geq k + 1$ , we need to consider  $\varepsilon \in \{\varepsilon_1 \oplus \varepsilon_2, \mu x.\varepsilon\}$ . For  $\varepsilon_1 \oplus \varepsilon_2$ , we first observe that  $(\varepsilon_1 \oplus \varepsilon_2)[a] = (\varepsilon_1)[a] \vee_B (\varepsilon_2)[a]$  and  $(\varepsilon_1 \oplus \varepsilon_2)_a = (\varepsilon_1)_a \oplus (\varepsilon_2)_a$ . Then we calculate:

$$\begin{aligned}
& \varepsilon_1 \oplus \varepsilon_2 \\
& \equiv \left( \bigoplus_{a \in A} a((\varepsilon_1)_a) \oplus a \downarrow \varepsilon_1[a] \right) \oplus \left( \bigoplus_{a \in A} a((\varepsilon_2)_a) \oplus a \downarrow \varepsilon_2[a] \right) \quad (\text{induction hypothesis}) \\
& \equiv \bigoplus_{a \in A} a((\varepsilon_1)_a \oplus (\varepsilon_2)_a) \oplus a \downarrow (\varepsilon_1[a] \vee_B \varepsilon_2[a]) \quad ((-^A - \oplus) \text{ and } (B - \oplus)) \\
& = \bigoplus_{a \in A} a((\varepsilon_1 \oplus \varepsilon_2)_a) \oplus a \downarrow ((\varepsilon_1 \oplus \varepsilon_2)[a])
\end{aligned}$$

Finally, if  $\varepsilon = \mu x.\gamma$ , we observe that  $\varepsilon[a] = (\gamma[\mu x.\gamma/x])[a]$  and  $\varepsilon_a = (\gamma[\mu x.\gamma/x])_a$  and thus:

$$\begin{aligned}
& \mu x.\gamma \\
& \equiv \gamma[\mu x.\gamma/x] \quad (\text{axiom (FP)}) \\
& \equiv \bigoplus_{a \in A} a((\gamma[\mu x.\gamma/x])_a) \oplus a \downarrow ((\gamma[\mu x.\gamma/x])[a]) \quad (\text{induction hypothesis}) \\
& = \bigoplus_{a \in A} a((\mu x.\gamma)_a) \oplus a \downarrow ((\mu x.\gamma)[a])
\end{aligned}$$

□

The goal is now to prove that the axiomatization presented above is sound and complete with respect to bisimilarity, that is, for all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{M}}$ :

$$\varepsilon_1 \equiv \varepsilon_2 \Leftrightarrow \varepsilon_1 \sim \varepsilon_2$$

To prove soundness ( $\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon_1 \sim \varepsilon_2$ ), we need the following lemma, which guarantees that the relation  $\equiv$  is a bisimulation.

**4.3.3 LEMMA.** *Let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{M}}$  and suppose that  $\varepsilon_1 \equiv \varepsilon_2$ . Then,*

$$\varepsilon_1[a] = \varepsilon_2[a] \text{ and } (\varepsilon_1)_a \equiv (\varepsilon_2)_a$$

PROOF. By induction on the length of derivations of  $\equiv$ .

The derivations of length 0 include all axioms apart from (*Unique*) and (*Cong*). We illustrate the proof for a couple of them.

$$\begin{aligned}
(\underline{\emptyset} \oplus \varepsilon)[a] &= \perp_{\mathbb{B}} \vee_{\mathbb{B}} (\varepsilon[a]) = \varepsilon[a] \quad \text{and} \quad (\underline{\emptyset} \oplus \varepsilon)_a = (\underline{\emptyset})_a \oplus \varepsilon_a = \underline{\emptyset} \oplus \varepsilon_a \equiv \varepsilon_a \\
(a(\underline{\emptyset}))[a'] &= \perp_{\mathbb{B}} = (\underline{\emptyset})[a'] \quad \text{and} \quad (a(\underline{\emptyset}))_{a'} = \underline{\emptyset} = (\underline{\emptyset})_a \\
(a \downarrow b_1 \oplus a \downarrow b_2)[a'] &= \begin{cases} b_1 \vee_{\mathbb{B}} b_2 & \text{if } a = a' \\ \perp_{\mathbb{B}} & \text{if } a \neq a' \end{cases} = (a \downarrow (b_1 \vee_{\mathbb{B}} b_2))[a'] \\
&\quad \text{and} \quad (a \downarrow b_1 \oplus a \downarrow b_2)_{a'} = \underline{\emptyset} \oplus \underline{\emptyset} \equiv \underline{\emptyset} = (a \downarrow (b_1 \vee_{\mathbb{B}} b_2))_{a'}
\end{aligned}$$

For the derivations of length greater than 0 we show the proof for (*Unique*), which uses (*Cong*). For the congruence axiom an auxiliary (easy) proof (by induction on the structure of  $\varepsilon$ ) would be needed. Suppose we have proved  $\gamma[\varepsilon/x] \equiv \varepsilon$ , which gives us as induction hypothesis  $(\gamma[\varepsilon/x])[a] = \varepsilon[a]$  and  $(\gamma[\varepsilon/x])_a \equiv \varepsilon_a$ . Now, we calculate:

$$\begin{aligned}
(\mu x. \gamma)[a] &= (\gamma[\mu x. \gamma/x])[a] \stackrel{(\text{Cong})}{\equiv} (\gamma[\varepsilon/x])[a] \stackrel{(\text{IH})}{\equiv} \varepsilon[a] \\
(\mu x. \gamma)_a &= (\gamma[\mu x. \gamma/x])_a \stackrel{(\text{Cong})}{\equiv} (\gamma[\varepsilon/x])_a \stackrel{(\text{IH})}{\equiv} \varepsilon_a
\end{aligned}$$

□

**4.3.4 THEOREM (Soundness).** *Let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{M}}$  and suppose that  $\varepsilon_1 \equiv \varepsilon_2$ . Then,  $\varepsilon_1 \sim \varepsilon_2$ .*

PROOF. Direct consequence of Lemma 4.3.3. □

To prove completeness, we need two extra things. First, we need to observe that Lemma 4.3.3 guarantees that the set  $\text{Exp}_{\mathcal{M}}/\equiv$  carries a coalgebra structure which makes  $[-]$  a coalgebra homomorphism, since  $\equiv$  is a bisimulation and, by Theorem 2.2.7, this guarantees the existence of a unique function  $\partial: \text{Exp}_{\mathcal{M}}/\equiv \rightarrow (\mathbb{B} \times \text{Exp}_{\mathcal{M}}/\equiv)^A$  which makes the following diagram commute.

$$\begin{array}{ccc}
\text{Exp}_{\mathcal{M}} & \xrightarrow{[-]} & \text{Exp}_{\mathcal{M}}/\equiv & \quad \quad \quad [\varepsilon][a] = \varepsilon[a] \text{ and } [\varepsilon]_a = [\varepsilon_a] \\
\delta \downarrow & & \downarrow \partial & \\
(\mathbb{B} \times \text{Exp}_{\mathcal{M}})^A & \xrightarrow{(id \times [-])^A} & (\mathbb{B} \times \text{Exp}_{\mathcal{M}}/\equiv)^A & 
\end{array}$$

Secondly, we will use (point 3. of) the following auxiliary lemma.

**4.3.5 LEMMA.**

1. *The map  $\partial: \text{Exp}_{\mathcal{M}}/\equiv \rightarrow (\mathbb{B} \times \text{Exp}_{\mathcal{M}}/\equiv)^A$  is an isomorphism.*
2. *There is a unique homomorphism  $h: (S, \alpha) \rightarrow (\text{Exp}_{\mathcal{M}}/\equiv, \partial)$ .*
3. *The unique map into the final coalgebra  $[[ - ]]_{\text{Exp}_{\mathcal{M}}/\equiv}: \text{Exp}_{\mathcal{M}}/\equiv \rightarrow \Phi$  is injective.*

PROOF. For point 1., define  $\partial^{-1}: (\mathbb{B} \times \text{Exp}_{\mathcal{M}/\equiv})^A \rightarrow \text{Exp}_{\mathcal{M}/\equiv}$ , for  $f: A \rightarrow (\mathbb{B} \times \text{Exp}_{\mathcal{M}/\equiv})$ , by

$$\partial^{-1}(f) = \left[ \bigoplus_{a \in A} f[a] \oplus a(\mathbf{r}(f_a)) \right] \text{ where } \mathbf{r}([\varepsilon]) = \varepsilon$$

Note that  $\partial^{-1}$  will give the same result, independently of the representative of  $f_a$ : take two different elements  $\varepsilon_1$  and  $\varepsilon_2$  of the equivalence class  $f_a$ ;  $\varepsilon_1$  and  $\varepsilon_2$  satisfy  $\varepsilon_1 \equiv \varepsilon_2$  and thus  $\bigoplus_{a \in A} f[a] \oplus a(\varepsilon_1) \equiv \bigoplus_{a \in A} f[a] \oplus a(\varepsilon_2)$ .

It is easy to check that  $(\partial^{-1} \circ \partial)([\varepsilon]) = [\varepsilon]$ :

$$\begin{aligned} \partial^{-1}(\partial([\varepsilon])) &= \partial^{-1}(\lambda a. \langle \varepsilon[a], [\varepsilon_a] \rangle) \quad ([-] \text{ is a coalgebra homomorphism}) \\ &= \left[ \bigoplus_{a \in A} \varepsilon[a] \oplus a(\varepsilon_a) \right] \quad (\text{definition of } \partial^{-1}) \\ &= [\varepsilon] \quad (\text{Fundamental theorem}) \end{aligned}$$

For the converse, we need to prove, for  $f: A \rightarrow (\mathbb{B} \times \text{Exp}_{\mathcal{M}/\equiv})$ , that  $(\partial \circ \partial^{-1})(f) = f$ :

$$\begin{aligned} \partial(\partial^{-1}(f))(a) &= \partial\left(\left[\bigoplus_{a \in A} f[a] \oplus a(\mathbf{r}(f_a))\right]\right) \quad (\text{def. } \partial^{-1}) \\ &= \langle f[a], [\mathbf{r}(f_a)] \rangle \quad ([-] \text{ is a coalgebra homomorphism}) \\ &= \langle f[a], [f_a] \rangle = f(a) \end{aligned}$$

For point 2., for the existence we define  $h: (S, \alpha) \rightarrow (\text{Exp}_{\mathcal{M}/\equiv}, \partial)$  by  $h(s) = [\varepsilon_s]$ , where  $\varepsilon_s$  is the expression constructed for every  $s \in S$  in the first part of Kleene's theorem for Mealy machines (Theorem 4.2.7). Recall that  $\varepsilon_s$  satisfies  $\varepsilon_s[a] = s[a]$  and  $(\varepsilon_s)_a = \varepsilon_{s_a}$ . We prove that  $h$  is a homomorphism:

$$\begin{array}{ll} h(s)[a] & h(s)_a \\ = ([\varepsilon_s])[a] & = ([\varepsilon_s])_a \\ = \varepsilon_s[a] \quad ([-] \text{ is a homomorphism}) & = [(\varepsilon_s)_a] \quad ([-] \text{ is a homomorphism}) \\ = s[a] \quad (\varepsilon_s[a] = s[a]) & = [\varepsilon_{s_a}] \quad ((\varepsilon_s)_a = \varepsilon_{s_a}) \end{array}$$

To prove the uniqueness, suppose we have another homomorphism

$$f: (S, \alpha) \rightarrow (\text{Exp}_{\mathcal{M}/\equiv}, \partial)$$

We shall prove that  $f(s) = h(s)$ . Let, for any  $s \in S$ ,  $f_s$  denote any representative of  $f(s)$  (that is,  $f(s) = [f_s]$ ). First, we observe that because  $f$  is a homomorphism the following holds for every  $s \in S$ :

$$f_s \equiv (\partial^{-1} \circ (id \times f)^A \circ \alpha)(s) \Leftrightarrow f_s \equiv \bigoplus_{a \in A} s[a] \oplus a(f_{s_a}) \quad (4.5)$$

where  $\partial^{-1}$  was defined above, in the proof of item 1..

We now prove that  $f_{s_i} \equiv \varepsilon_{s_i}$ , which yields the intended result  $f = h$  (since  $h(s) = [\varepsilon_s]$ ). We show the case when  $S = \{s_1, \dots, s_n\}$  for  $n = 3$ ; the general case is completely analogous but notationally heavier. The key point of this proof is the use of the axiom (*Unique*) stating the uniqueness of fixed points.

First, we prove that  $f_{s_1} \equiv A_1[f_{s_2}/x_2][f_{s_3}/x_3]$ .

$$\begin{aligned}
f_{s_1} &\equiv \bigoplus_{a \in A} s_1[a] \oplus a(x_{s_a})[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (4.5))} \\
\Leftrightarrow f_{s_1} &\equiv \bigoplus_{a \in A} s_1[a] \oplus a(x_{s_a})[f_{s_2}/x_2][f_{s_3}/x_3][f_{s_1}/x_1] && \text{(all } f_{s_i} \text{ are closed)} \\
\Rightarrow f_{s_1} &\equiv \mu x_1. \bigoplus_{a \in A} s_1[a] \oplus a(x_{s_a})[f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by uniqueness)} \\
\Leftrightarrow f_{s_1} &\equiv A_1[f_{s_2}/x_2][f_{s_3}/x_3] && \text{(def. of } A_1)
\end{aligned}$$

Now, using what we have computed for  $f_{s_1}$  we prove that  $f_{s_2} \equiv A_2^1[f_{s_3}/x_3]$ .

$$\begin{aligned}
f_{s_2} &\equiv \bigoplus_{a \in A} s_2[a] \oplus a(x_{s_a})[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (4.5))} \\
f_{s_2} &\equiv \bigoplus_{a \in A} s_2[a] \oplus a(x_{s_a})[A_1/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(expr. for } f_{s_1} \text{ and (4.2))} \\
\Leftrightarrow f_{s_2} &\equiv \bigoplus_{a \in A} s_2[a] \oplus a(x_{s_a})[A_1/x_1][f_{s_3}/x_3][f_{s_2}/x_2] && \text{(all } f_{s_i} \text{ are closed)} \\
\Rightarrow f_{s_2} &\equiv \mu x_2. \bigoplus_{a \in A} s_2[a] \oplus a(x_{s_a})[A_1/x_1][f_{s_3}/x_3] && \text{(by uniqueness)} \\
\Leftrightarrow f_{s_2} &\equiv A_2^1[f_{s_3}/x_3] && \text{(def. of } A_2^1)
\end{aligned}$$

At this point we substitute  $f_{s_2}$  in the expression for  $f_{s_1}$  by  $A_2^1[f_{s_3}/x_3]$  which yields:

$$f_{s_1} \equiv A_1[A_2^1[f_{s_3}/x_3]/x_2][f_{s_3}/x_3] \equiv A_1[A_2^1/x_2][f_{s_3}/x_3]$$

Finally, we prove that  $f_{s_3} \equiv A_3^2$ :

$$\begin{aligned}
f_{s_3} &\equiv \bigoplus_{a \in A} s_3[a] \oplus a(x_{s_a})[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (4.5))} \\
\Leftrightarrow f_{s_3} &\equiv \bigoplus_{a \in A} s_3[a] \oplus a(x_{s_a})[A_1/x_1][A_2^1/x_2][f_{s_3}/x_3] && \text{(expr. for } f_{s_i} \text{ and (4.2))} \\
\Rightarrow f_{s_3} &\equiv \mu x_3. \bigoplus_{a \in A} s_3[a] \oplus a(x_{s_a})[A_1/x_1][A_2^1/x_2] && \text{(by uniqueness)} \\
\Leftrightarrow f_{s_3} &\equiv A_3^2 && \text{(def. of } A_3^2)
\end{aligned}$$

Thus, we have:

$$f_{s_1} \equiv A_1[A_2^1/x_2][A_3^2/x_3] \quad f_{s_2} \equiv A_2^1[A_3^2/x_3] \quad f_{s_3} \equiv A_3^2$$

Note that  $A_2^1[A_3^2/x_3] \equiv A_2^1\{A_3^2/x_3\}$  since  $x_2$  is not free in  $A_3^2$ . Similarly, it also holds  $[A_2^1/x_2][A_3^2/x_3] \equiv \{A_2^1/x_2\}\{A_3^2/x_3\}$ . Thus  $f_{s_i} \equiv \varepsilon_{s_i}$ , for all  $i = 1, 2, 3$ .

For point 3., take the factorization of  $[[ - ]]_{\text{Exp}_{\mathcal{M}/\equiv}}$  into a surjective followed by an injective map.

$$[[ - ]]_{\text{Exp}_{\mathcal{M}/\equiv}} = ( \text{Exp}_{\mathcal{M}/\equiv} \xrightarrow{e} I \xrightarrow{m} \Psi )$$

By Theorem 2.2.8 we have that  $I$  carries a coalgebra structure  $\bar{\varphi}$  which makes  $e$  and  $m$  coalgebra homomorphisms. We can now show that the coalgebra  $(I, \bar{\varphi})$  is isomorphic to  $(\text{Exp}_{\mathcal{M}/\equiv}, \partial)$ , which will yield the intended result that  $[[ - ]]_{\text{Exp}_{\mathcal{M}/\equiv}}$  is injective (since it is the composition of an isomorphism with a monomorphism).



To show the aforementioned isomorphism we observe that  $(I, \overline{\varphi})$  and  $(\text{Exp}_{\mathcal{M}/\equiv}, \partial)$  are final among the locally finite Mealy coalgebras: Mealy machines  $(S, \alpha)$  for which the subcoalgebra  $\langle s \rangle$  generated by any  $s \in S$  is finite. The fact that the coalgebra  $(\text{Exp}_{\mathcal{M}/\equiv}, \partial)$  is locally finite is a direct consequence of the second part of Kleene's theorem (Theorem 4.2.8), where we proved that the subcoalgebra  $\langle \varepsilon \rangle$ , when taken modulo  $ACI$ , is finite. Point 2. above now guarantees that  $(\text{Exp}_{\mathcal{M}/\equiv}, \partial)$  is final (among the locally finite<sup>1</sup>). The Mealy machine  $(I, \overline{\varphi})$  is locally finite because it is the image of a locally finite one. Moreover, it is final: the existence of a homomorphism is given by Kleene's theorem and  $e: \text{Exp}_{\mathcal{M}/\equiv} \rightarrow I$ ; the uniqueness follows by finality of  $\Phi$ . Indeed, take any two homomorphisms  $f, g: (S, \alpha) \rightarrow (I, \overline{\varphi})$ ; by finality we have that  $m \circ f = m \circ g$  and, since  $m$  is a monomorphism,  $f = g$ .

Final objects are unique up to isomorphism and hence  $e: \text{Exp}_{\mathcal{M}/\equiv} \rightarrow I$  is an isomorphism, which implies that  $[[ - ]]_{\text{Exp}_{\mathcal{M}/\equiv}} = m \circ e$  is injective.  $\square$

We have all we need to prove that the axiomatization is complete with respect to bisimulation.

**4.3.6 THEOREM (Completeness).** *For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{M}}$ , if  $\varepsilon_1 \sim \varepsilon_2$  then  $\varepsilon_1 \equiv \varepsilon_2$ .*

PROOF. Let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{M}}$  and suppose that  $\varepsilon_1 \sim \varepsilon_2$ , that is,  $[[ \varepsilon_1 ]] = [[ \varepsilon_2 ]]$ . Since  $[-]$  is a homomorphism, we have  $[[ [\varepsilon_1] ]]_{\text{Exp}_{\mathcal{M}/\equiv}} = [[ [\varepsilon_2] ]]_{\text{Exp}_{\mathcal{M}/\equiv}}$ . Because  $[[ - ]]_{\text{Exp}_{\mathcal{M}/\equiv}}$  is injective, the latter equality implies that  $[\varepsilon_1] = [\varepsilon_2]$  and hence  $\varepsilon_1 \equiv \varepsilon_2$ .  $\square$

## 4.4 Discussion

In this chapter, we have explored the extension of Kleene's results to another particular type of automata, Mealy machines. We showed that, in the spirit of regular expressions, it is possible to define a language which denotes precisely the behaviours of finite Mealy machines. Moreover, analogous to Kleene's theorem, we proved that each expression in the language is equivalent to a finite Mealy machine and, conversely, each state of a finite Mealy machine is equivalent to an expression in the language. Finally, we provided an equational system, sound and complete with respect to bisimilarity, which allows for syntactic reasoning on the expressions in the same way that Kleene algebras allow for Kleene's regular expressions.

Both Mealy machines and deterministic automata are instances of  $\mathcal{F}$ -coalgebras, for a functor  $\mathcal{F}: \mathbf{Set} \rightarrow \mathbf{Set}$ . An  $\mathcal{F}$ -coalgebra is a pair  $(S, f)$  where  $S$  is a set of states and  $f: S \rightarrow \mathcal{F}(S)$  is the transition function, which determines the dynamics of the system. Mealy machines are coalgebras for the functor  $(\mathbf{B} \times \text{Id})^A$ , whereas deterministic automata are coalgebras for the functor  $2 \times \text{Id}^A$ . Many systems can be obtained by varying the functor under consideration. We will show in the next two chapters how all the results presented in this chapter can be extended to a large class of functors.

---

<sup>1</sup>Note that the proof of item 2. above, as well as the one of Kleene's theorem, was for finite Mealy machines, but it is easily adapted for locally finite ones: everywhere where we consider  $S$  we must then consider  $\langle s \rangle$ , for a given  $s \in S$ . The use of finiteness was just for convenience.

In his seminal paper [64], S. Kleene introduced an algebraic description of regular languages: regular expressions. This was the precursor of many research, including the one presented in this chapter. The connection between Kleene's work (and the work of some researchers who followed up on his work) and the research presented in this thesis has been discussed in the introduction.

The connection between regular expressions (and deterministic automata) and coalgebras was first explored in [95]. There deterministic automata, the set of formal languages and regular expressions are all presented as coalgebras of the functor  $2 \times \text{Id}^A$  (where  $A$  is the alphabet, and  $2$  is the two element set). It is then shown that the standard semantics of language acceptance of automata and the assignment of languages to regular expressions both arise as the unique homomorphism into the final coalgebra of formal languages. The coalgebra structure on the set of regular expressions is determined by their so-called *Brzozowski* derivatives [29]. In the present chapter, the definition of a coalgebra structure on the set of expressions is very much inspired by both [29, 95].

The second part of Kleene's theorem provides a synthesis algorithm to produce a Mealy machine from an expression. Automata synthesis is a popular and very active research area [37, 54, 74, 90, 112]. Most of the work done on synthesis has as main goal to find a proper and sufficiently expressive type of automata to encode a specific type of logic (such as LTL [112] or  $\mu$ -calculus [74]). Technically, the synthesis from a  $\mu$ -calculus formula  $\varphi$  consists in translating  $\varphi$  into an alternating automaton  $\mathcal{A}_\varphi$ , reducing  $\mathcal{A}_\varphi$  into a non-deterministic automaton which is then checked for non-emptiness [74]. The same process has been recently generalized to  $\mathcal{F}$ -coalgebras in [75]. In this paper, we use a different approach. We construct a deterministic Mealy machine for a formula directly, by considering the formula as a state of the automaton containing enough information about its successors.

The logic most similar to our language of expressions is the one presented in [37]. There a logic for formal specification of hardware protocols is presented, and an algorithm for the synthesis of a Mealy machine is given. Their logic corresponds to the conjunctive fragment of LTL. Their synthesis process is standard: first a non-deterministic Büchi automaton is synthesized, secondly a powerset construction is used to make the automaton deterministic and, finally, the propositions on the states are used to determine the inputs and outputs for each state of the Mealy machine. Because of our coalgebraic approach, the set of expressions comes with an equational system that is sound and complete with respect to bisimilarity. Further, our synthesis process remains within standard Mealy machines and the behaviour of the synthesized automata is exactly characterized by the original expression.

Apart from [54, 99], where synthesis for a special case of 2-adic arithmetic is treated, we did not find any other work on the direct synthesis of deterministic Mealy machines. From these papers we inherit the basic coalgebraic approach, that we use here to derive our expressive specification language for Mealy machines.

The modal fragment of our language (that is, the set of expressions  $\varepsilon \in \text{Exp}_{\mathcal{M}}$  without the  $\mu$  operator) is a special case of the coalgebraic logic obtained by a Stone-type duality [24, 25].

In the previous chapter, we presented a language to describe the behavior of Mealy machines and a sound and complete axiomatization thereof. The defined language and axiomatization can be seen as the analogue of classical regular expressions [64] and Kleene algebra [66], for deterministic finite automata (DFA), or the process algebra and axiomatization for labeled transition systems (LTS) [84].

We now extend the previous approach and devise a framework wherein languages and axiomatizations can be uniformly derived for a large class of systems, including DFA, LTS and Mealy machines. The key point of our framework is to model systems as coalgebras.

Coalgebras provide a general framework for the study of dynamical systems such as DFA, Mealy machines and LTS. For a functor  $\mathcal{G}: \mathbf{Set} \rightarrow \mathbf{Set}$ , a  $\mathcal{G}$ -coalgebra or  $\mathcal{G}$ -system is a pair  $(S, g)$ , consisting of a set  $S$  of states and a function  $g: S \rightarrow \mathcal{G}(S)$  defining the “transitions” of the states. We call the functor  $\mathcal{G}$  the *type* of the system. For instance, DFA can be modeled as coalgebras of the functor  $\mathcal{G}(S) = 2 \times S^A$ , Mealy machines are obtained by taking  $\mathcal{G}(S) = (B \times S)^A$  and image-finite LTS are coalgebras for the functor  $\mathcal{G}(S) = (\mathcal{P}_\omega(S))^A$ , where  $\mathcal{P}_\omega$  is finite powerset.

Under mild conditions, functors  $\mathcal{G}$  have a *final coalgebra* (as defined in Chapter 2) into which every  $\mathcal{G}$ -coalgebra can be mapped via a unique so-called  $\mathcal{G}$ -homomorphism. The final coalgebra can be viewed as the universe of all possible  $\mathcal{G}$ -behaviors: the unique homomorphism into the final coalgebra maps every state of a coalgebra to a canonical representative of its behavior. This provides a general notion of behavioral equivalence: two states are equivalent if and only if they are mapped to the same element of the final coalgebra. Instantiating the notion of final coalgebra for the aforementioned examples, the result is as expected: for DFA the final coalgebra is the set  $2^A$  of all languages over  $A$ ; for Mealy machines it is the set of causal functions  $f: A^\omega \rightarrow B^\omega$ ; and for LTS it is the set of finitely branching trees with arcs labeled by  $a \in A$  modulo bisimilarity. The notion of equivalence also specializes to the familiar notions: for DFA, two states are equivalent when they accept the same language; for Mealy machines, if they realize (or compute) the same causal function; and for LTS if they are bisimilar.

It is the main aim of this chapter to show how the type of a system, given by the functor  $\mathcal{G}$ , is not only enough to determine a notion of behavior and behavioral equivalence, but also allows for a uniform derivation of both a set of expressions describing behavior and a corresponding axiomatization. The theory of universal coalgebra [96] provides a standard equivalence and a universal domain of behaviors, uniquely based on the functor  $\mathcal{G}$ . The main contributions of the chapter are (1) the definition of a set of expressions  $\text{Exp}_{\mathcal{G}}$  describing  $\mathcal{G}$ -behaviors, (2) the proof of the correspondence between behaviors described by  $\varepsilon \in \text{Exp}_{\mathcal{G}}$  and locally finite  $\mathcal{G}$ -coalgebras (this is the analogue of Kleene's theorem), and (3) a corresponding sound and complete axiomatization, with respect to behavioral equivalence, of  $\text{Exp}_{\mathcal{G}}$  (this is the analogue of Kleene algebra). All these results are solely based on the type of the system, given by the functor  $\mathcal{G}$ .

**Organization of the chapter.** In Section 5.1 we introduce the class of non-deterministic functors and coalgebras. In Section 5.2 we associate with each non-deterministic functor  $\mathcal{G}$  a generalized language  $\text{Exp}_{\mathcal{G}}$  of regular expressions and we present an analogue of Kleene's theorem, which makes precise the connection between  $\text{Exp}_{\mathcal{G}}$  and  $\mathcal{G}$ -coalgebras. A sound and complete axiomatization of  $\text{Exp}_{\mathcal{G}}$  is presented in Section 5.3. Section 5.4 contains two more examples of application of the framework and Section 5.5 shows a language and axiomatization for the class of polynomial and finitary coalgebras. Section 5.6 presents concluding remarks, directions for future work and discusses related work.

## 5.1 Non-deterministic coalgebras

A non-deterministic coalgebra is a pair  $(S, f : S \rightarrow \mathcal{G}(S))$ , where  $S$  is a set of states and  $\mathcal{G}$  is a non-deterministic functor. Non-deterministic functors are functors  $\mathcal{G} : \mathbf{Set} \rightarrow \mathbf{Set}$ , built inductively from the identity and constants, using  $\times$ ,  $\oplus$ ,  $(-)^A$  and  $\mathcal{P}_{\omega}$ .

**5.1.1 DEFINITION.** The class *NDF* of non-deterministic functors on  $\mathbf{Set}$  is inductively defined by putting:

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

where  $B$  is a (non-empty) finite join-semilattice and  $A$  is a finite set. ♣

Since we only consider finite exponents  $A = \{a_1, \dots, a_n\}$ , the functor  $(-)^A$  is not really needed, since it is subsumed by a product with  $n$  components. However, to simplify the presentation, we decided to include it.

Next, we show the explicit definition of the functors above on a set  $X$  and on a morphism  $f : X \rightarrow Y$  (note that  $\mathcal{G}(f) : \mathcal{G}(X) \rightarrow \mathcal{G}(Y)$ ).

$$\begin{array}{lll} \text{Id}(X) = X & B(X) = B & (\mathcal{G}_1 \oplus \mathcal{G}_2)(X) = \mathcal{G}_1(X) \oplus \mathcal{G}_2(X) \\ \text{Id}(f) = f & B(f) = \text{id}_B & (\mathcal{G}_1 \oplus \mathcal{G}_2)(f) = \mathcal{G}_1(f) \oplus \mathcal{G}_2(f) \\ (\mathcal{G}^A)(X) = \mathcal{G}(X)^A & (\mathcal{P}_{\omega} \mathcal{G})(X) = \mathcal{P}_{\omega}(\mathcal{G}(X)) & (\mathcal{G}_1 \times \mathcal{G}_2)(X) = \mathcal{G}_1(X) \times \mathcal{G}_2(X) \\ (\mathcal{G}^A)(f) = \mathcal{G}(f)^A & (\mathcal{P}_{\omega} \mathcal{G})(f) = \mathcal{P}_{\omega}(\mathcal{G}(f)) & (\mathcal{G}_1 \times \mathcal{G}_2)(f) = \mathcal{G}_1(f) \times \mathcal{G}_2(f) \end{array}$$

Typical examples of non-deterministic functors include  $\mathcal{M} = (\mathbb{B} \times \text{Id})^A$ ,  $\mathcal{D} = 2 \times \text{Id}^A$ ,  $\mathcal{Q} = (1 \oplus \text{Id})^A$  and  $\mathcal{N} = 2 \times (\mathcal{P}_\omega \text{Id})^A$ , where  $2 = \{0, 1\}$  is a two-element join semilattice with 0 as bottom element ( $1 \vee 0 = 1$ ) and  $1 = \{*\}$  is a one element join-semilattice. These functors represent, respectively, the type of Mealy, deterministic, partial deterministic and non-deterministic automata. In this chapter, we will use the last three as running examples. In Chapter 4, we have studied in detail regular expressions for Mealy automata. Similarly to what happened there, we impose a join semilattice structure on the constant functor. The product, exponentiation and powerset functors preserve the join-semilattice structure and thus do not need to be changed. This is not the case for the classical coproduct and thus we use  $\oplus$  instead (the operation  $\oplus$  was defined in the preliminaries of this thesis), which also guarantees that the join semilattice structure is preserved.

Next, we give the definition of the ingredient relation, which relates a non-deterministic functor  $\mathcal{G}$  with its *ingredients*, *i.e.* the functors used in its inductive construction. We shall use this relation later for typing our expressions.

**5.1.2 DEFINITION.** Let  $\triangleleft \subseteq \text{NDF} \times \text{NDF}$  be the least reflexive and transitive relation on non-deterministic functors such that

$$\mathcal{G}_1 \triangleleft \mathcal{G}_1 \times \mathcal{G}_2, \quad \mathcal{G}_2 \triangleleft \mathcal{G}_1 \times \mathcal{G}_2, \quad \mathcal{G}_1 \triangleleft \mathcal{G}_1 \oplus \mathcal{G}_2, \quad \mathcal{G}_2 \triangleleft \mathcal{G}_1 \oplus \mathcal{G}_2, \quad \mathcal{G} \triangleleft \mathcal{G}^A, \quad \mathcal{G} \triangleleft \mathcal{P}_\omega \mathcal{G}$$



Here and throughout this document we use  $\mathcal{F} \triangleleft \mathcal{G}$  as a shorthand for  $\langle \mathcal{F}, \mathcal{G} \rangle \in \triangleleft$ . If  $\mathcal{F} \triangleleft \mathcal{G}$ , then  $\mathcal{F}$  is said to be an *ingredient* of  $\mathcal{G}$ . For example,  $2$ ,  $\text{Id}$ ,  $\text{Id}^A$  and  $\mathcal{D}$  itself are all the ingredients of the deterministic automata functor  $\mathcal{D} = 2 \times \text{Id}^A$ .

## 5.2 A language of expressions for non-deterministic coalgebras

In this section, we generalize the classical notion of regular expressions to non-deterministic coalgebras. We start by introducing an untyped language of expressions and then we single out the well-typed ones via an appropriate typing system, thereby associating expressions to non-deterministic functors. The language of expressions presented in Chapter 4 for Mealy machines can be recovered as a special case of this language by taking the corresponding functor.

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

$$\varepsilon ::= \emptyset \mid x \mid \varepsilon \oplus \varepsilon \mid \mu x. \gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid \{\varepsilon\}$$

where  $\gamma$  is a *guarded expression* given by:

$$\gamma ::= \emptyset \mid \gamma \oplus \gamma \mid \mu x. \gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid \{\varepsilon\}$$

The only difference between the BNF of  $\gamma$  and  $\varepsilon$  is the occurrence of  $x$ .



In the expression  $\mu x.\gamma$ ,  $\mu$  is a binder for all the free occurrences of  $x$  in  $\gamma$ . Variables that are not bound are free. A *closed expression* is an expression without free occurrences of fixed point variables  $x$ . We denote the set of closed expressions by  $\text{Exp}^c$ .

Intuitively, expressions denote elements of the final coalgebra. The expressions  $\emptyset$ ,  $\varepsilon_1 \oplus \varepsilon_2$  and  $\mu x.\varepsilon$  will play a similar role to, respectively, the empty language, the union of languages and the Kleene star in classical regular expressions for deterministic automata. The expressions  $l(\varepsilon)$  and  $r(\varepsilon)$  refer to the left and right hand-side of products. Similarly,  $l[\varepsilon]$  and  $r[\varepsilon]$  refer to the left and right hand-side of sums. The expressions  $a(\varepsilon)$  and  $\{\varepsilon\}$  denote function application and a singleton set, respectively. We shall soon illustrate, by means of examples, the role of these expressions.

Our language does not have any operator denoting intersection or complement (it only includes the sum operator  $\oplus$ ). This is a natural restriction, very much in the spirit of Kleene's regular expressions for deterministic finite automata. We will prove that this simple language is expressive enough to denote exactly all locally finite coalgebras.

Next, we present a typing assignment system for associating expressions to non-deterministic functors. This will allow us to associate with each functor  $\mathcal{G}$  the expressions  $\varepsilon \in \text{Exp}^c$  that are valid specifications of  $\mathcal{G}$ -coalgebras. The typing proceeds following the structure of the expressions and the ingredients of the functors.

**5.2.2 DEFINITION (Type system).** We define a typing relation  $\vdash \subseteq \text{Exp} \times \text{NDF} \times \text{NDF}$  that will associate an expression  $\varepsilon$  with two non-deterministic functors  $\mathcal{F}$  and  $\mathcal{G}$ , which are related by the ingredient relation ( $\mathcal{F}$  is an ingredient of  $\mathcal{G}$ ). We shall write  $\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{G}$  for  $\langle \varepsilon, \mathcal{F}, \mathcal{G} \rangle \in \vdash$ . The rules that define  $\vdash$  are the following:

$$\begin{array}{c}
\frac{}{\vdash \emptyset : \mathcal{F} \triangleleft \mathcal{G}} \qquad \frac{}{\vdash b : \mathbf{B} \triangleleft \mathcal{G}} \qquad \frac{}{\vdash x : \mathcal{G} \triangleleft \mathcal{G}} \qquad \frac{\vdash \varepsilon : \mathcal{G} \triangleleft \mathcal{G}}{\vdash \mu x.\varepsilon : \mathcal{G} \triangleleft \mathcal{G}} \\
\frac{\vdash \varepsilon_1 : \mathcal{F} \triangleleft \mathcal{G} \quad \vdash \varepsilon_2 : \mathcal{F} \triangleleft \mathcal{G}}{\vdash \varepsilon_1 \oplus \varepsilon_2 : \mathcal{F} \triangleleft \mathcal{G}} \qquad \frac{}{\vdash \varepsilon : \mathcal{G} \triangleleft \mathcal{G}} \qquad \frac{}{\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{G}} \qquad \frac{\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{G}}{\vdash a(\varepsilon) : \mathcal{F}^A \triangleleft \mathcal{G}} \\
\frac{}{\vdash \varepsilon : \mathcal{F}_1 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash \varepsilon : \mathcal{F}_2 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash \varepsilon : \mathcal{F}_1 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash \varepsilon : \mathcal{F}_2 \triangleleft \mathcal{G}} \\
\frac{}{\vdash l(\varepsilon) : \mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash r(\varepsilon) : \mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash l[\varepsilon] : \mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}} \qquad \frac{}{\vdash r[\varepsilon] : \mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}
\end{array}$$

♣

Intuitively,  $\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{G}$  (for a closed expression  $\varepsilon$ ) means that  $\varepsilon$  denotes an element of  $\mathcal{F}(\Omega_{\mathcal{G}})$ , where  $\Omega_{\mathcal{G}}$  is the final coalgebra of  $\mathcal{G}$ . As expected, there is a rule for each expression construct. The extra rule involving  $\text{Id} \triangleleft \mathcal{G}$  reflects the isomorphism between the final coalgebra  $\Omega_{\mathcal{G}}$  and  $\mathcal{G}(\Omega_{\mathcal{G}})$  (Lambek's lemma, cf. [96]). Only fixed points at the outermost level of the functor are allowed. This does not mean however that we disallow nested fixed points. For instance,  $\mu x.a(x \oplus \mu y.a(y))$  would be a well-typed expression for the functor  $\mathcal{D}$  of deterministic automata, as it will become clear below, when we will present more examples of well-typed and non-well-typed

expressions. The presented type system is decidable (expressions are of finite length and the system is inductive on the structure of  $\varepsilon \in \text{Exp}$ ).

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

**5.2.3 DEFINITION** ( $\mathcal{G}$ -expressions). Let  $\mathcal{G}$  be a non-deterministic functor and  $\mathcal{F}$  an ingredient of  $\mathcal{G}$ . We define  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$  by:

$$\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} = \{\varepsilon \in \text{Exp}^c \mid \vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{G}\}.$$

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

Let us instantiate the definition of  $\mathcal{G}$ -expressions to the functors of deterministic automata  $\mathcal{D} = 2 \times \text{Id}^A$ .

**5.2.4 EXAMPLE** (Deterministic expressions). Let  $A$  be a finite set of input actions and let  $X$  be a set of fixed point variables. The set  $\text{Exp}_{\mathcal{D}}$  of *deterministic expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For  $a \in A$  and  $x \in X$ :

$$\begin{aligned} \text{Exp}_{\mathcal{D}} \ni \varepsilon &::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \mid x \mid l(\varepsilon_1) \mid r(\varepsilon_2) \\ \varepsilon_1 &::= \underline{\emptyset} \mid 0 \mid 1 \mid \varepsilon_1 \oplus \varepsilon_1 \\ \varepsilon_2 &::= \underline{\emptyset} \mid a(\varepsilon) \mid \varepsilon_2 \oplus \varepsilon_2 \end{aligned}$$

♠

Examples of well-typed expressions for the functor  $\mathcal{D} = 2 \times \text{Id}^A$  (with  $2 = \{0, 1\}$  a two-element join-semilattice with 0 as bottom element; recall that the ingredients of  $\mathcal{D}$  are 2,  $\text{Id}^A$  and  $\mathcal{D}$  itself) include  $r(a(\underline{\emptyset}))$ ,  $l(1) \oplus r(a(l(0)))$  and  $\mu x. r(a(x)) \oplus l(1)$ . The expressions  $l[1]$ ,  $l(1) \oplus 1$  and  $\mu x. 1$  are examples of non well-typed expressions for  $\mathcal{D}$ , because the functor  $\mathcal{D}$  does not involve  $\oplus$ , the subexpressions in the sum have different type, and recursion is not at the outermost level (1 has type  $2 \triangleleft \mathcal{D}$ ), respectively.

It is easy to see that the closed (and guarded) expressions generated by the grammar presented above are exactly the elements of  $\text{Exp}_{\mathcal{D}}$ . The most interesting case to check is the expression  $r(a(\varepsilon))$ . Note that  $a(\varepsilon)$  has type  $\text{Id}^A \triangleleft \mathcal{D}$  as long as  $\varepsilon$  has type  $\text{Id} \triangleleft \mathcal{D}$ . And the crucial remark here is that, by definition of  $\vdash$ ,  $\text{Exp}_{\text{Id} \triangleleft \mathcal{G}} \subseteq \text{Exp}_{\mathcal{G}}$ . Therefore,  $\varepsilon$  has type  $\text{Id} \triangleleft \mathcal{D}$  if it is of type  $\mathcal{D} \triangleleft \mathcal{D}$ , or more precisely, if  $\varepsilon \in \text{Exp}_{\mathcal{D}}$ , which explains why the grammar above is correct.

At this point, we should remark that the syntax of our expressions differs from the classical regular expressions in the use of  $\mu$  and action prefixing  $a(\varepsilon)$  instead of Kleene star and full concatenation. We shall prove later that these two syntactically different formalisms are equally expressive (Theorems 5.2.12 and 5.2.14), but, to increase the intuition behind our expressions, let us present the syntactic translation from classical regular expressions to  $\text{Exp}_{\mathcal{D}}$  (this translation is inspired by [84]) and back.

**5.2.5 DEFINITION.** The set of regular expressions is given by the following syntax

$$RE \ni r ::= \underline{0} \mid \underline{1} \mid a \mid r + r \mid r \cdot r \mid r^*$$

where  $a \in A$  and  $\cdot$  denotes sequential composition. We define the following translations between regular expressions and deterministic expressions:

$(-)^{\dagger}: RE \rightarrow \text{Exp}_{\mathcal{D}}$	$(-)^{\ddagger}: \text{Exp}_{\mathcal{D}} \rightarrow RE$
$(\underline{0})^{\dagger} = \underline{\emptyset}$	$(\underline{\emptyset})^{\ddagger} = \underline{0}$
$(\underline{1})^{\dagger} = l\langle 1 \rangle$	$(l\langle \underline{\emptyset} \rangle)^{\ddagger} = (l\langle 0 \rangle)^{\ddagger} = (r\langle \underline{\emptyset} \rangle)^{\ddagger} = \underline{0}$
$(a)^{\dagger} = r\langle a(l\langle 1 \rangle) \rangle$	$(l\langle 1 \rangle)^{\ddagger} = \underline{1}$
$(r_1 + r_2)^{\dagger} = (r_1)^{\dagger} \oplus (r_2)^{\dagger}$	$(l\langle \varepsilon_1 \oplus \varepsilon'_1 \rangle)^{\ddagger} = (l\langle \varepsilon_1 \rangle)^{\ddagger} + (l\langle \varepsilon'_1 \rangle)^{\ddagger}$
$(r_1 \cdot r_2)^{\dagger} = (r_1)^{\dagger} [(r_2)^{\dagger} / l\langle 1 \rangle]$	$(r\langle a(\varepsilon) \rangle)^{\ddagger} = a \cdot (\varepsilon)^{\ddagger}$
$(r^*)^{\dagger} = \mu x. (r)^{\dagger} [x / l\langle 1 \rangle] \oplus l\langle 1 \rangle$	$(r\langle \varepsilon_2 \oplus \varepsilon'_2 \rangle)^{\ddagger} = (r\langle \varepsilon_2 \rangle)^{\ddagger} + (r\langle \varepsilon'_2 \rangle)^{\ddagger}$
	$(\varepsilon_1 \oplus \varepsilon_2)^{\ddagger} = (\varepsilon_1)^{\ddagger} + (\varepsilon_2)^{\ddagger}$
	$(\mu x. \varepsilon)^{\ddagger} = \mathbf{sol}(\mathbf{eqs}(\mu x. \varepsilon))$

The function  $\mathbf{eqs}$  translates  $\mu x. \varepsilon$  into a system of equations in the following way. Let  $\mu x_1. \varepsilon_1, \dots, \mu x_n. \varepsilon_n$  be all the fixed point subexpressions of  $\mu x. \varepsilon$ , with  $x_1 = x$  and  $\varepsilon_1 = \varepsilon$ . We define  $n$  equations  $x_i = (\bar{\varepsilon}_i)^{\dagger}$ , where  $\bar{\varepsilon}_i$  is obtained from  $\varepsilon_i$  by replacing each subexpression  $\mu x_i. \varepsilon_i$  by  $x_i$ , for all  $i = 1, \dots, n$ . The solution of the system,  $\mathbf{sol}(\mathbf{eqs}(\mu x. \varepsilon))$ , is then computed in the usual way (the solution of an equation of shape  $x = rx + t$  is  $r^*t$ ).

In the previous chapter, regular expressions were given a coalgebraic structure, using Brzozowski derivatives [29]. Later in this chapter, we will provide a coalgebra structure to  $\text{Exp}_{\mathcal{D}}$ , after which the soundness of the above translations can be stated and proved:  $r \sim r^{\dagger}$  and  $\varepsilon \sim \varepsilon^{\ddagger}$ , where  $\sim$  will coincide with language equivalence. ♣

Thus, the regular expression  $aa^*$  is translated to  $r\langle a(\mu x. r\langle a(x) \rangle \oplus l\langle 1 \rangle) \rangle$ , whereas the expression  $\mu x. r\langle a(r\langle a(x) \rangle) \rangle \oplus l\langle 1 \rangle$  is transformed into  $(aa)^*$ .

We present next the syntax for the expressions in  $\text{Exp}_{\mathcal{Q}}$  and in  $\text{Exp}_{\mathcal{N}}$  (recall that  $\mathcal{Q} = (1 \oplus \text{Id})^A$  and  $\mathcal{N} = 2 \times (\mathcal{P}_{\omega} \text{Id})^A$ ).

**5.2.6 EXAMPLE** (Partial expressions). Let  $A$  be a finite set of input actions and  $X$  be a set of fixed point variables. The set  $\text{Exp}_{\mathcal{Q}}$  of *partial expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For  $a \in A$  and  $x \in X$ :

$$\begin{aligned} \text{Exp}_{\mathcal{Q}} \ni \varepsilon &::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \mid x \mid a(\varepsilon_1) \\ \varepsilon_1 &::= \underline{\emptyset} \mid \varepsilon_1 \oplus \varepsilon_1 \mid l[\varepsilon_2] \mid r[\varepsilon] \\ \varepsilon_2 &::= \underline{\emptyset} \mid \varepsilon_2 \oplus \varepsilon_2 \mid * \end{aligned}$$

Intuitively, the expressions  $a(l[*])$  and  $a(r[\varepsilon])$  specify, respectively, a state which has no defined transition for input  $a$  and a state with an outgoing transition to another state as specified by  $\varepsilon$ . ♠

**5.2.7 EXAMPLE** (Non-deterministic expressions). Let  $A$  be a finite set of input actions and  $X$  be a set of fixed point variables. The set  $\text{Exp}_{\mathcal{N}}$  of *non-deterministic expressions*



is given by the set of closed and guarded expressions generated by the following BNF grammar. For  $a \in A$  and  $x \in X$ :

$$\begin{aligned} \text{Exp}_{\mathcal{N}} \ni \varepsilon &::= \underline{\emptyset} \mid x \mid l\langle \varepsilon_1 \rangle \mid r\langle \varepsilon_2 \rangle \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \\ \varepsilon_1 &::= \underline{\emptyset} \mid \varepsilon_1 \oplus \varepsilon_1 \mid 1 \mid 0 \\ \varepsilon_2 &::= \underline{\emptyset} \mid \varepsilon_2 \oplus \varepsilon_2 \mid a(\varepsilon') \\ \varepsilon' &::= \underline{\emptyset} \mid \varepsilon' \oplus \varepsilon' \mid \{\varepsilon\} \end{aligned}$$

Intuitively, the expression  $r\langle a(\{\varepsilon_1\} \oplus \{\varepsilon_2\}) \rangle$  specifies a state which has two outgoing transitions labeled with  $a$ , one to a state specified by  $\varepsilon_1$  and another to a state specified by  $\varepsilon_2$ .  $\spadesuit$

We have defined a language of expressions which gives us an algebraic description of systems. We should also remark at this point that in the examples we strictly follow the type system to derive the syntax of the expressions. However, it is obvious that many simplifications can be made in order to obtain a more polished language. In particular, after the axiomatization we will be able to decrease the number of levels in the above grammars, since we will have axioms of the shape  $a(\varepsilon) \oplus a(\varepsilon') \equiv a(\varepsilon \oplus \varepsilon')$ . In Section 5.4, we will sketch two examples where we apply some simplification to the syntax. Also note that the language presented in Chapter 4 can be obtained from  $\text{Exp}_{\mathcal{M}}$ , with  $\mathcal{M} = (\mathbb{B} \times \text{Id})^A$ , by abbreviating  $a(r\langle \varepsilon \rangle)$  by  $a(\varepsilon)$  and  $a(l\langle b \rangle)$  by  $a \downarrow b$  and applying the simplification mentioned above to eliminate redundant expressions.

The goal is now to present a generalization of Kleene's theorem for non-deterministic coalgebras (Theorems 5.2.12 and 5.2.14). Recall that, for regular languages, the theorem states that a language is regular if and only if it is recognized by a finite automaton. In order to achieve our goal we will first show that the set  $\text{Exp}_{\mathcal{G}}$  of  $\mathcal{G}$ -expressions carries a  $\mathcal{G}$ -coalgebra structure.

### 5.2.1 Brzozowski derivatives for non-deterministic expressions

In this section, we show that the set of  $\mathcal{G}$ -expressions for a given non-deterministic functor  $\mathcal{G}$  has a coalgebraic structure  $\delta_{\mathcal{G}}: \text{Exp}_{\mathcal{G}} \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}})$ . More precisely, we are going to define a function

$$\delta_{\mathcal{F} \triangleleft \mathcal{G}}: \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}})$$

for every ingredient  $\mathcal{F}$  of  $\mathcal{G}$ , and then set  $\delta_{\mathcal{G}} = \delta_{\mathcal{G} \triangleleft \mathcal{G}}$ . Our definition of the function  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}$  will make use of the following.

**5.2.8 DEFINITION.** For every  $\mathcal{G} \in \text{NDF}$  and for every  $\mathcal{F}$  with  $\mathcal{F} \triangleleft \mathcal{G}$ :

- (i) we define a constant  $\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}} \in \mathcal{F}(\text{Exp}_{\mathcal{G}})$  by induction on the syntactic structure of  $\mathcal{F}$ :

$$\begin{array}{ll} \text{Empty}_{\text{Id} \triangleleft \mathcal{G}} &= \underline{\emptyset} & \text{Empty}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}} &= \perp \\ \text{Empty}_{\mathbb{B} \triangleleft \mathcal{G}} &= \perp_{\mathbb{B}} & \text{Empty}_{\mathcal{F}^A \triangleleft \mathcal{G}} &= \lambda a. \text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}} \\ \text{Empty}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}} &= \langle \text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}, \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle & \text{Empty}_{\mathcal{P}_{\omega} \mathcal{F} \triangleleft \mathcal{G}} &= \emptyset \end{array}$$

- (ii) we define a function  $\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}} : \mathcal{F}(\text{Exp}_{\mathcal{G}}) \times \mathcal{F}(\text{Exp}_{\mathcal{G}}) \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}})$  by induction on the syntactic structure of  $\mathcal{F}$ :

$$\begin{aligned}
\text{Plus}_{\text{id} \triangleleft \mathcal{G}}(\varepsilon_1, \varepsilon_2) &= \varepsilon_1 \oplus \varepsilon_2 \\
\text{Plus}_{\mathbb{B} \triangleleft \mathcal{G}}(b_1, b_2) &= b_1 \vee_{\mathbb{B}} b_2 \\
\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\langle \varepsilon_1, \varepsilon_2 \rangle, \langle \varepsilon_3, \varepsilon_4 \rangle) &= \langle \text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1, \varepsilon_3), \text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon_2, \varepsilon_4) \rangle \\
\text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\kappa_i(\varepsilon_1), \kappa_i(\varepsilon_2)) &= \kappa_i(\text{Plus}_{\mathcal{F}_i \triangleleft \mathcal{G}}(\varepsilon_1, \varepsilon_2)), \quad i \in \{1, 2\} \\
\text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\kappa_i(\varepsilon_1), \kappa_j(\varepsilon_2)) &= \top \quad i, j \in \{1, 2\} \text{ and } i \neq j \\
\text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(x, \top) &= \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\top, x) = \top \\
\text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(x, \perp) &= \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\perp, x) = x \\
\text{Plus}_{\mathcal{F}^A \triangleleft \mathcal{G}}(f, g) &= \lambda a. \text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(f(a), g(a)) \\
\text{Plus}_{\mathcal{P}_o \mathcal{F} \triangleleft \mathcal{G}}(s_1, s_2) &= s_1 \cup s_2
\end{aligned}$$

♣

Intuitively, one can think of the constant  $\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}}$  and the function  $\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}$  as liftings of  $\emptyset$  and  $\oplus$  to the level of  $\mathcal{F}(\text{Exp}_{\mathcal{G}})$ .

We need two more things to define  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}$ . First, we define an order  $\preceq$  on the types of expressions. For  $\mathcal{F}_1, \mathcal{F}_2$  and  $\mathcal{G}$  non-deterministic functors such that  $\mathcal{F}_1 \triangleleft \mathcal{G}$  and  $\mathcal{F}_2 \triangleleft \mathcal{G}$ , we define

$$(\mathcal{F}_1 \triangleleft \mathcal{G}) \preceq (\mathcal{F}_2 \triangleleft \mathcal{G}) \Leftrightarrow \mathcal{F}_1 \triangleleft \mathcal{F}_2$$

The order  $\preceq$  is a partial order (structure inherited from  $\triangleleft$ ). Note also that  $(\mathcal{F}_1 \triangleleft \mathcal{G}) = (\mathcal{F}_2 \triangleleft \mathcal{G}) \Leftrightarrow \mathcal{F}_1 = \mathcal{F}_2$ . Second, we define a measure  $N(\varepsilon)$  based on the maximum number of nested unguarded occurrences of  $\mu$ -expressions in  $\varepsilon$  and unguarded occurrences of  $\oplus$ . We say that a subexpression  $\mu x. \varepsilon_1$  of  $\varepsilon$  occurs unguarded if it is not in the scope of one of the operators  $l(-), r(-), l[-], r[-], a(-)$  or  $\{-\}$ .

**5.2.9 DEFINITION.** For every guarded expression  $\varepsilon$ , we define  $N(\varepsilon)$  as follows:

$$\begin{aligned}
N(\emptyset) &= N(b) = N(a(\varepsilon)) = N(l(\varepsilon)) = N(r(\varepsilon)) = N(l[\varepsilon]) = N(r[\varepsilon]) = N(\{\varepsilon\}) = 0 \\
N(\varepsilon_1 \oplus \varepsilon_2) &= 1 + \max\{N(\varepsilon_1), N(\varepsilon_2)\} \\
N(\mu x. \varepsilon) &= 1 + N(\varepsilon)
\end{aligned}$$

♣

The measure  $N$  induces a partial order on the set of expressions:  $\varepsilon_1 \ll \varepsilon_2 \Leftrightarrow N(\varepsilon_1) \leq N(\varepsilon_2)$ , where  $\leq$  is just the ordinary inequality of natural numbers.

Now we have all we need to define  $\delta_{\mathcal{F} \triangleleft \mathcal{G}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}})$ .

**5.2.10 DEFINITION.** For every ingredient  $\mathcal{F}$  of a non-deterministic functor  $\mathcal{G}$  and an

expression  $\varepsilon \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ , we define  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)$  as follows:

$$\begin{aligned}
\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\emptyset) &= \text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}} \\
\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1 \oplus \varepsilon_2) &= \text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1), \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2)) \\
\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mu x. \varepsilon) &= \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon[\mu x. \varepsilon/x]) \\
\delta_{\text{Id} \triangleleft \mathcal{G}}(\varepsilon) &= \varepsilon \text{ for } \mathcal{G} \neq \text{Id} \\
\delta_{\text{B} \triangleleft \mathcal{G}}(b) &= b \\
\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\varepsilon)) &= \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle \\
\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(r(\varepsilon)) &= \langle \text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}, \delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon) \rangle \\
\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon]) &= \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon)) \\
\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(r[\varepsilon]) &= \kappa_2(\delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon)) \\
\delta_{\mathcal{F}^A \triangleleft \mathcal{G}}(a(\varepsilon)) &= \lambda a'. \begin{cases} \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon) & \text{if } a = a' \\ \text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}} & \text{otherwise} \end{cases} \\
\delta_{\mathcal{P}_\omega \mathcal{F} \triangleleft \mathcal{G}}(\{\varepsilon\}) &= \{\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)\}
\end{aligned}$$

Here,  $\varepsilon[\mu x. \varepsilon/x]$  denotes syntactic substitution, replacing every free occurrence of  $x$  in  $\varepsilon$  by  $\mu x. \varepsilon$ . ♣

In order to see that the definition of  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}$  is well-formed, we have to observe that  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}$  can be seen as a function having two arguments: the type  $\mathcal{F} \triangleleft \mathcal{G}$  and the expression  $\varepsilon$ . Then, we use induction on the Cartesian product of types and expressions with orders  $\preceq$  and  $\ll$ , respectively. More precisely, given two pairs  $\langle \mathcal{F}_1 \triangleleft \mathcal{G}, \varepsilon_1 \rangle$  and  $\langle \mathcal{F}_2 \triangleleft \mathcal{G}, \varepsilon_2 \rangle$  we have an order

$$\begin{aligned}
\langle \mathcal{F}_1 \triangleleft \mathcal{G}, \varepsilon_1 \rangle \leq \langle \mathcal{F}_2 \triangleleft \mathcal{G}, \varepsilon_2 \rangle &\iff \begin{aligned} &\text{(i) } \langle \mathcal{F}_1 \triangleleft \mathcal{G} \rangle \preceq \langle \mathcal{F}_2 \triangleleft \mathcal{G} \rangle \\ &\text{or (ii) } \langle \mathcal{F}_1 \triangleleft \mathcal{G} \rangle = \langle \mathcal{F}_2 \triangleleft \mathcal{G} \rangle \text{ and } \varepsilon_1 \ll \varepsilon_2 \end{aligned} \quad (5.1)
\end{aligned}$$

Observe that in the definition above it is always true that  $\langle \mathcal{F}' \triangleleft \mathcal{G}, \varepsilon' \rangle \leq \langle \mathcal{F} \triangleleft \mathcal{G}, \varepsilon \rangle$ , for all occurrences of  $\delta_{\mathcal{F}' \triangleleft \mathcal{G}}(\varepsilon')$  occurring in the right hand side of the equation defining  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)$ . In all cases, but the ones that  $\varepsilon$  is a fixed point or a sum expression, the inequality comes from point (i) above. For the case of the sum, note that  $\langle \mathcal{F} \triangleleft \mathcal{G}, \varepsilon_1 \rangle \leq \langle \mathcal{F} \triangleleft \mathcal{G}, \varepsilon_1 \oplus \varepsilon_2 \rangle$  and  $\langle \mathcal{F} \triangleleft \mathcal{G}, \varepsilon_2 \rangle \leq \langle \mathcal{F} \triangleleft \mathcal{G}, \varepsilon_1 \oplus \varepsilon_2 \rangle$  by point (ii), since  $N(\varepsilon_1) < N(\varepsilon_1 \oplus \varepsilon_2)$  and  $N(\varepsilon_2) < N(\varepsilon_1 \oplus \varepsilon_2)$ . Similarly, in the case of  $\mu x. \varepsilon$  we have that  $N(\varepsilon) = N(\varepsilon[\mu x. \varepsilon/x])$ , which can easily be proved by (standard) induction on the syntactic structure of  $\varepsilon$ , since  $\varepsilon$  is guarded (in  $x$ ), and this guarantees that  $N(\varepsilon[\mu x. \varepsilon/x]) < N(\mu x. \varepsilon)$ . Hence,  $\langle \mathcal{G} \triangleleft \mathcal{G}, \varepsilon \rangle \leq \langle \mathcal{G} \triangleleft \mathcal{G}, \mu x. \varepsilon \rangle$ . Also note that clause 4 of the above definition overlaps with clauses 1 and 2 (by taking  $\mathcal{F} = \text{Id}$ ). However, they give the same result and thus the function  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}$  is well-defined.

**5.2.11 DEFINITION.** We define, for each non-deterministic functor  $\mathcal{G}$ , a  $\mathcal{G}$ -coalgebra

$$\delta_{\mathcal{G}}: \text{Exp}_{\mathcal{G}} \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}})$$

by putting  $\delta_{\mathcal{G}} = \delta_{\mathcal{G} \triangleleft \mathcal{G}}$ . ♣

The function  $\delta_{\mathcal{G}}$  can be thought of as the generalization of the well-known notion of Brzozowski derivative [29] for regular expressions and, moreover, it provides an operational semantics for expressions, as we shall see in Section 5.2.2.

The observation that the set of expressions has a coalgebra structure will be crucial for the proof of the generalized Kleene theorem, as will be shown in the next two sections.

## 5.2.2 From coalgebras to expressions

Having a  $\mathcal{G}$ -coalgebra structure on  $\text{Exp}_{\mathcal{G}}$  has two advantages. First, it provides us, by finality, directly with a natural semantics because of the existence of a (unique) homomorphism  $[[\cdot]]: \text{Exp}_{\mathcal{G}} \rightarrow \Omega_{\mathcal{G}}$ , that assigns to every expression  $\varepsilon$  an element  $[[\varepsilon]]$  of the behavior final coalgebra  $\Omega_{\mathcal{G}}$ .

The second advantage of the coalgebra structure on  $\text{Exp}_{\mathcal{G}}$  is that it allows us to use the notion of  $\mathcal{G}$ -bisimulation to relate  $\mathcal{G}$ -coalgebras  $(S, g)$  and expressions  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ . If one can construct a bisimulation relation between an expression  $\varepsilon$  and a state  $s$  of a given coalgebra, then the behavior represented by  $\varepsilon$  is equal to the behavior of the state  $s$ . This is the analogue of computing the language  $L(r)$  represented by a given regular expression  $r$  and the language  $L(s)$  accepted by a state  $s$  of a finite state automaton and checking whether  $L(r) = L(s)$ .

The following theorem states that every state in a locally finite  $\mathcal{G}$ -coalgebra can be represented by an expression in our language. This generalizes *half* of Kleene's theorem for deterministic automata: if a language is accepted by a finite automaton then it is regular (*i.e.* it can be denoted by a regular expression). The generalization of the other *half* of the theorem (if a language is regular then it is accepted by a finite automaton) will be presented in Section 5.2.3. It is worth to remark that in the usual definition of deterministic automaton the initial state of the automaton is included and, thus, in the original Kleene's theorem, it was enough to consider finite automata. In the coalgebraic approach, the initial state is not explicitly modeled and thus we need to consider locally-finite coalgebras: coalgebras where each state will generate a finite subcoalgebra.

**5.2.12 THEOREM.** *Let  $\mathcal{G}$  be a non-deterministic functor and let  $(S, g)$  be a locally-finite  $\mathcal{G}$ -coalgebra. Then, for any  $s \in S$ , there exists an expression  $\langle\langle s \rangle\rangle \in \text{Exp}_{\mathcal{G}}$  such that  $s \sim \langle\langle s \rangle\rangle$ .*

PROOF. Let  $s \in S$  and let  $\langle s \rangle = \{s_1, \dots, s_n\}$  with  $s_1 = s$ . We construct, for every state  $s_i \in \langle s \rangle$ , an expression  $\langle\langle s_i \rangle\rangle$  such that  $s_i \sim \langle\langle s_i \rangle\rangle$ .

If  $\mathcal{G} = \text{Id}$ , we set, for every  $i$ ,  $\langle\langle s_i \rangle\rangle = \emptyset$ . It is easy to see that  $\{\langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle\}$  is a bisimulation and, thus, we have that  $s \sim \langle\langle s \rangle\rangle$ .

For  $\mathcal{G} \neq \text{Id}$ , we proceed in the following way. Let, for every  $i$ ,  $A_i = \mu x_i \cdot \gamma_{g(s_i)}^{\mathcal{G}}$  where, for  $\mathcal{F} \triangleleft \mathcal{G}$  and  $c \in \mathcal{F}(s)$ , the expression  $\gamma_c^{\mathcal{F}} \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$  is defined by induction on the

structure of  $\mathcal{F}$ :

$$\begin{aligned}
\gamma_{s_i}^{\text{Id}} &= x_i & \gamma_b^{\text{B}} &= b & \gamma_{(c,c')}^{\mathcal{F}_1 \times \mathcal{F}_2} &= l[\gamma_c^{\mathcal{F}_1}] \oplus r[\gamma_{c'}^{\mathcal{F}_2}] \\
\gamma_f^{\mathcal{F}^A} &= \bigoplus_{a \in A} a(\gamma_{f(a)}^{\mathcal{F}}) & \gamma_{\kappa_1(c)}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= l[\gamma_c^{\mathcal{F}_1}] & \gamma_{\kappa_2(c)}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= r[\gamma_c^{\mathcal{F}_2}] \\
\gamma_{\perp}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= \underline{\emptyset} & \gamma_{\top}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= l[\underline{\emptyset}] \oplus r[\underline{\emptyset}] \\
\gamma_C^{\mathcal{P}_\omega \mathcal{F}} &= \begin{cases} \bigoplus_{c \in C} \{\gamma_c^{\mathcal{F}}\} & \text{if } C \neq \emptyset \\ \underline{\emptyset} & \text{otherwise} \end{cases}
\end{aligned}$$

Note that here the choice of  $l[\underline{\emptyset}] \oplus r[\underline{\emptyset}]$  to represent inconsistency is arbitrary but *canonical*, in the sense that any other expression involving sum of  $l[\varepsilon_1]$  and  $r[\varepsilon_2]$  will be bisimilar. Formally, the definition of  $\gamma$  above is parametrized by a function from  $\{s_1, \dots, s_n\}$  to a fixed set of variables  $\{x_1, \dots, x_n\}$ . It should also be noted that, similarly to what happened in the previous chapter,  $\bigoplus E$ , for an ordered set  $E = \{\varepsilon_1, \dots, \varepsilon_n\}$  of expressions, stands for  $\varepsilon_1 \oplus (\varepsilon_2 \oplus (\varepsilon_3 \oplus \dots))$ .

Let  $A_i^0 = A_i$ , define  $A_i^{k+1} = A_i^k \{A_{k+1}^k / x_{k+1}\}$  and then set  $\langle\langle s_i \rangle\rangle = A_i^n$ . Here,  $A\{A'/x\}$  denotes syntactic replacement (that is, substitution without renaming of bound variables in  $A$  which are also free variables in  $A'$ ). The definition of  $\langle\langle s_i \rangle\rangle$  does not depend in the chosen order of  $\{s_1, \dots, s_n\}$ : the expressions obtained are just different modulo renaming of variables.

Observe that the term

$$A_i^n = (\mu x_i. \gamma_{g(s_i)}^{\mathcal{G}}) \{A_1^0 / x_1\} \dots \{A_n^{n-1} / x_n\}$$

is a closed term because, for every  $j = 1, \dots, n$ , the term  $A_j^{j-1}$  contains at most  $n - j$  free variables in the set  $\{x_{j+1}, \dots, x_n\}$ .

It remains to prove that  $s_i \sim \langle\langle s_i \rangle\rangle$ . We show that  $R = \{\langle s_i, \langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle\}$  is a bisimulation. For that, we first define, for  $\mathcal{F} \triangleleft \mathcal{G}$  and  $c \in \mathcal{F}\langle s \rangle$ ,  $\xi_c^{\mathcal{F}} = \gamma_c^{\mathcal{F}} \{A_1^0 / x_1\} \dots \{A_n^{n-1} / x_n\}$  and the relation

$$R_{\mathcal{F} \triangleleft \mathcal{G}} = \{\langle c, \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\xi_c^{\mathcal{F}}) \mid c \in \mathcal{F}\langle s \rangle\}.$$

Then, we prove that ①  $R_{\mathcal{F} \triangleleft \mathcal{G}} = \overline{\mathcal{F}}(R)$  and ②  $\langle g(s_i), \delta_{\mathcal{G}}(\langle\langle s_i \rangle\rangle) \rangle \in R_{\mathcal{G} \triangleleft \mathcal{G}}$ .

① By induction on the structure of  $\mathcal{F}$ .

$\boxed{\mathcal{F} = \text{Id}}$  Note that  $R_{\text{Id} \triangleleft \mathcal{G}} = \{\langle s_i, \xi_{s_i}^{\text{Id}} \rangle \mid s_i \in \langle s \rangle\}$  which is equal to  $\overline{\text{Id}}(R) = R$  provided

that  $\xi_{s_i}^{\text{ld}} = \langle\langle s_i \rangle\rangle$ . The latter is indeed the case:

$$\begin{aligned}
\xi_{s_i}^{\text{ld}} &= \gamma_{s_i}^{\text{ld}} \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\} && \text{(def. } \xi_{s_i}^{\text{ld}}) \\
&= x_i \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\} && \text{(def. } \gamma_{s_i}^{\text{ld}}) \\
&= A_i^{i-1} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} && (\{A_i^{i-1}/x_i\}) \\
&= A_i^0 \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\} && \text{(def. } A_i^{i-1}) \\
&= \langle\langle s_i \rangle\rangle && \text{(def. } \langle\langle s_i \rangle\rangle)
\end{aligned}$$

$\boxed{\mathcal{F} = \mathbf{B}}$  Note that, for  $b \in \mathbf{B}$ ,  $\xi_b^{\mathbf{B}} = \gamma_b^{\mathbf{B}} \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\} = b$ . Thus, we have that  $R_{\mathbf{B} \triangleleft \mathcal{G}} = \{\langle s_i, \xi_{s_i}^{\mathbf{B}} \rangle \mid s_i \in \mathbf{B}\langle s \rangle\} = \{\langle b, b \rangle \mid b \in \mathbf{B}\} = \overline{\mathbf{B}}(R)$ .

$\boxed{\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2}$

$$\begin{aligned}
&\langle\langle u, v \rangle, \langle e, f \rangle \rangle \in \overline{\mathcal{F}_1 \times \mathcal{F}_2}(R) \\
\iff &\langle u, e \rangle \in \overline{\mathcal{F}_1}(R) \text{ and } \langle v, f \rangle \in \overline{\mathcal{F}_2}(R) && \text{(def. } \overline{\mathcal{F}_1 \times \mathcal{F}_2}) \\
\iff &\langle u, e \rangle \in R_{\mathcal{F}_1 \triangleleft \mathcal{G}} \text{ and } \langle v, f \rangle \in R_{\mathcal{F}_2 \triangleleft \mathcal{G}} && \text{(ind. hyp.)} \\
\iff &\langle u, e \rangle = \langle c, \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\xi_c^{\mathcal{F}_1}) \rangle \text{ and } \langle v, f \rangle = \langle c', \delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\xi_{c'}^{\mathcal{F}_2}) \rangle && \text{(def. } R_{\mathcal{F}_i \triangleleft \mathcal{G}}) \\
\iff &\langle u, v \rangle = \langle c, c' \rangle \text{ and } \langle e, f \rangle = \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\xi_c^{\mathcal{F}_1}) \oplus r(\xi_{c'}^{\mathcal{F}_2})) && \text{(def. } \delta_{\mathcal{F} \triangleleft \mathcal{G}}) \\
\iff &\langle u, v \rangle = \langle c, c' \rangle \text{ and } \langle e, f \rangle = \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\xi_{\langle c, c' \rangle}^{\mathcal{F}_1 \times \mathcal{F}_2}) && \text{(def. } \xi^{\mathcal{F}}) \\
\iff &\langle\langle u, v \rangle, \langle e, f \rangle \rangle \in R_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}
\end{aligned}$$

$\boxed{\mathcal{F} = \mathcal{F}_1 \diamond \mathcal{F}_2}$ ,  $\boxed{\mathcal{F} = \mathcal{F}_1^A}$  and  $\boxed{\mathcal{F} = \mathcal{P}_\omega \mathcal{F}_1}$ : similar to  $\mathcal{F}_1 \times \mathcal{F}_2$ .

- ② We want to prove that  $\langle g(s_i), \delta_{\mathcal{G}}(\langle\langle s_i \rangle\rangle) \rangle \in R_{\mathcal{G} \triangleleft \mathcal{G}}$ . For that, we must show that  $g(s_i) \in \mathcal{G}\langle s \rangle$  and  $\delta_{\mathcal{G}}(\langle\langle s_i \rangle\rangle) = \delta_{\mathcal{G}}(\xi_{g(s_i)}^{\mathcal{G}})$ . The former follows by definition of  $\langle s \rangle$ , whereas for the latter we observe that:

$$\begin{aligned}
&\delta_{\mathcal{G}}(\langle\langle s_i \rangle\rangle) \\
&= \delta_{\mathcal{G}}((\mu x_i. \gamma_{g(s_i)}^{\mathcal{G}}) \{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\}) && \text{(def. of } \langle\langle s_i \rangle\rangle) \\
&= \delta_{\mathcal{G}}(\mu x_i. \gamma_{g(s_i)}^{\mathcal{G}} \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}) \\
&= \delta_{\mathcal{G}}(\gamma_{g(s_i)}^{\mathcal{G}} \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} [A_i^n/x_i]) && \text{(def. of } \delta_{\mathcal{G}}) \\
&= \delta_{\mathcal{G}}(\gamma_{g(s_i)}^{\mathcal{G}} \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\} \{A_i^n/x_i\}) && ([A_i^n/x_i] = \{A_i^n/x_i\}) \\
&= \delta_{\mathcal{G}}(\gamma_{g(s_i)}^{\mathcal{G}} \{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\} \{A_i^n/x_i\} \{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}) \\
&= \delta_{\mathcal{G}}(\xi_{g(s_i)}^{\mathcal{G}})
\end{aligned}$$

Here, note that  $[A_i^n/x_i] = \{A_i^n/x_i\}$ , because  $A_i^n$  has no free variables. The last two steps follow, respectively, because  $x_i$  is not free in  $A_{i+1}^i, \dots, A_n^{n-1}$  and:

$$\begin{aligned} & \{A_i^n/x_i\}\{A_{i+1}^i/x_{i+1}\}\dots\{A_n^{n-1}/x_n\} \\ = & \{A_i^{i-1}\{A_{i+1}^i/x_{i+1}\}\dots\{A_n^{n-1}/x_n\}/x_i\}\{A_{i+1}^i/x_{i+1}\}\dots\{A_n^{n-1}/x_n\} \\ = & \{A_i^{i-1}/x_i\}\{A_{i+1}^i/x_{i+1}\}\dots\{A_n^{n-1}/x_n\} \end{aligned} \quad (5.2)$$

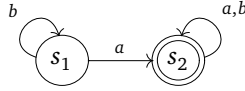
Equation (5.2) uses the syntactic identity

$$A\{B\{C/y\}/x\}\{C/y\} = A\{B/x\}\{C/y\}, \quad y \text{ not free in } C \quad (5.3)$$

□

Let us illustrate the construction appearing in the proof of Theorem 5.2.12 by some examples. These examples will illustrate the similarity with the proof of the original Kleene theorem, where a regular expression denoting the language recognized by a state of a deterministic automaton is built using a system of equations, which we recalled in Section 3.1.1.

Consider the following deterministic automaton over a two letter alphabet  $A = \{a, b\}$ , whose transition function  $g$  is given by the following picture (recall that  $\odot s$  represents that the state  $s$  is final):



We define  $A_1 = \mu x_1. \gamma_{g(s_1)}^D$  and  $A_2 = \mu x_2. \gamma_{g(s_2)}^D$  where

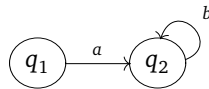
$$\gamma_{g(s_1)}^D = l\langle 0 \rangle \oplus r\langle b(x_1) \oplus a(x_2) \rangle \quad \gamma_{g(s_2)}^D = l\langle 1 \rangle \oplus r\langle a(x_2) \oplus b(x_2) \rangle$$

We have  $A_1^2 = A_1\{A_2^1/x_2\}$  and  $A_2^2 = A_2\{A_1^0/x_1\}$ . Thus,  $\langle\langle s_2 \rangle\rangle = A_2$  and, since  $A_2^1 = A_2$ ,  $\langle\langle s_1 \rangle\rangle$  is the expression

$$\mu x_1. l\langle 0 \rangle \oplus r\langle b(x_1) \oplus a(\mu x_2. l\langle 1 \rangle \oplus r\langle a(x_2) \oplus b(x_2) \rangle)\rangle$$

By construction we have  $s_1 \sim \langle\langle s_1 \rangle\rangle$  and  $s_2 \sim \langle\langle s_2 \rangle\rangle$ .

For another example, take the following partial automaton, also over a two letter alphabet  $A = \{a, b\}$ :



In the graphical representation of a partial automaton  $(S, p)$  we omit transitions for which  $p(s)(a) = \kappa_1(*)$ . In this case, this happens in  $q_1$  for the input letter  $b$  and in  $q_2$  for  $a$ .

We will have the equations

$$A_1 = A_1^0 = A_1^1 = \mu x_1. b(l[*]) \oplus a(r[x_2])$$

$$A_2 = A_2^0 = A_2^1 = \mu x_2. a(l[*]) \oplus b(r[x_2])$$

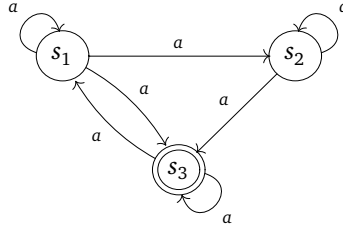
Thus:

$$\langle\langle s_1 \rangle\rangle = A_1^2 = \mu x_1. b(l[*]) \oplus a(r[\mu x_2. a(l[*]) \oplus b(r[x_2])])$$

$$\langle\langle s_2 \rangle\rangle = \mu x_2. a(l[*]) \oplus b(r[x_2])$$

Again we have  $s_1 \sim \langle\langle s_1 \rangle\rangle$  and  $s_2 \sim \langle\langle s_2 \rangle\rangle$ .

As a last example, let us consider the following non-deterministic automaton, over a one letter alphabet  $A = \{a\}$ :



We start with the equations:

$$A_1 = \mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\} \oplus \{x_2\} \oplus \{x_3\}) \rangle$$

$$A_2 = \mu x_2. l\langle 0 \rangle \oplus r\langle a(\{x_2\} \oplus \{x_3\}) \rangle$$

$$A_3 = \mu x_3. l\langle 1 \rangle \oplus r\langle a(\{x_1\} \oplus \{x_3\}) \rangle$$

Then we have the following iterations:

$$A_1^1 = A_1$$

$$A_1^2 = A_1\{A_2^1/x_2\} = \mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\} \oplus \{A_2\} \oplus \{x_3\}) \rangle$$

$$A_1^3 = A_1\{A_2^1/x_2\}\{A_3^2/x_3\} = \mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\} \oplus \{(A_2\{A_3^2/x_3\})\} \oplus \{A_3^2\}) \rangle$$

$$A_2^1 = A_2\{A_1/x_1\} = A_2$$

$$A_2^2 = A_2\{A_1/x_1\} = A_2$$

$$A_2^3 = A_2\{A_1/x_1\}\{A_3^2/x_3\} = \mu x_2. l\langle 0 \rangle \oplus r\langle a(\{x_2\} \oplus \{A_3^2\}) \rangle$$

$$A_3^1 = A_3\{A_1/x_1\} = \mu x_3. l\langle 1 \rangle \oplus r\langle a(\{A_1\} \oplus \{x_3\}) \rangle$$

$$A_3^2 = A_3\{A_1/x_1\}\{A_2^1/x_2\} = \mu x_3. l\langle 1 \rangle \oplus r\langle a(\{(A_1\{A_2^1/x_2\})\} \oplus \{x_3\}) \rangle$$

$$A_3^3 = A_3^2$$

This yields the following expressions:

$$\langle\langle s_1 \rangle\rangle = \mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\} \oplus \{\langle\langle s_2 \rangle\rangle\} \oplus \{\langle\langle s_3 \rangle\rangle\}) \rangle$$

$$\langle\langle s_2 \rangle\rangle = \mu x_2. l\langle 0 \rangle \oplus r\langle a(\{x_2\} \oplus \{\langle\langle s_3 \rangle\rangle\}) \rangle$$

$$\langle\langle s_3 \rangle\rangle = \mu x_3. l\langle 1 \rangle \oplus r\langle a(\{\mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\} \oplus \{\mu x_2. l\langle 0 \rangle \oplus r\langle a(\{x_2\} \oplus \{x_3\}) \rangle\} \oplus \{x_3\}) \rangle\} \oplus \{x_3\}) \rangle$$



### 5.2.3 From expressions to coalgebras

We prove the converse of Theorem 5.2.12, that is, we show how to construct a *finite*  $\mathcal{G}$ -coalgebra  $(S, g)$  from an arbitrary expression  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ , such that there exists a state  $s \in S$  with  $\varepsilon \sim_{\mathcal{G}} s$ .

The immediate way of obtaining a coalgebra from an expression  $\varepsilon \in \text{Exp}_{\mathcal{G}}$  is to compute the subcoalgebra  $\langle \varepsilon \rangle$ , since we have provided the set  $\text{Exp}_{\mathcal{G}}$  with a coalgebra structure  $\delta_{\mathcal{G}}: \text{Exp}_{\mathcal{G}} \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}})$ . However, the subcoalgebra generated by an expression  $\varepsilon \in \text{Exp}_{\mathcal{G}}$  by repeatedly applying  $\delta_{\mathcal{G}}$  is, in general, infinite. Take for instance the deterministic expression  $\varepsilon_1 = \mu x. r\langle a(x \oplus \mu y. r\langle a(y) \rangle) \rangle$  (for simplicity, we consider  $A = \{a\}$  and below we will write, in the second component of  $\delta_{\mathcal{D}}$ , an expression  $\varepsilon$  instead of the function mapping  $a$  to  $\varepsilon$ ) and observe that:

$$\begin{aligned} \delta_{\mathcal{D}}(\varepsilon_1) &= \langle 0, \varepsilon_1 \oplus \mu y. r\langle a(y) \rangle \rangle \\ \delta_{\mathcal{D}}(\varepsilon_1 \oplus \mu y. r\langle a(y) \rangle) &= \langle 0, \varepsilon_1 \oplus \mu y. r\langle a(y) \rangle \oplus \mu y. r\langle a(y) \rangle \rangle \\ \delta_{\mathcal{D}}(\varepsilon_1 \oplus \mu y. r\langle a(y) \rangle \oplus \mu y. r\langle a(y) \rangle) &= \langle 0, \varepsilon_1 \oplus \mu y. r\langle a(y) \rangle \oplus \mu y. r\langle a(y) \rangle \oplus \mu y. r\langle a(y) \rangle \rangle \\ &\vdots \end{aligned}$$

As one would expect, all the new states are equivalent and will be identified by  $[[ - ]]$  (the morphism into the final coalgebra). However, the function  $\delta_{\mathcal{D}}$  does not make any state identification and thus yields an infinite coalgebra.

This phenomenon occurs also in classical regular expressions. In Section 3.1.2, we showed that normalizing the expressions using the axioms for associativity, commutativity and idempotency was enough to guarantee finiteness<sup>1</sup>. We will show in this section that this also holds in our setting.

Consider the following axioms (only the first three are essential, but we include the fourth to obtain smaller coalgebras):

$$\begin{array}{ll} \text{(Associativity)} & \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \\ \text{(Commutativity)} & \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1 \\ \text{(Idempotency)} & \varepsilon \oplus \varepsilon \equiv \varepsilon \\ \text{(Empty)} & \underline{\emptyset} \oplus \varepsilon \equiv \varepsilon \end{array}$$

We define the relation  $\equiv_{ACIE} \subseteq \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \times \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ , written infix, as the least equivalence relation containing the four identities above. The relation  $\equiv_{ACIE}$  gives rise to the (surjective) equivalence map  $[\varepsilon]_{ACIE} = \{\varepsilon' \mid \varepsilon \equiv_{ACIE} \varepsilon'\}$ . The following diagram shows

<sup>1</sup>Actually, to guarantee finiteness, it is enough to eliminate double occurrences of expressions  $\varepsilon$  at the outermost level of an expression  $\dots \oplus \varepsilon \oplus \dots \oplus \varepsilon \oplus \dots$  (and to do this one needs the ACI axioms). Note that this is weaker than taking expressions modulo the ACI axioms: for instance, the expressions  $\varepsilon_1 \oplus \varepsilon_2$  and  $\varepsilon_2 \oplus \varepsilon_1$ , for  $\varepsilon_1 \neq \varepsilon_2$ , would not be identified in the process above.

the maps defined so far:

$$\begin{array}{ccc} \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} & \xrightarrow{[-]_{ACIE}} & \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv_{ACIE} \\ \delta_{\mathcal{F} \triangleleft \mathcal{G}} \downarrow & & \\ \mathcal{F}(\text{Exp}_{\mathcal{G}}) & \xrightarrow{\mathcal{F}([-]_{ACIE})} & \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv_{ACIE}) \end{array}$$

In order to complete the diagram, we next prove that  $\equiv_{ACIE}$  is contained in the kernel of  $\mathcal{F}([-]_{ACIE}) \circ \delta_{\mathcal{F} \triangleleft \mathcal{G}}$ <sup>2</sup>. This will guarantee the existence of a function

$$\bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv_{ACIE} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv_{ACIE})$$

which, when  $\mathcal{F} = \mathcal{G}$ , provides  $\text{Exp}_{\mathcal{G}} / \equiv$  with a coalgebraic structure

$$\bar{\delta}_{\mathcal{G}} : \text{Exp}_{\mathcal{G}} / \equiv_{ACIE} \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}} / \equiv_{ACIE})$$

(as before we write  $\bar{\delta}_{\mathcal{G}}$  for  $\bar{\delta}_{\mathcal{G} \triangleleft \mathcal{G}}$ ) and which makes  $[-]_{ACIE}$  a homomorphism of coalgebras.

**5.2.13 LEMMA.** *Let  $\mathcal{G}$  and  $\mathcal{F}$  be non-deterministic functors, with  $\mathcal{F} \triangleleft \mathcal{G}$ . For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ ,*

$$\varepsilon_1 \equiv_{ACIE} \varepsilon_2 \Rightarrow (\mathcal{F}([-]_{ACIE}))(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1)) = (\mathcal{F}([-]_{ACIE}))(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2))$$

**PROOF.** In order to improve readability, in this proof we will use  $[-]$  to denote  $[-]_{ACIE}$ . It is enough to prove that for all  $x_1, x_2, x_3 \in \mathcal{F}(\text{Exp}_{\mathcal{G}})$ :

- ①  $\mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_1, x_2), x_3)) = \mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_1, \text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_2, x_3)))$
- ②  $\mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_1, x_2)) = \mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_2, x_1))$
- ③  $\mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_1, x_1)) = \mathcal{F}([-])(x_1)$
- ④  $\mathcal{F}([-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}}, x_1)) = \mathcal{F}([-])(x_1)$

By induction on the structure of  $\mathcal{F}$ . We illustrate a few cases, the omitted ones are proved in a similar way.

$$\boxed{\mathcal{F} = \text{Id}} \quad x_1, x_2, x_3 \in \text{Exp}_{\mathcal{G}}$$

- ①  $\begin{aligned} & [\text{Plus}_{\text{Id} \triangleleft \mathcal{G}}(\text{Plus}_{\text{Id} \triangleleft \mathcal{G}}(x_1, x_2), x_3)] \\ &= [(x_1 \oplus x_2) \oplus x_3] && \text{(def. Plus)} \\ &= [x_1 \oplus (x_2 \oplus x_3)] && \text{(Associativity)} \\ &= [\text{Plus}_{\text{Id} \triangleleft \mathcal{G}}(x_1, \text{Plus}_{\text{Id} \triangleleft \mathcal{G}}(x_2, x_3))] && \text{(def. Plus)} \end{aligned}$
- ④  $\begin{aligned} & [\text{Plus}_{\text{Id} \triangleleft \mathcal{G}}(\text{Empty}_{\text{Id} \triangleleft \mathcal{G}}, x_1)] \\ &= [\emptyset \oplus x_1] && \text{(def. Plus and Empty)} \\ &= [x_1] && \text{(Empty)} \end{aligned}$

<sup>2</sup>This is equivalent to prove that  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv_{ACIE}$ , together with  $[-]_{ACIE}$ , is the coequalizer of the projection morphisms from  $\equiv_{ACIE}$  to  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ .

$$\boxed{\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2} \quad x_1 = \langle u_1, v_1 \rangle, x_2 = \langle u_2, v_2 \rangle \in (\mathcal{F}_1 \times \mathcal{F}_2)(\text{Exp}_{\mathcal{G}})$$

$$\begin{aligned} \textcircled{2} \quad & (\mathcal{F}_1 \times \mathcal{F}_2)([-])(\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle)) \\ &= \langle \mathcal{F}_1([-])(\text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(u_1, u_2)), \mathcal{F}_2([-])(\text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(v_1, v_2)) \rangle \quad (\text{def. Plus}) \\ &= \langle \mathcal{F}_1([-])(\text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(u_2, u_1)), \mathcal{F}_2([-])(\text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(v_2, v_1)) \rangle \quad (\text{ind. hyp.}) \\ &= (\mathcal{F}_1 \times \mathcal{F}_2)([-])(\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\langle u_2, v_2 \rangle, \langle u_1, v_1 \rangle)) \quad (\text{def. Plus}) \\ \\ \textcircled{3} \quad & (\mathcal{F}_1 \times \mathcal{F}_2)([-])(\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\langle u_1, v_1 \rangle, \langle u_1, v_1 \rangle)) \\ &= \langle \mathcal{F}_1([-])(\text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(u_1, u_1)), \mathcal{F}_2([-])(\text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(v_1, v_1)) \rangle \quad (\text{def. Plus}) \\ &= \langle \mathcal{F}_1([-])(u_1), \mathcal{F}_2([-])(v_1) \rangle \quad (\text{ind. hyp.}) \\ &= (\mathcal{F}_1 \times \mathcal{F}_2)([-])(\langle u_1, v_1 \rangle) \end{aligned}$$

$$\boxed{\mathcal{F} = \mathcal{P}_{\omega} \mathcal{F}_1} \quad x_1, x_2, x_3 \in \mathcal{P}_{\omega} \mathcal{F}_1(\text{Exp}_{\mathcal{G}})$$

$$\begin{aligned} \textcircled{1} \quad & \mathcal{P}_{\omega} \mathcal{F}_1([-])(\text{Plus}_{\mathcal{P}_{\omega} \mathcal{F}_1 \triangleleft \mathcal{G}}(x_1, \text{Plus}_{\mathcal{P}_{\omega} \mathcal{F}_1 \triangleleft \mathcal{G}}(x_2, x_3))) \\ &= \mathcal{P}_{\omega} \mathcal{F}_1([-])(x_1 \cup (x_2 \cup x_3)) \quad (\text{def. Plus}) \\ &= \mathcal{P}_{\omega} \mathcal{F}_1([-])(x_1 \cup x_2) \cup x_3 \\ &= \mathcal{P}_{\omega} \mathcal{F}_1([-])(\text{Plus}_{\mathcal{P}_{\omega} \mathcal{F}_1 \triangleleft \mathcal{G}}(\text{Plus}_{\mathcal{P}_{\omega} \mathcal{F}_1 \triangleleft \mathcal{G}}(x_1, x_2), x_3)) \quad (\text{def. Plus}) \end{aligned}$$

In the last but one step, we use the fact that, for any set  $X$ ,  $(\mathcal{P}_{\omega}(X), \cup, \emptyset)$  is a join-semilattice (hence,  $x_1 \cup (x_2 \cup x_3) = (x_1 \cup x_2) \cup x_3$ ). Due to this fact, in the case  $\mathcal{F} = \mathcal{P}_{\omega} \mathcal{F}_1$ , in this particular proof, the induction hypothesis will not be used.  $\square$

Thus, by Theorem 2.2.7, we have a well-defined function

$$\bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv_{ACIE} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv_{ACIE})$$

such that  $\bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}}([\varepsilon]_{ACIE}) = (\mathcal{F}[-]_{ACIE})(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon))$ .

We are ready to state and prove the second half of Kleene's theorem.

**5.2.14 THEOREM.** *Let  $\mathcal{G}$  be a non-deterministic functor. For every  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ , there exists  $\Delta_{\mathcal{G}}(\varepsilon) = (S, g)$  such that  $S$  is finite and there exists  $s \in S$  with  $\varepsilon \sim s$ .*

PROOF. For every  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ , we set  $\Delta_{\mathcal{G}}(\varepsilon) = \langle [\varepsilon]_{ACIE} \rangle$  (recall that  $\langle s \rangle$  denotes the smallest subcoalgebra generated by  $s$ ). First note that, by Lemma 5.2.13, the map  $[-]_{ACIE}$  is a homomorphism and thus  $\varepsilon \sim [\varepsilon]_{ACIE}$ . We prove, for every  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ , that the subcoalgebra  $\langle [\varepsilon]_{ACIE} \rangle = (V, \bar{\delta}_{\mathcal{G}})$  has a finite state space  $V$  (here,  $\bar{\delta}_{\mathcal{G}}$  actually stands for the restriction of  $\bar{\delta}_{\mathcal{G}}$  to  $V$ ). Again, in order to improve readability, in this proof we will use  $[-]$  to denote  $[-]_{ACIE}$ .

More precisely, we prove, for all  $\varepsilon \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ , the following inclusion

$$V \subseteq \bar{V} = \{[\varepsilon_1 \oplus \dots \oplus \varepsilon_k] \mid \varepsilon_1, \dots, \varepsilon_k \in cl(\varepsilon) \text{ all distinct, } \varepsilon_1, \dots, \varepsilon_k \in \text{Exp}_{\mathcal{G}}\} \quad (5.4)$$

Here, if  $k = 0$  we take the sum above to be  $\emptyset$  and  $cl(\varepsilon)$  denotes the smallest set containing all subformulas of  $\varepsilon$  and the unfoldings of  $\mu$  (sub)formulas, that is, the

smallest subset satisfying:

$$\begin{aligned}
cl(\emptyset) &= \{\emptyset\} \\
cl(\varepsilon_1 \oplus \varepsilon_2) &= \{\varepsilon_1 \oplus \varepsilon_2\} \cup cl(\varepsilon_1) \cup cl(\varepsilon_2) \\
cl(\mu x. \varepsilon_1) &= \{\mu x. \varepsilon_1\} \cup cl(\varepsilon_1[\mu x. \varepsilon_1/x]) \\
cl(l\langle \varepsilon_1 \rangle) &= \{l\langle \varepsilon_1 \rangle\} \cup cl(\varepsilon_1) \\
cl(r\langle \varepsilon_1 \rangle) &= \{r\langle \varepsilon_1 \rangle\} \cup cl(\varepsilon_1) \\
cl(l[\varepsilon_1]) &= \{l[\varepsilon_1]\} \cup cl(\varepsilon_1) \\
cl(r[\varepsilon_1]) &= \{r[\varepsilon_1]\} \cup cl(\varepsilon_1) \\
cl(a(\varepsilon_1)) &= \{a(\varepsilon_1)\} \cup cl(\varepsilon_1) \\
cl(\{\varepsilon_1\}) &= \{\{\varepsilon_1\}\} \cup cl(\varepsilon_1)
\end{aligned}$$

Note that the set  $cl(\varepsilon)$  is finite (the number of different unfoldings of  $\mu$ -expressions is finite) and has the property  $\varepsilon \in cl(\varepsilon)$ .

We prove the inclusion in equation (5.4) in the following way. First, we observe that  $[\varepsilon] \in \bar{V}$ , because  $\varepsilon \in cl(\varepsilon)$ . Then, we prove that  $(\bar{V}, \bar{\delta}_{\mathcal{G}})$  (again,  $\bar{\delta}_{\mathcal{G}}$  actually stands for the restriction of  $\bar{\delta}_{\mathcal{G}}$  to  $\bar{V}$ ) is a subcoalgebra of  $(\text{Exp}_{\mathcal{G}}/\equiv_{ACIE}, \bar{\delta}_{\mathcal{G}})$ . Thus,  $V \subseteq \bar{V}$ , since  $V$ , the state space of  $\langle [\varepsilon] \rangle$  is equal to the intersection of all state spaces of subcoalgebras of  $(\text{Exp}_{\mathcal{G}}/\equiv_{ACIE}, \bar{\delta}_{\mathcal{G}})$  containing  $[\varepsilon]$ .

To prove that  $(\bar{V}, \bar{\delta}_{\mathcal{G}})$  is a subcoalgebra we prove that, for  $\varepsilon_1, \dots, \varepsilon_k \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ ,

$$\varepsilon_1, \dots, \varepsilon_k \in cl(\varepsilon) \text{ all distinct} \Rightarrow \bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}}([\varepsilon_1 \oplus \dots \oplus \varepsilon_k]) \in \mathcal{F}(\bar{V}) \quad (5.5)$$

The intended result then follows by taking  $\mathcal{F} = \mathcal{G}$ .

We first prove two auxiliary results, by induction on the structure of  $\mathcal{F}$ :

- ①  $(\mathcal{F}[-])(\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}}) \in \mathcal{F}(\bar{V})$
- ②  $(\mathcal{F}[-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(u, v)) \in \mathcal{F}(\bar{V}) \Leftrightarrow (\mathcal{F}[-])(u) \in \mathcal{F}(\bar{V}) \text{ and } (\mathcal{F}[-])(v) \in \mathcal{F}(\bar{V})$

for  $u, v \in \mathcal{F}(\text{Exp}_{\mathcal{G}})$ .

$\boxed{\mathcal{F} = \text{Id}}$

- ①  $(\mathcal{F}[-])(\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}}) = [\emptyset] \in \bar{V}$
- ②  $(\mathcal{F}[-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(u, v)) = [u \oplus v] \in \bar{V} \Leftrightarrow [u] \in \bar{V} \text{ and } [v] \in \bar{V} \quad u, v \in \text{Exp}_{\mathcal{G}}$

The right to left implication follows because, using the (*Associativity*), (*Commutativity*) and (*Idempotency*) axioms, we can rewrite  $u \oplus v$  as  $\varepsilon_1 \oplus \dots \oplus \varepsilon_k$ , with all  $\varepsilon_1, \dots, \varepsilon_k \in cl(\varepsilon)$  distinct.

$\boxed{\mathcal{F} = \text{B}}$

- ①  $(\text{B}[-])(\text{Empty}_{\text{B} \triangleleft \mathcal{G}}) = \perp_{\text{B}} \in \text{B}(\bar{V})$
- ②  $(\text{B}[-])(\text{Plus}_{\text{B} \triangleleft \mathcal{G}}(u, v)) = u \vee_{\text{B}} v \in \text{B}(\bar{V}) \Leftrightarrow u \in \text{B}(\bar{V}) \text{ and } v \in \text{B}(\bar{V}) \quad u, v \in \text{B}(\text{Exp}_{\mathcal{G}})$

$$\boxed{\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2}$$

- ①  $(\mathcal{F}_1 \times \mathcal{F}_2[-])(\text{Empty}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}})$   
 $= \langle (\mathcal{F}_1[-])(\text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}), (\mathcal{F}_2[-])(\text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}}) \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\bar{\mathcal{V}})$
- ②  $(\mathcal{F}_1 \times \mathcal{F}_2[-])(\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle)) =$   
 $\langle (\mathcal{F}_1[-])(\text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(u_1, v_1)), (\mathcal{F}_2[-])(\text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(u_2, v_2)) \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\bar{\mathcal{V}})$   
 $\stackrel{(IH)}{\Leftrightarrow} u_1, v_1 \in \mathcal{F}_1(\bar{\mathcal{V}}) \text{ and } u_2, v_2 \in \mathcal{F}_2(\bar{\mathcal{V}})$   
 $\Leftrightarrow \langle u, v \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\bar{\mathcal{V}}), \quad u = \langle u_1, u_2 \rangle, v = \langle v_1, v_2 \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\text{Exp}_{\mathcal{G}})$

$$\boxed{\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2} \text{ and } \boxed{\mathcal{F} = \mathcal{F}_1^A}: \text{ similar to } \mathcal{F}_1 \times \mathcal{F}_2.$$

$$\boxed{\mathcal{F} = \mathcal{P}_{\omega} \mathcal{F}_1}$$

- ①  $(\mathcal{P}_{\omega} \mathcal{F}[-])(\text{Empty}_{\mathcal{P}_{\omega} \mathcal{F} \triangleleft \mathcal{G}}) = \emptyset \in \mathcal{P}_{\omega} \mathcal{F}(\bar{\mathcal{V}})$
- ②  $(\mathcal{P}_{\omega} \mathcal{F}[-])(\text{Plus}_{\mathcal{P}_{\omega} \mathcal{F} \triangleleft \mathcal{G}}(u, v)) = ((\mathcal{P}_{\omega} \mathcal{F}[-])(u) \cup (\mathcal{P}_{\omega} \mathcal{F}[-])(v)) \in \mathcal{P}_{\omega} \mathcal{F}(\bar{\mathcal{V}})$   
 $\Leftrightarrow (\mathcal{P}_{\omega} \mathcal{F}[-])(u) \in \mathcal{P}_{\omega} \mathcal{F}(\bar{\mathcal{V}}) \text{ and } (\mathcal{P}_{\omega} \mathcal{F}[-])(v) \in \mathcal{P}_{\omega} \mathcal{F}(\bar{\mathcal{V}})$

Using ②, we can simplify our proof goal (equation (5.5)) as follows:

$$\bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}}([\varepsilon_1 \oplus \dots \oplus \varepsilon_k]) \in \mathcal{F}(\bar{\mathcal{V}}) \Leftrightarrow (\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_i)) \in \mathcal{F}(\bar{\mathcal{V}}), \quad \varepsilon_i \in cl(\varepsilon), \quad i = 1, \dots, k$$

Using induction on the product of types of expressions and expressions (using the order defined in equation (5.1)), ① and ②, we prove that  $(\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_i)) \in \mathcal{F}(\bar{\mathcal{V}})$ , for any  $\varepsilon_i \in cl(\varepsilon)$ .

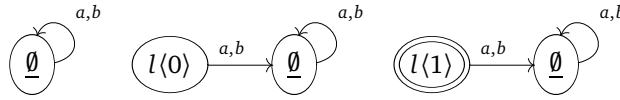
$$\begin{aligned} (\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\emptyset)) &= (\mathcal{F}[-])(\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}}) \in \mathcal{F}(\bar{\mathcal{V}}) && \text{(by ①)} \\ (\mathcal{F}[-])(\bar{\delta}_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1 \oplus \varepsilon_2)) &= (\mathcal{F}[-])(\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1), \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2))) \in \mathcal{F}(\bar{\mathcal{V}}) && \text{(IH and ②)} \\ (\mathcal{G}[-])(\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mu x. \varepsilon)) &= (\mathcal{G}[-])(\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon[\mu x. \varepsilon/x])) \in \mathcal{G}(\bar{\mathcal{V}}) && \text{(IH)} \\ (\text{Id}[-])(\delta_{\text{Id} \triangleleft \mathcal{G}}(\varepsilon_i)) &= [\varepsilon_i] \in \text{Id}(\bar{\mathcal{V}}) \text{ for } \mathcal{G} \neq \text{Id} && (\varepsilon_i \in cl(\varepsilon)) \\ (\text{B}[-])(\delta_{\text{B} \triangleleft \mathcal{G}}(b)) &= b \in \text{B}(\bar{\mathcal{V}}) && (\text{B}(\bar{\mathcal{V}}) = \text{B}) \\ (\mathcal{F}_1 \times \mathcal{F}_2[-])(\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\varepsilon))) &= \langle (\mathcal{F}_1[-])(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon)), (\mathcal{F}_2[-])(\text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}}) \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\bar{\mathcal{V}}) && \text{(IH and ①)} \\ (\mathcal{F}_1 \times \mathcal{F}_2[-])(\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(r(\varepsilon))) &= \langle (\mathcal{F}_1[-])(\text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}), (\mathcal{F}_2[-])(\delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon)) \rangle \in \mathcal{F}_1 \times \mathcal{F}_2(\bar{\mathcal{V}}) && \text{(IH and ①)} \\ (\mathcal{F}_1 \oplus \mathcal{F}_2[-])(\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon])) &= \kappa_1((\mathcal{F}_1[-])(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon))) \in \mathcal{F}_1 \oplus \mathcal{F}_2(\bar{\mathcal{V}}) && \text{(IH)} \\ (\mathcal{F}_1 \oplus \mathcal{F}_2[-])(\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(r[\varepsilon])) &= \kappa_2((\mathcal{F}_2[-])(\delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon))) \in \mathcal{F}_1 \oplus \mathcal{F}_2(\bar{\mathcal{V}}) && \text{(IH)} \\ (\mathcal{F}^A[-])(\delta_{\mathcal{F}^A \triangleleft \mathcal{G}}(a(\varepsilon))) &= \left( \lambda a'. \begin{cases} (\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)) & \text{if } a = a' \\ \text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}} & \text{otherwise} \end{cases} \right) \in \mathcal{F}^A(\bar{\mathcal{V}}) && \text{(IH and ①)} \\ (\mathcal{P}_{\omega} \mathcal{F}[-])(\delta_{\mathcal{P}_{\omega} \mathcal{F} \triangleleft \mathcal{G}}(\{\varepsilon\})) &= \{ (\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)) \} \in \mathcal{P}_{\omega} \mathcal{F}(\bar{\mathcal{V}}) && \text{(IH)} \end{aligned}$$

□

### Examples

Next we will illustrate the construction described in the proof of Theorem 5.2.14: given an expression  $\varepsilon \in \text{Exp}_{\mathcal{G}}$  we construct a  $\mathcal{G}$ -coalgebra  $(S, g)$  such that there is  $s \in S$  with  $s \sim \varepsilon$ . For simplicity, we will consider deterministic and partial automata expressions over  $A = \{a, b\}$ .

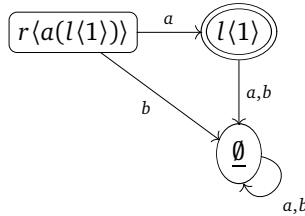
Let us start by showing the synthesized automata for the most simple deterministic expressions –  $\underline{\emptyset}$ ,  $l\langle 0 \rangle$  and  $l\langle 1 \rangle$ .



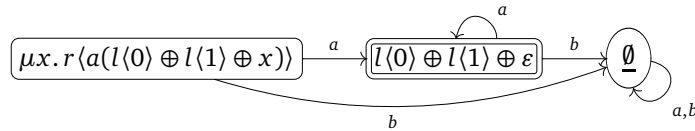
The first two automata recognize the empty language  $\emptyset$  and the last the language  $\{\varepsilon\}$  containing only the empty word.

We note that the generated automata are not minimal (for instance, the automata for  $l\langle 0 \rangle$  and  $\underline{\emptyset}$  are bisimilar). Our goal has been to generate a finite automaton from an expression. From this the minimal automaton can always be obtained by identifying bisimilar states.

The following automaton, generated from the expression  $r\langle a(l\langle 1 \rangle) \rangle$ , recognizes the language  $\{a\}$ ,

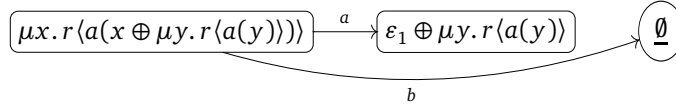


For an example of an expression containing fixed points, consider  $\varepsilon = \mu x. r\langle a(l\langle 0 \rangle \oplus l\langle 1 \rangle \oplus x) \rangle$ . One can easily compute the synthesized automaton:



and observe that it recognizes the language  $aa^*$ . Here, the role of the join-semilattice structure is also visible:  $l\langle 0 \rangle \oplus l\langle 1 \rangle \oplus \varepsilon$  specifies that this state is supposed to be non-final ( $l\langle 0 \rangle$ ) and final ( $l\langle 1 \rangle$ ). The conflict of these two specifications is solved, when they are combined with  $\oplus$ , using the join-semilattice structure: because  $1 \vee 0 = 1$  the state is set to be final.

As a last example of deterministic expressions consider  $\varepsilon_1 = \mu x. r(a(x \oplus \mu y. r(a(y))))$ . Applying  $\delta_{\mathcal{D}}$  to  $\varepsilon_1$  one gets the following (partial) automaton:

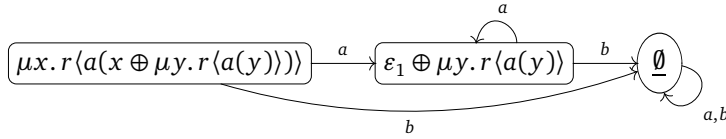


Calculating  $\delta_{\mathcal{D}}(\varepsilon_1 \oplus \mu y. r(a(y)))$  yields

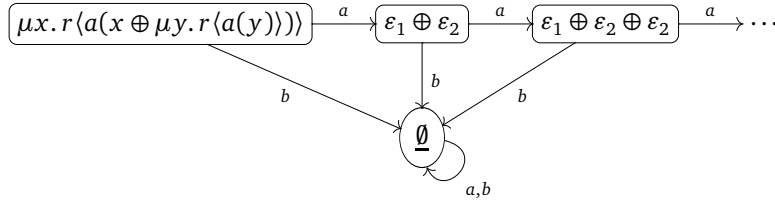
$$\delta_{\mathcal{D}}(\varepsilon_1 \oplus \mu y. r(a(y))) = \langle 0, t \rangle$$

where  $t(a) = \varepsilon_1 \oplus \mu y. r(a(y)) \oplus \mu y. r(a(y))$   
 $t(b) = \emptyset$

Note that the expression  $\varepsilon_1 \oplus \mu y. r(a(y)) \oplus \mu y. r(a(y))$  is in the same equivalence class as  $\varepsilon_1 \oplus \mu y. r(a(y))$ , which is a state that already exists. As we saw in the beginning of Section 5.2.1, by only applying  $\delta_{\mathcal{D}}$ , without *ACI*, one would always generate syntactically different states which instead of the automaton computed now:

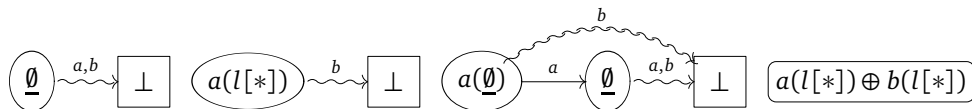


would yield the following infinite automaton (with  $\varepsilon_2 = \mu y. r(a(y))$ ):



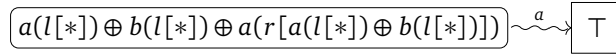
Let us next see a few examples of synthesis for partial automata expressions, where we will illustrate the role of  $\perp$  and  $\top$ . In the graphical representation of a partial automaton  $(S, p)$ , we will omit transitions for inputs  $a$  with  $g(s)(a) = \kappa_1(*)$  and we draw  $(s) \xrightarrow{a} \boxed{g(s)(a)}$  whenever  $g(s)(a) \in \{\perp, \top\}$ . Note however that  $\perp \notin S$  and  $\top \notin S$  and thus will have no defined transitions.

As before, let us first present the corresponding automata for simple expressions –  $\emptyset$ ,  $a(l[*])$ ,  $a(\emptyset)$  and  $a(l[*]) \oplus b(l[*])$ .



Note how  $\perp$  is used to encode underspecification, working as a kind of deadlock state. In the first three expressions the behavior for one or both of the inputs is missing, whereas in the last expression the specification is complete.

The element  $\top$  is used to deal with inconsistent specifications. For instance, consider the expression  $a(l[*]) \oplus b(l[*]) \oplus a(r[a(l[*]) \oplus b(l[*])])$ . All inputs are specified, but note that at the outermost level input  $a$  appears in two different sub-expressions –  $a(l[*])$  and  $a(r[a(l[*]) \oplus b(l[*])])$  – specifying at the same time that input  $a$  leads to successful termination and that it leads to a state where  $a(l[*]) \oplus b(l[*])$  holds, which is contradictory, giving rise to the following automaton.



### 5.3 A sound and complete axiomatization

In the previous section, we have shown how to derive from the type of a system, given by a functor  $\mathcal{G}$ , a language  $\text{Exp}_{\mathcal{G}}$  that allows for the specification of  $\mathcal{G}$ -behaviors. Analogously to Kleene's theorem, we have proved the correspondence between the behaviors denoted by  $\text{Exp}_{\mathcal{G}}$  and locally finite  $\mathcal{G}$ -coalgebras. In this section, we will show how to provide  $\text{Exp}_{\mathcal{G}}$  with a sound and complete axiomatization. Again, the functor  $\mathcal{G}$  will serve as a main guide for the definition. The defined axiomatization is closely related to Kleene algebra (the set of expressions has a join semilattice structure) and to the axiomatization provided by Milner for CCS (uniqueness of fixed points will be required). When instantiating the definition below to concrete functors one will recover known axiomatizations, such as the one for CCS mentioned above or the one for labeled transition systems (with explicit termination) presented in [2]. The latter will be discussed in detail in Section 5.4.

We now introduce an equational system for expressions of type  $\mathcal{F} \triangleleft \mathcal{G}$ . We define the relation  $\equiv \subseteq \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \times \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ , written infix, as the least equivalence relation containing the following identities:

1.  $(\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}, \oplus, \emptyset)$  is a join-semilattice.

$$\begin{array}{ll} \varepsilon \oplus \varepsilon \equiv \varepsilon & (\text{Idempotency}) \\ \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1 & (\text{Commutativity}) \\ \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 & (\text{Associativity}) \\ \emptyset \oplus \varepsilon \equiv \varepsilon & (\text{Empty}) \end{array}$$

2.  $\mu$  is the unique fixed point.

$$\begin{array}{ll} \gamma[\mu x. \gamma/x] \equiv \mu x. \gamma & (\text{FP}) \\ \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon & (\text{Unique}) \end{array}$$



3. The join-semilattice structure propagates through the expressions.

$$\begin{array}{llllll}
\emptyset & \equiv \perp_{\mathbb{B}} & (\mathbb{B} - \emptyset) & b_1 \oplus b_2 & \equiv & b_1 \vee_{\mathbb{B}} b_2 & (\mathbb{B} - \oplus) \\
l\langle \emptyset \rangle & \equiv \emptyset & (\times - \emptyset - L) & l\langle \varepsilon_1 \oplus \varepsilon_2 \rangle & \equiv & l\langle \varepsilon_1 \rangle \oplus l\langle \varepsilon_2 \rangle & (\times - \oplus - L) \\
r\langle \emptyset \rangle & \equiv \emptyset & (\times - \emptyset - R) & r\langle \varepsilon_1 \oplus \varepsilon_2 \rangle & \equiv & r\langle \varepsilon_1 \rangle \oplus r\langle \varepsilon_2 \rangle & (\times - \oplus - R) \\
a(\emptyset) & \equiv \emptyset & (-^A - \emptyset) & a(\varepsilon_1 \oplus \varepsilon_2) & \equiv & a(\varepsilon_1) \oplus a(\varepsilon_2) & (-^A - \oplus) \\
& & & l[\varepsilon_1 \oplus \varepsilon_2] & \equiv & l[\varepsilon_1] \oplus l[\varepsilon_2] & (+ - \oplus - L) \\
& & & r[\varepsilon_1 \oplus \varepsilon_2] & \equiv & r[\varepsilon_1] \oplus r[\varepsilon_2] & (+ - \oplus - R) \\
& & & l[\varepsilon_1] \oplus r[\varepsilon_2] & \equiv & l[\emptyset] \oplus r[\emptyset] & (+ - \oplus - \tau)
\end{array}$$

4.  $\equiv$  is a congruence.

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x] \quad \text{for } x \text{ free in } \varepsilon \quad (\text{Cong})$$

5.  $\alpha$ -equivalence

$$\mu x. \gamma \equiv \mu y. \gamma[y/x] \quad \text{for } y \text{ not free in } \gamma \quad (\alpha - \text{equiv})$$

It is important to remark that in the third group of rules there does not exist any equation applicable to expressions of type  $\mathcal{P}_{\omega}\mathcal{F}$ .

**5.3.1 EXAMPLE.** Consider the non-deterministic automata over the alphabet  $A = \{a\}$ :



Applying  $\langle\langle - \rangle\rangle$  (as defined in the proof of Theorem 5.2.12) one can easily compute the expressions corresponding to  $s_1$  and  $s_2$ :

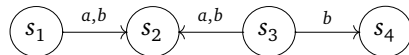
$$\begin{aligned}
\varepsilon_1 &= \langle\langle s_1 \rangle\rangle = \mu x_1. l\langle 0 \rangle \oplus r\langle a(\{x_1\}) \rangle \\
\varepsilon_2 &= \langle\langle s_2 \rangle\rangle = \mu y_1. l\langle 0 \rangle \oplus r\langle a(\{\mu y_2. l\langle 0 \rangle \oplus r\langle a(\{y_2\}) \rangle\}) \rangle
\end{aligned}$$

We prove that  $\varepsilon_2 \equiv \varepsilon_1$ . In the following calculations let  $\varepsilon = \mu x_1. r\langle a(\{x_1\}) \rangle$ .

$$\begin{aligned}
\varepsilon_2 &\equiv \varepsilon_1 \\
\Leftrightarrow & r\langle a(\{\mu y_2. r\langle a(\{r\langle a(\{y_2\}) \rangle\}) \rangle\}) \rangle \equiv \varepsilon && ((\mathbb{B} - \emptyset), (\times - \emptyset - L), (FP) \text{ and } (Empty)) \\
\Leftrightarrow & \mu y_2. r\langle a(\{r\langle a(\{y_2\}) \rangle\}) \rangle \equiv \varepsilon && ((FP) \text{ on } \varepsilon \text{ and } (Cong) \text{ twice}) \\
\Leftarrow & r\langle a(\{r\langle a(\{\varepsilon\}) \rangle\}) \rangle \equiv \varepsilon && (\text{uniqueness of fixed points}) \\
\Leftarrow & r\langle a(\{\varepsilon\}) \rangle \equiv \varepsilon && (\text{fixed point axiom}) \\
\Leftarrow & \varepsilon \equiv \varepsilon && (\text{fixed point axiom})
\end{aligned}$$

Note that the *(Cong)* rule was used in almost every step.

For another example, consider the non-deterministic automaton over the alphabet  $A = \{a, b\}$ :



Using the definition of  $\langle\langle - \rangle\rangle$  one can compute the following expressions for  $s_1, s_2, s_3$  and  $s_4$ :

$$\begin{aligned}\varepsilon_1 &= \langle\langle s_1 \rangle\rangle = \mu x_1. l(0) \oplus r(a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\})) \\ \varepsilon_2 &= \langle\langle s_2 \rangle\rangle = \mu x_2. l(0) \oplus \underline{0} \\ \varepsilon_3 &= \langle\langle s_3 \rangle\rangle = \mu x_3. l(0) \oplus r(a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\} \oplus \{\varepsilon_4\})) \\ \varepsilon_4 &= \langle\langle s_4 \rangle\rangle = \mu x_4. l(0) \oplus \underline{0}\end{aligned}$$

For  $\varepsilon_2$  we calculate:

$$\begin{aligned}\varepsilon_2 &\equiv l(0) \oplus \underline{0} && (FP) \\ &\equiv l(\underline{0}) && (Empty) \text{ and } (B - \underline{0}) \\ &\equiv \underline{0} && (\times - \underline{0} - L)\end{aligned}$$

Similarly, one has that  $\varepsilon_4 \equiv \underline{0}$ . Now, we prove  $\varepsilon_1 \equiv \varepsilon_3$ :

$$\begin{aligned}\varepsilon_1 &\equiv \varepsilon_3 \\ \Leftrightarrow l(0) \oplus r(a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\})) &\equiv l(0) \oplus r(a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\} \oplus \{\varepsilon_4\})) && (FP) \\ \Leftrightarrow l(0) \oplus r(a(\{\underline{0}\}) \oplus b(\{\underline{0}\})) &\equiv l(0) \oplus r(a(\{\underline{0}\}) \oplus b(\{\underline{0}\} \oplus \{\underline{0}\})) && (\varepsilon_2 \equiv \underline{0} \equiv \varepsilon_4) \\ \Leftrightarrow l(0) \oplus r(a(\{\underline{0}\}) \oplus b(\{\underline{0}\})) &\equiv l(0) \oplus r(a(\{\underline{0}\}) \oplus b(\{\underline{0}\})) && (Idempotency)\end{aligned}$$

♠

The equivalence relation  $\equiv$  gives rise to the (surjective) equivalence map

$$[-]: \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \rightarrow \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv$$

defined by  $[\varepsilon] = \{\varepsilon' \mid \varepsilon \equiv \varepsilon'\}$ . The following diagram summarizes the maps we have defined so far:

$$\begin{array}{ccc} \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} & \xrightarrow{[-]} & \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv \\ \delta_{\mathcal{F} \triangleleft \mathcal{G}} \downarrow & & \\ \mathcal{F}(\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}) & \xrightarrow{\mathcal{F}([-])} & \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv) \end{array}$$

In order to complete the diagram, we next prove that  $\equiv$  is contained in the kernel of  $\mathcal{F}([-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{G}}$ . This will, by Theorem 2.2.7, guarantee the existence of a well-defined function

$$\partial_{\mathcal{F} \triangleleft \mathcal{G}}: \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv)$$

which, when  $\mathcal{F} = \mathcal{G}$ , provides  $\text{Exp}_{\mathcal{G}} / \equiv$  with a coalgebraic structure  $\partial_{\mathcal{G}}: \text{Exp}_{\mathcal{G}} / \equiv \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}} / \equiv)$  (as before, we write  $\partial_{\mathcal{G}}$  to abbreviate  $\partial_{\mathcal{G} \triangleleft \mathcal{G}}$ ) and which makes  $[-]$  a homomorphism of coalgebras.

**5.3.2 LEMMA.** *Let  $\mathcal{G}$  and  $\mathcal{F}$  be non-deterministic functors, with  $\mathcal{F} \triangleleft \mathcal{G}$ . For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$  with  $\varepsilon_1 \equiv \varepsilon_2$ ,*

$$\mathcal{F}([-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1) = \mathcal{F}([-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2)$$

PROOF. By induction on the length of derivations of  $\equiv$ .

First, let us consider derivations of length 1. We need to prove the result for all the axioms in items 1. and 3. plus the axioms *FP* and ( $\alpha$  - *equiv*).

For the axioms in 1. the result follows by Lemma 5.2.13. The axiom *FP* follows trivially because of the definition of  $\delta_{\mathcal{G}}$ , since  $\delta_{\mathcal{G}}(\mu x.\gamma) = \delta_{\mathcal{G}}(\gamma[\mu x.\gamma/x])$  and thus  $\mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\mu x.\gamma) = \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[\mu x.\gamma/x])$ .

For the axiom ( $\alpha$  - *equiv*) we use the (*Cong*) rule, which is proved below:

$$\begin{aligned}
& \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\mu x.\gamma) \\
&= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[\mu x.\gamma/x]) && \text{(def. of } \delta_{\mathcal{G}}) \\
&= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[\mu y.\gamma[y/x]/x]) && \text{(by (Cong))} \\
&= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[y/x][\mu y.\gamma[y/x]/y]) && (A[B[y/x]/x] = A[y/x][B[y/x]/y], y \text{ not free in } \gamma) \\
&= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\mu y.\gamma[y/x]) && \text{(def. of } \mathcal{G}([-]) \circ \delta_{\mathcal{G}})
\end{aligned}$$

Let us show the proof for some of the axioms in 3.. The omitted cases are similar. We show for each axiom  $\varepsilon_1 \equiv \varepsilon_2$  that  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1) = \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2)$ .

$$\boxed{\perp_{\mathcal{B}} \equiv \emptyset}$$

$$\delta_{\mathcal{B} \triangleleft \mathcal{G}}(\perp_{\mathcal{B}}) = \perp_{\mathcal{B}} = \delta_{\mathcal{B} \triangleleft \mathcal{G}}(\emptyset)$$

$$\boxed{b_1 \oplus b_2 \equiv b_1 \vee_{\mathcal{B}} b_2}$$

$$\delta_{\mathcal{B} \triangleleft \mathcal{G}}(b_1 \vee_{\mathcal{B}} b_2) = b_1 \vee_{\mathcal{B}} b_2 = \delta_{\mathcal{B} \triangleleft \mathcal{G}}(b_1 \oplus b_2)$$

$$\boxed{l(\emptyset) \equiv \emptyset}$$

$$\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\emptyset)) = \langle \text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}, \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle = \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(\emptyset)$$

$$\boxed{l(\varepsilon_1 \oplus \varepsilon_2) \equiv l(\varepsilon_1) \oplus l(\varepsilon_2)}$$

$$\begin{aligned}
& \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\varepsilon_1 \oplus \varepsilon_2)) \\
&= \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1 \oplus \varepsilon_2), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle \\
&= \langle \text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1), \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_2)), \text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}}, \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}}) \rangle \\
&= \text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2}(\langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle, \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_2), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle) \\
&= \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l(\varepsilon_1) \oplus l(\varepsilon_2))
\end{aligned}$$

$$\boxed{l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2]}$$

$$\begin{aligned}
& \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon_1 \oplus \varepsilon_2]) \\
&= \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1 \oplus \varepsilon_2)) \\
&= \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2}(\kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1)), \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_2))) \\
&= \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon_1] \oplus l[\varepsilon_2])
\end{aligned}$$

$$\boxed{l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset]}$$

$$\begin{aligned}
& \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon_1] \oplus r[\varepsilon_2]) \\
&= \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2}(\kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon_1)), \kappa_2(\delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\varepsilon_2))) \\
&= \top \\
&= \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2}(\kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\emptyset)), \kappa_2(\delta_{\mathcal{F}_2 \triangleleft \mathcal{G}}(\emptyset))) \\
&= \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\emptyset] \oplus r[\emptyset])
\end{aligned}$$

Note that if we would have the axioms  $l[\emptyset] \equiv \emptyset$  and  $r[\emptyset] \equiv \emptyset$  in the axiomatization

presented above, this theorem would not hold.

$$\begin{aligned}\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\underline{\emptyset}]) &= \kappa_1([\perp]) \neq \perp = \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\underline{\emptyset}) \\ \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(r[\underline{\emptyset}]) &= \kappa_2([\perp]) \neq \perp = \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(\underline{\emptyset})\end{aligned}$$

Derivations with length  $k > 1$  can be obtained by two rules: (*Unique*) or (*Cong*). For the first (which uses the second), suppose that we have derived  $\mu x.\gamma \equiv \varepsilon$  and that we have already proved  $\gamma[\varepsilon/x] \equiv \varepsilon$ . Then, we have:

$$\begin{aligned}\mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\mu x.\gamma) &= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[\mu x.\gamma/x]) \quad (\text{def. } \delta_{\mathcal{G}}) \\ &= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\gamma[\varepsilon/x]) \quad (\text{by } (\text{Cong})) \\ &= \mathcal{G}([-]) \circ \delta_{\mathcal{G}}(\varepsilon) \quad (\text{induction hypothesis})\end{aligned}$$

For (*Cong*), suppose that we have derived  $\varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x_2]$  and that we have already derived  $\varepsilon_1 \equiv \varepsilon_2$ , which gives us, as induction hypothesis, the equality

$$(\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_1)) = (\mathcal{F}[-])(\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon_2)) \quad (5.6)$$

This equation is precisely what we need to prove the case  $\varepsilon = x$  (and thus  $\varepsilon_1, \varepsilon_2 : \mathcal{G} \triangleleft \mathcal{G}$ ):

$$\begin{aligned}(\mathcal{G}[-])(\delta_{\mathcal{G}}(x[\varepsilon_1/x])) &= (\mathcal{G}[-])(\delta_{\mathcal{G}}(\varepsilon_1)) \\ &= (\mathcal{G}[-])(\delta_{\mathcal{G}}(\varepsilon_2)) \quad (5.6) \\ &= (\mathcal{G}[-])(\delta_{\mathcal{G}}(x[\varepsilon_2/x]))\end{aligned}$$

For the cases  $\varepsilon \neq x$ , we prove that  $\delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_1/x]) = \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_2/x])$ , by induction on the product of types of expressions and expressions (using the order defined in equation (5.1)). We show a few cases, the omitted ones are similar.

$$\begin{aligned}\delta_{\mathcal{G} \triangleleft \mathcal{G}}((\mu y.\varepsilon)[\varepsilon_1/x]) &= \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_1/x][\mu y.\varepsilon/y]) \\ &\stackrel{(IH)}{=} \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_2/x][\mu y.\varepsilon/y]) = \delta_{\mathcal{G} \triangleleft \mathcal{G}}((\mu y.\varepsilon)[\varepsilon_2/x]) \\ \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l\langle \varepsilon \rangle[\varepsilon_1/x]) &= \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_1/x]), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle \\ &\stackrel{(IH)}{=} \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_2/x]), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}} \rangle = \delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}(l\langle \varepsilon \rangle[\varepsilon_2/x]) \\ \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon][\varepsilon_1/x]) &= \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_1/x])) \\ &\stackrel{(IH)}{=} \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{G}}(\varepsilon[\varepsilon_2/x])) = \delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}(l[\varepsilon][\varepsilon_2/x])\end{aligned}$$

□

Thus, we have a well-defined function  $\partial_{\mathcal{F} \triangleleft \mathcal{G}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} / \equiv \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{G}} / \equiv)$ , which makes the above diagram commute, that is  $\partial_{\mathcal{F} \triangleleft \mathcal{G}}([\varepsilon]) = (\mathcal{F}[-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{G}}(\varepsilon)$ . This provides the set  $\text{Exp}_{\mathcal{G}} / \equiv$  with a coalgebraic structure  $\partial_{\mathcal{G}} : \text{Exp}_{\mathcal{G}} / \equiv \rightarrow \mathcal{G}(\text{Exp}_{\mathcal{G}} / \equiv)$  which makes  $[-]$  a homomorphism between the coalgebras  $(\text{Exp}_{\mathcal{G}}, \delta_{\mathcal{G}})$  and  $(\text{Exp}_{\mathcal{G}} / \equiv, \partial_{\mathcal{G}})$ .

### Soundness and completeness

Next we show that the axiomatization introduced in the previous section is sound and complete.

Soundness is a direct consequence of the fact that the equivalence map  $[-]$  is a coalgebra homomorphism.

**5.3.3 THEOREM (Soundness).** *Let  $\mathcal{G}$  be a non-deterministic functor. For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{G}}$ ,*

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon_1 \sim \varepsilon_2$$

PROOF. Let  $\mathcal{G}$  be a non-deterministic functor, let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{G}}$  and suppose that  $\varepsilon_1 \equiv \varepsilon_2$ . Then,  $[\varepsilon_1] = [\varepsilon_2]$  and, thus

$$\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_1]) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_2])$$

where  $\mathbf{beh}_{\mathcal{G}}$  denotes, for any  $\mathcal{G}$ -coalgebra  $(S, g)$ , the unique map into the final coalgebra. The uniqueness of the map into the final coalgebra and the fact that  $[-]$  is a coalgebra homomorphism implies that  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv} \circ [-] = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}}$  which then yields

$$\mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_2)$$

Since in the final coalgebra only the bisimilar elements are identified,  $\varepsilon_1 \sim \varepsilon_2$  follows.  $\square$

For completeness a bit more of work is required. Let us explain upfront the key steps of the proof. The goal is to prove that  $\varepsilon_1 \sim \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$ . First, note that we have

$$\varepsilon_1 \sim \varepsilon_2 \Leftrightarrow \mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_2) \Leftrightarrow \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_1]) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_2]) \quad (5.7)$$

We then prove that  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}$  is injective, which is a sufficient condition to guarantee that  $\varepsilon_1 \equiv \varepsilon_2$  (since it implies, together with (5.7), that  $[\varepsilon_1] = [\varepsilon_2]$ ).

We proceed as follows. First, we factorize  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}$  into an epimorphism followed by a monomorphism (Theorem 2.2.8) as shown in the following diagram (where  $I = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}(\text{Exp}_{\mathcal{G}}/\equiv)$ ):

$$\begin{array}{ccccc} & & \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv} & & \\ & & \text{---} & & \\ \text{Exp}_{\mathcal{G}}/\equiv & \xrightarrow{e} & I & \xrightarrow{m} & \Omega_{\mathcal{G}} \\ & \searrow \partial_{\mathcal{G}} & \downarrow \overline{\omega}_{\mathcal{G}} & & \downarrow \omega_{\mathcal{G}} \\ \mathcal{G}(\text{Exp}_{\mathcal{G}}/\equiv) & \longrightarrow & \mathcal{G}(I) & \longrightarrow & \mathcal{G}(\Omega_{\mathcal{G}}) \end{array}$$

Then, we prove that (1)  $(\text{Exp}_{\mathcal{G}}/\equiv, \partial_{\mathcal{G}})$  is a locally finite coalgebra (Lemma 5.3.4) and (2) both coalgebras  $(\text{Exp}_{\mathcal{G}}/\equiv, \partial_{\mathcal{G}})$  and  $(I, \overline{\omega}_{\mathcal{G}})$  are final in the category of locally finite  $\mathcal{G}$ -coalgebras (Lemmas 5.3.7 and 5.3.8, respectively). Since final coalgebras are unique up to isomorphism, it follows that  $e: \text{Exp}_{\mathcal{G}}/\equiv \rightarrow I$  is in fact an isomorphism and therefore  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}$  is injective, which will give us completeness.

In the case of the deterministic automata functor  $\mathcal{D} = 2 \times \text{Id}^A$ , the set  $I$  will be precisely the set of regular languages. This means that final locally finite coalgebras generalize regular languages (in the same way that final coalgebras generalize the set of all languages).

We proceed with presenting and proving the extra lemmas needed in order to prove completeness. We start by showing that the coalgebra  $(\text{Exp}_{\mathcal{G}}/\equiv, \partial_{\mathcal{G}})$  is locally finite (note that this implies that  $(I, \overline{\omega}_{\mathcal{G}})$  is also locally finite) and that  $\partial_{\mathcal{G}}$  is an isomorphism.

**5.3.4 LEMMA.** *The coalgebra  $(\text{Exp}_{\mathcal{G}}/\equiv, \partial_{\mathcal{G}})$  is a locally finite coalgebra. Moreover,  $\partial_{\mathcal{G}}$  is an isomorphism.*

PROOF. Local finiteness is a direct consequence of the generalized Kleene's theorem (Theorem 5.2.14). In the proof of Theorem 5.2.14 we showed that, given  $\varepsilon \in \text{Exp}_{\mathcal{G}}$ , the subcoalgebra  $\langle [\varepsilon]_{ACIE} \rangle$  is finite. Thus, the subcoalgebra  $\langle [\varepsilon] \rangle$  is also finite (since  $\text{Exp}_{\mathcal{G}}/\equiv$  is a quotient of  $\text{Exp}_{\mathcal{G}}/\equiv_{ACIE}$ ).

To see that  $\partial_{\mathcal{G}}$  is an isomorphism, first define, for every  $\mathcal{F} \triangleleft \mathcal{G}$ ,

$$\partial_{\mathcal{F} \triangleleft \mathcal{G}}^{-1}(c) = [\bar{\gamma}_c^{\mathcal{F}}] \quad (5.8)$$

where  $\bar{\gamma}_c^{\mathcal{F}}$  is defined, for  $\mathcal{F} \neq \text{Id}$ , as  $\gamma_c^{\mathcal{F}}$  in the proof of Theorem 5.2.12, and for  $\mathcal{F} = \text{Id}$  as  $\bar{\gamma}_{[\varepsilon]}^{\text{Id}} = \varepsilon$ . Then, we prove that  $\partial_{\mathcal{F} \triangleleft \mathcal{G}}^{-1}$  has indeed the properties ①  $\partial_{\mathcal{F} \triangleleft \mathcal{G}}^{-1} \circ \partial_{\mathcal{F} \triangleleft \mathcal{G}} = \text{id}_{\text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}/\equiv}$  and ②  $\partial_{\mathcal{F} \triangleleft \mathcal{G}} \circ \partial_{\mathcal{F} \triangleleft \mathcal{G}}^{-1} = \text{id}_{\langle [\varepsilon] \rangle}$ . Instantiating  $\mathcal{F} = \mathcal{G}$  one derives that  $\delta_{\mathcal{G}}$  is an isomorphism. It is enough to prove for ① that  $\bar{\gamma}_{\partial_{\mathcal{F} \triangleleft \mathcal{G}}^{-1}([\varepsilon])}^{\mathcal{F}} \equiv \varepsilon$  and for ② that  $\partial_{\mathcal{F} \triangleleft \mathcal{G}}([\bar{\gamma}_c^{\mathcal{F}}]) = c$ . We illustrate a few cases. The omitted ones are similar.

① By induction on the product of types of expressions and expressions (using the order defined in equation (5.1)).

$$\begin{aligned} \bar{\gamma}_{\partial_{\text{Id} \triangleleft \mathcal{G}}^{-1}([\varepsilon])}^{\text{Id}} &= \varepsilon \\ \bar{\gamma}_{\partial_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}^{-1}([r(\varepsilon)])}^{\mathcal{F}_1 \times \mathcal{F}_2} &= l \langle \bar{\gamma}_{\partial_{\mathcal{F}_1 \triangleleft \mathcal{G}}^{-1}(\emptyset)}^{\mathcal{F}_1} \rangle \oplus r \langle \bar{\gamma}_{\partial_{\mathcal{F}_2 \triangleleft \mathcal{G}}^{-1}(\varepsilon)}^{\mathcal{F}_2} \rangle \stackrel{(IH)}{\equiv} l \langle \emptyset \rangle \oplus r \langle \varepsilon \rangle \equiv r \langle \varepsilon \rangle \\ \bar{\gamma}_{\partial_{\mathcal{G}}^{-1}([\mu x. \varepsilon])}^{\mathcal{G}} &= \bar{\gamma}_{\partial_{\mathcal{G}}^{-1}([\varepsilon[\mu x. \varepsilon/x]])}^{\mathcal{G}} \stackrel{(IH)}{\equiv} \varepsilon[\mu x. \varepsilon/x] \equiv \mu x. \varepsilon \end{aligned}$$

Note that the cases  $\varepsilon = \emptyset$  and  $\varepsilon = \varepsilon_1 \oplus \varepsilon_2$  require an extra proof (by induction on  $\mathcal{F}$ ). More precisely, one needs to prove that

$$\textcircled{a} \bar{\gamma}_{\mathcal{F}[-](\text{Empty}_{\mathcal{F} \triangleleft \mathcal{G}})}^{\mathcal{F}} \equiv \emptyset \text{ and } \textcircled{b} \bar{\gamma}_{\mathcal{F}[-](\text{Plus}_{\mathcal{F} \triangleleft \mathcal{G}}(x_1, x_2))}^{\mathcal{F}} \equiv \bar{\gamma}_{\mathcal{F}[-](x_1)}^{\mathcal{F}} \oplus \bar{\gamma}_{\mathcal{F}[-](x_2)}^{\mathcal{F}}$$

It is an easy proof by induction. We illustrate here only the cases  $\mathcal{F} = \text{Id}$ ,  $\mathcal{F} = \text{B}$  and

$\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2$ .

$$\begin{aligned}
\textcircled{a} \quad & \overline{\gamma}_{[\emptyset]}^{\text{Id}} = \underline{\emptyset} \\
& \overline{\gamma}_{[\perp_B]}^B = \perp_B \equiv \underline{\emptyset} \\
& \overline{\gamma}_{(\mathcal{F}_1[-](\text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}}), \mathcal{F}_2[-](\text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}}))}^{\mathcal{F}_1 \times \mathcal{F}_2} = l \langle \overline{\gamma}_{\mathcal{F}_1[-](\text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{G}})}^{\mathcal{F}_1} \rangle \oplus r \langle \overline{\gamma}_{\mathcal{F}_2[-](\text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{G}})}^{\mathcal{F}_2} \rangle \\
& \stackrel{(IH)}{\equiv} l \langle \underline{\emptyset} \rangle \oplus r \langle \underline{\emptyset} \rangle \equiv \underline{\emptyset} \\
\textcircled{b} \quad & \overline{\gamma}_{[x_1 \oplus x_2]}^{\text{Id}} = x_1 \oplus x_2 = \overline{\gamma}_{[x_1]}^{\text{Id}} \oplus \overline{\gamma}_{[x_2]}^{\text{Id}} \\
& \overline{\gamma}_{[x_1 \vee_B x_2]}^B = x_1 \vee_B x_2 \equiv x_1 \oplus x_2 = \overline{\gamma}_{[x_1]}^B \oplus \overline{\gamma}_{[x_2]}^B \\
& \overline{\gamma}_{\mathcal{F}_1 \times \mathcal{F}_2}^{\mathcal{F}_1 \times \mathcal{F}_2} = \overline{\gamma}_{\mathcal{F}_1 \times \mathcal{F}_2}^{\text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}((u_1, v_1), (u_2, v_2))} \\
& = \overline{\gamma}_{(\text{Plus}_{\mathcal{F}_1}(u_1, v_1), \text{Plus}_{\mathcal{F}_2}(u_2, v_2))}^{\mathcal{F}_1 \times \mathcal{F}_2} \\
& = l \langle \overline{\gamma}_{\text{Plus}_{\mathcal{F}_1}(u_1, v_1)}^{\mathcal{F}_1} \rangle \oplus r \langle \overline{\gamma}_{\text{Plus}_{\mathcal{F}_2}(u_2, v_2)}^{\mathcal{F}_2} \rangle \\
& \stackrel{(IH)}{\equiv} l \langle \gamma_{u_1}^{\mathcal{F}_1} \oplus \gamma_{v_1}^{\mathcal{F}_1} \rangle \oplus r \langle \gamma_{u_2}^{\mathcal{F}_2} \oplus \gamma_{v_2}^{\mathcal{F}_2} \rangle \\
& \equiv (l \langle \gamma_{u_1}^{\mathcal{F}_1} \rangle \oplus r \langle \gamma_{u_2}^{\mathcal{F}_2} \rangle) \oplus (l \langle \gamma_{v_1}^{\mathcal{F}_1} \rangle \oplus r \langle \gamma_{v_2}^{\mathcal{F}_2} \rangle) \\
& = \overline{\gamma}_{(u_1, u_2)}^{\mathcal{F}_1 \times \mathcal{F}_2} \oplus \overline{\gamma}_{(v_1, v_2)}^{\mathcal{F}_1 \times \mathcal{F}_2}
\end{aligned}$$

② By induction on the structure of  $\mathcal{F}$ .

$$\begin{aligned}
\partial_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}([\overline{\gamma}_c^{\mathcal{F}_1 \oplus \mathcal{F}_2}]) &= \begin{cases} \partial_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}([l[\overline{\gamma}_{c'}^{\mathcal{F}_1}]]]) = \kappa_1(\partial_{\mathcal{F}_1 \triangleleft \mathcal{G}}([\overline{\gamma}_{c'}^{\mathcal{F}_1}])) & c = \kappa_1(c') \\ \partial_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}([r[\overline{\gamma}_{c'}^{\mathcal{F}_2}]]]) = \kappa_2(\partial_{\mathcal{F}_2 \triangleleft \mathcal{G}}([\overline{\gamma}_{c'}^{\mathcal{F}_2}])) & c = \kappa_2(c') \\ \partial_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}([\underline{\emptyset}]) = \perp & c = \perp \\ \partial_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{G}}([l[\underline{\emptyset}] \oplus r[\underline{\emptyset}]]) = \top & c = \top \end{cases} \\
& \stackrel{(IH)}{\equiv} c \\
\partial_{\mathcal{P}_\omega \mathcal{F} \triangleleft \mathcal{G}}([\overline{\gamma}_c^{\mathcal{P}_\omega \mathcal{F}}]) &= \begin{cases} \partial_{\mathcal{P}_\omega \mathcal{F} \triangleleft \mathcal{G}}([\underline{\emptyset}]) = \emptyset & C = \emptyset \\ \partial_{\mathcal{P}_\omega \mathcal{F} \triangleleft \mathcal{G}}([\bigoplus_{c \in C} \overline{\gamma}_c^{\mathcal{F}_1}]) = \{\partial_{\mathcal{F} \triangleleft \mathcal{G}}([\overline{\gamma}_c^{\mathcal{F}_1}]) \mid c \in C\} & \text{otherwise} \end{cases} \\
& \stackrel{(IH)}{\equiv} C
\end{aligned}$$

□

Next, we prove the analogue of the following useful and intuitive equality on regular expressions (which we showed in Theorem 3.1.14). Given a deterministic automaton  $\langle o, t \rangle : S \rightarrow 2 \times S^A$  and a state  $s \in S$ , the associated regular expression  $r_s$  can be written as

$$r_s = o(s) + \sum_{a \in A} a \cdot r_{t(s)(a)} \quad (5.9)$$

using the axioms of Kleene algebra [29, Theorem 4.4].

**5.3.5 LEMMA.** *Let  $(S, g)$  be a locally finite  $\mathcal{G}$ -coalgebra, with  $\mathcal{G} \neq \text{Id}$ , and let  $s \in S$ , with  $\langle s \rangle = \{s_1, \dots, s_n\}$  (where  $s_1 = s$ ). Then:*

$$\langle\langle s_i \rangle\rangle \equiv \gamma_{g(s_i)}^{\mathcal{G}} \{ \langle\langle s_1 \rangle\rangle / x_1 \} \dots \{ \langle\langle s_n \rangle\rangle / x_n \} \quad (5.10)$$

PROOF. Let  $A_i^k$ , where  $i$  and  $k$  range from 1 to  $n$ , be the terms defined as in the proof of Theorem 5.2.12. Recall that  $\langle\langle s_i \rangle\rangle = A_i^n$ . We calculate:

$$\begin{aligned} & \langle\langle s_i \rangle\rangle \\ &= A_i^n \\ &= (\mu x_i. \gamma_{g(s_i)}^{\mathcal{G}}) \{ A_1^0 / x_1 \} \dots \{ A_n^{n-1} / x_n \} \\ &= \mu x_i. (\gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^0 / x_1 \} \dots \{ A_{i-1}^{i-2} / x_{i-2} \} \{ A_{i+1}^i / x_{i+1} \} \dots \{ A_n^{n-1} / x_n \}) \\ &\equiv \gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^0 / x_1 \} \dots \{ A_{i-1}^{i-2} / x_{i-2} \} \{ A_{i+1}^i / x_{i+1} \} \dots \{ A_n^{n-1} / x_n \} \{ A_i^n / x_i \} \quad (\text{fixed point axiom}^3) \\ &= \gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^0 / x_1 \} \dots \{ A_n^{n-1} / x_n \} \quad (\text{by 5.2}) \\ &= \gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^0 \{ A_2^1 / x_2 \} \dots \{ A_n^{n-1} / x_n \} / x_1 \} \dots \{ A_n^{n-1} / x_n \} \quad (\text{by 5.3}) \\ &= \gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^n / x_1 \} \{ A_2^1 / x_2 \} \dots \{ A_n^{n-1} / x_n \} \quad (\text{def. } A_1^n) \\ &\vdots \quad (\text{last 2 steps for } A_2^1, \dots, A_{n-1}^{n-2}) \\ &= \gamma_{g(s_i)}^{\mathcal{G}} \{ A_1^n / x_1 \} \{ A_2^n / x_2 \} \dots \{ A_n^n / x_n \} \quad (A_{n-1}^n = A_n^n) \end{aligned}$$

□

Instantiating (5.10) for  $\langle o, t \rangle: S \rightarrow 2 \times S^A$ , one can easily spot the similarity with equation (5.9) above:

$$\langle\langle s \rangle\rangle \equiv l \langle o(s) \rangle \oplus r \left\langle \bigoplus_{a \in A} a (\langle\langle t(s)(a) \rangle\rangle) \right\rangle$$

Next, we prove that there exists a coalgebra homomorphism between any locally finite  $\mathcal{G}$ -coalgebra  $(S, g)$  and  $(\text{Exp}_{\mathcal{G}} / \equiv, \partial_{\mathcal{G}})$ .

**5.3.6 LEMMA.** *Let  $(S, g)$  be a locally finite  $\mathcal{G}$ -coalgebra. There exists a coalgebra homomorphism  $\lceil - \rceil: S \rightarrow \text{Exp}_{\mathcal{G}} / \equiv$ .*

PROOF. We define  $\lceil - \rceil = [-] \circ \langle\langle - \rangle\rangle$ , where  $\langle\langle - \rangle\rangle$  is as in the proof of Theorem 5.2.12, associating to a state  $s$  of a locally finite coalgebra an expression  $\langle\langle s \rangle\rangle$  with  $s \sim \langle\langle s \rangle\rangle$ . To prove that  $\lceil - \rceil$  is a homomorphism we need to verify that  $(\mathcal{G} \lceil - \rceil) \circ g = \partial_{\mathcal{G}} \circ \lceil - \rceil$ . If  $\mathcal{G} = \text{Id}$ , then  $(\mathcal{G} \lceil - \rceil) \circ g(s_i) = [\emptyset] = \partial_{\mathcal{G}}(\lceil s_i \rceil)$ . For  $\mathcal{G} \neq \text{Id}$  we calculate, using Lemma 5.3.5:

$$\partial_{\mathcal{G}}(\lceil s_i \rceil) = \partial_{\mathcal{G}}([\gamma_{g(s_i)}^{\mathcal{G}} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n]])$$

and we then prove the more general equality, for  $\mathcal{F} \triangleleft \mathcal{G}$  and  $c \in \mathcal{F}(s)$ :

$$\partial_{\mathcal{F} \triangleleft \mathcal{G}}([\gamma_c^{\mathcal{F}} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n]]) = \mathcal{F} \lceil - \rceil(c) \quad (5.11)$$

<sup>3</sup>Note that the fixed point axiom can be formulated using syntactic replacement rather than substitution  $-\gamma\{\mu x. \gamma / x\} \equiv \mu x. \gamma$  – since  $\mu x. \gamma$  is a closed term.



The intended equality then follows by taking  $\mathcal{F} = \mathcal{G}$  and  $c = g(s_i)$ . Let us prove the equation (5.11) by induction on  $\mathcal{F}$ .

$$\boxed{\mathcal{F} = \text{ld}} \quad c = s_j \in \langle s \rangle$$

$$\partial_{\text{d} \triangleleft \mathcal{G}}([\gamma_{s_j}^{\text{ld}}[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]) = [\langle s_j \rangle] = \lceil s_j \rceil$$

$$\boxed{\mathcal{F} = \text{B}} \quad c = b \in \text{B}$$

$$\partial_{\text{B} \triangleleft \mathcal{G}}([\gamma_b^{\text{B}}[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]) = [b] = \text{B}[\lceil - \rceil(b)]$$

$$\boxed{\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2} \quad c = \langle c_1, c_2 \rangle \in (\mathcal{F}_1 \times \mathcal{F}_2)\langle s \rangle$$

$$\begin{aligned} & \partial_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}([\gamma_{\langle c_1, c_2 \rangle}^{\mathcal{F}_1 \times \mathcal{F}_2}[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]) \\ &= \partial_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{G}}([\iota(\gamma_{c_1}^{\mathcal{F}_1}) \oplus r(\gamma_{c_2}^{\mathcal{F}_2})[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]) \\ &= \langle \partial_{\mathcal{F}_1 \triangleleft \mathcal{G}}([\gamma_{c_1}^{\mathcal{F}_1}[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]), \partial_{\mathcal{F}_2 \triangleleft \mathcal{G}}([\gamma_{c_2}^{\mathcal{F}_2}[\langle s_1 \rangle/x_1] \dots [\langle s_n \rangle/x_n]]) \rangle \\ &\stackrel{\text{IH}}{=} \langle \mathcal{F}_1[\lceil - \rceil(c_1)], \mathcal{F}_2[\lceil - \rceil(c_2)] \rangle \\ &= (\mathcal{F}_1 \times \mathcal{F}_2[\lceil - \rceil])(c) \end{aligned}$$

$$\boxed{\mathcal{F} = \mathcal{F}_1 \diamond \mathcal{F}_2}, \quad \boxed{\mathcal{F} = \mathcal{F}_1^A} \quad \text{and} \quad \boxed{\mathcal{F} = \mathcal{P}_\omega \mathcal{F}_1} : \text{similar to } \mathcal{F}_1 \times \mathcal{F}_2.$$

□

We can now prove that the coalgebras  $(\text{Exp}_{\mathcal{G}/\equiv}, \partial_{\mathcal{G}})$  and  $(I, \overline{\omega}_{\mathcal{G}})$  are both final in the category of locally finite  $\mathcal{G}$ -coalgebras.

**5.3.7 LEMMA.** *The coalgebra  $(I, \overline{\omega}_{\mathcal{G}})$  is final in the category  $\text{Coalg}(\mathcal{G})_{\text{LF}}$ .*

PROOF. We want to show that for any locally finite  $\mathcal{G}$ -coalgebra  $(S, g)$ , there exists a *unique* homomorphism  $(S, g) \rightarrow (I, \overline{\omega}_{\mathcal{G}})$ . The existence is guaranteed by Lemma 5.3.6, where  $\lceil - \rceil : S \rightarrow \text{Exp}_{\mathcal{G}/\equiv}$  is defined. Postcomposing this homomorphism with  $e$  (defined above) we get a coalgebra homomorphism  $e \circ \lceil - \rceil : S \rightarrow I$ . If there is another homomorphism  $f : S \rightarrow I$ , then by postcomposition with the inclusion  $m : I \hookrightarrow \Omega$  we get two homomorphisms  $(m \circ f$  and  $m \circ e \circ \lceil - \rceil)$  into the final  $\mathcal{G}$ -coalgebra. Thus,  $f$  and  $e \circ \lceil - \rceil$  must be equal. □

**5.3.8 LEMMA.** *The coalgebra  $(\text{Exp}_{\mathcal{G}/\equiv}, \partial_{\mathcal{G}})$  is final in the category  $\text{Coalg}(\mathcal{G})_{\text{LF}}$ .*

PROOF. We want to show that for any locally finite  $\mathcal{G}$ -coalgebra  $(S, g)$ , there exists a *unique* homomorphism  $(S, g) \rightarrow (\text{Exp}_{\mathcal{G}/\equiv}, \partial_{\mathcal{G}})$ . We only need to prove uniqueness, since the existence is guaranteed by Lemma 5.3.6, where  $\lceil - \rceil : S \rightarrow \text{Exp}_{\mathcal{G}/\equiv}$  is defined. Suppose we have another homomorphism  $f : S \rightarrow \text{Exp}_{\mathcal{G}/\equiv}$ . Then, we shall prove that  $f = \lceil - \rceil$ . Let, for any  $s \in S$ ,  $f_s$  denote any representative of  $f(s)$  (that is,  $f(s) = [f_s]$ ). First, observe that because  $f$  is a homomorphism the following holds for every  $s \in S$ :

$$f(s) = (\partial_{\mathcal{G}}^{-1} \circ \mathcal{G}(f) \circ g)(s) \Leftrightarrow f_s \equiv \gamma_{g(s)}^{\mathcal{G}}[f_{s_1}/x_1] \dots [f_{s_n}/x_n] \quad (5.12)$$

where  $\langle s \rangle = \{s_1, \dots, s_n\}$ , with  $s_1 = s$  (recall that  $\partial_{\mathcal{G}}^{-1}$  was defined in (5.8) and note that  $\overline{\gamma}_{(\mathcal{G}(f) \circ \mathcal{G})(s)}^{\mathcal{G}} = \gamma_{g(s)}^{\mathcal{G}}[f_{s_i}/x_i]$ ).

We now prove that  $f_{s_i} \equiv \langle\langle s_i \rangle\rangle$  (which is equivalent to  $f(s_i) = \lceil s_i \rceil$ ), for all  $i = 1, \dots, n$ . For simplicity we will here prove the case  $n = 3$ . The general case is identical but notationally heavier. First, we prove that  $f_{s_1} \equiv A_1[f_{s_2}/x_2][f_{s_3}/x_3]$ .

$$\begin{aligned} f_{s_1} &\equiv \gamma_{g(s_1)}^{\mathcal{G}}[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (5.12))} \\ \Leftrightarrow f_{s_1} &\equiv \gamma_{g(s_1)}^{\mathcal{G}}[f_{s_2}/x_2][f_{s_3}/x_3][f_{s_1}/x_1] && \text{(all } f(s_i) \text{ are closed)} \\ \Rightarrow f_{s_1} &\equiv \mu x_1. \gamma_{g(s_1)}^{\mathcal{G}}[f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by uniqueness of fixed points)} \\ \Leftrightarrow f_{s_1} &\equiv A_1[f_{s_2}/x_2][f_{s_3}/x_3] && \text{(def. of } A_1) \end{aligned}$$

Now, using what we have computed for  $f_{s_1}$  we prove that  $f_{s_2} \equiv A_2^1[f_{s_3}/x_3]$ .

$$\begin{aligned} f_{s_2} &\equiv \gamma_{g(s_2)}^{\mathcal{G}}[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (5.12))} \\ \Leftrightarrow f_{s_2} &\equiv \gamma_{g(s_2)}^{\mathcal{G}}[A_1/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(expressions for } f_{s_1} \text{ and (5.3))} \\ \Leftrightarrow f_{s_2} &\equiv \gamma_{g(s_2)}^{\mathcal{G}}[A_1/x_1][f_{s_3}/x_3][f_{s_2}/x_2] && \text{(all } f(s_i) \text{ are closed)} \\ \Rightarrow f_{s_2} &\equiv \mu x_2. \gamma_{g(s_2)}^{\mathcal{G}}[A_1/x_1][f_{s_3}/x_3] && \text{(by uniqueness of fixed points)} \\ \Leftrightarrow f_{s_2} &\equiv A_2^1[f_{s_3}/x_3] && \text{(def. of } A_2^1) \end{aligned}$$

At this point we substitute  $f_{s_2}$  in the expression for  $f_{s_1}$  by  $A_2^1[f_{s_3}/x_3]$  which yields:

$$f_{s_1} \equiv A_1[A_2^1[f_{s_3}/x_3]/x_2][f_{s_3}/x_3] \equiv A_1[A_2^1/x_2][f_{s_3}/x_3]$$

Finally, we prove that  $f_{s_3} \equiv A_3^2$ :

$$\begin{aligned} f_{s_3} &\equiv \gamma_{g(s_3)}^{\mathcal{G}}[f_{s_1}/x_1][f_{s_2}/x_2][f_{s_3}/x_3] && \text{(by (5.12))} \\ \Leftrightarrow f_{s_3} &\equiv \gamma_{g(s_3)}^{\mathcal{G}}[A_1/x_1][A_2^1/x_2][f_{s_3}/x_3] && \text{(expr. for } f(s_i) \text{ and (5.3))} \\ \Rightarrow f_{s_3} &\equiv \mu x_3. \gamma_{g(s_3)}^{\mathcal{G}}[A_1/x_1][A_2^1/x_2] && \text{(by uniqueness of fixed points)} \\ \Leftrightarrow f_{s_3} &\equiv A_3^2 && \text{(def. of } A_3^2) \end{aligned}$$

Thus, we have  $f_{s_1} \equiv A_1[A_2^1/x_2][A_3^2/x_3]$ ,  $f_{s_2} \equiv A_2^1[A_3^2/x_3]$  and  $f_{s_3} \equiv A_3^2$ . Observe that  $A_2^1[A_3^2/x_3] \equiv A_2^1\{A_3^2/x_3\}$ , since  $x_2$  is not free in  $A_3^2$ . Similarly, since  $x_1$  is not free in  $A_2^1$  and  $A_3^2$ , we have that  $A_1[A_2^1/x_2][A_3^2/x_3] \equiv A_1\{A_2^1/x_2\}\{A_3^2/x_3\}$ . Thus  $f(s_i) = \lceil s_i \rceil$ , for all  $i = 1, 2, 3$ .  $\square$

As a consequence of Lemma 5.3.8, we have that if  $\mathcal{G}_1$  and  $\mathcal{G}_2$  are isomorphic functors then  $\text{Exp}_{\mathcal{G}_1/\equiv}$  and  $\text{Exp}_{\mathcal{G}_2/\equiv}$  are also isomorphic (for instance, this would be true for  $\mathcal{G}_1(X) = B \times (X \times A)$  and  $\mathcal{G}_2(X) = A \times (B \times X)$ ). At this point, because final objects are unique up-to isomorphism, we know that  $e: \text{Exp}_{\mathcal{G}/\equiv} \rightarrow I$  is an isomorphism and hence we can conclude that the map  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}/\equiv}}$  is injective, since it factorizes into an isomorphism followed by a mono. This fact is the last thing we need to prove completeness.

**5.3.9 THEOREM (Completeness).** *Let  $\mathcal{G}$  be a non-deterministic functor. For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{G}}$ ,*

$$\varepsilon_1 \sim \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$$

PROOF. Let  $\mathcal{G}$  be a non-deterministic functor, let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{G}}$  and suppose that  $\varepsilon_1 \sim \varepsilon_2$ . Because only bisimilar elements are identified in the final coalgebra we know that it must be the case that  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}}(\varepsilon_2)$  and thus, since the equivalence class map  $[-]$  is a homomorphism,  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_1]) = \mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}([\varepsilon_2])$ . Now, because  $\mathbf{beh}_{\text{Exp}_{\mathcal{G}}/\equiv}$  is injective we have that  $[\varepsilon_1] = [\varepsilon_2]$ . Hence,  $\varepsilon_1 \equiv \varepsilon_2$ .  $\square$

## 5.4 Two more examples

In this section we apply our framework to two other examples: labeled transition systems (with explicit termination) and automata on guarded strings. These two automata models are directly connected to, respectively, basic process algebra and Kleene algebra with tests. To improve readability we will present the corresponding languages using a more user-friendly syntax than the canonically derived one.

**Labeled transition systems.** Labeled transition systems (with explicit termination) are coalgebras for the functor  $1 + (\mathcal{P}_{\omega}\text{Id})^A$ . As we will show below, instantiating our framework for this functor produces a language that is equivalent to the closed and guarded expressions generated by the following grammar, where  $a \in A$  and  $x \in X$  ( $X$  is a set of fixed point variables):

$$P ::= \mathbf{0} \mid P + P \mid a.P \mid \delta \mid \surd \mid \mu x.P \mid x$$

together with the equations (omitting the congruence and  $\alpha$ -equivalence rules)

$$\begin{array}{ll} P_1 + P_2 \equiv P_2 + P_1 & P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3 \\ P + P \equiv P & P + \mathbf{0} \equiv P \\ P + \delta \equiv P \quad (\star) & \surd + \delta \equiv \surd + P \quad (\star) \quad (\star) \text{ if } P \not\equiv \mathbf{0} \text{ and } P \not\equiv \surd \\ P[\mu x.P/x] \equiv \mu x.P & P[Q/x] \equiv Q \Rightarrow (\mu x.P) \equiv Q \end{array}$$

Note that, as expected, there is no law that allows us to prove  $a.(P + Q) \equiv a.P + a.Q$ . Moreover, observe that this syntax and axiomatization is very similar to the one presented in [2]. In the syntax above,  $\delta$  represents deadlock,  $\surd$  successful termination and  $\mathbf{0}$  the totally undefined process.

We will next show how the beautified syntax above was derived from the canonically derived syntax for the expressions  $\varepsilon \in \text{Exp}_{1+(\mathcal{P}_{\omega}\text{Id})^A}$ , which is given by the set of closed and guarded expressions defined by the following BNF:

$$\begin{array}{l} \varepsilon ::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid l[\varepsilon_1] \mid r[\varepsilon_2] \\ \varepsilon_1 ::= \underline{\emptyset} \mid \varepsilon_1 \oplus \varepsilon_1 \mid * \\ \varepsilon_2 ::= \underline{\emptyset} \mid \varepsilon_2 \oplus \varepsilon_2 \mid a(\varepsilon') \\ \varepsilon' ::= \underline{\emptyset} \mid \varepsilon' \oplus \varepsilon' \mid \{\varepsilon\} \end{array}$$

We define two maps between this grammar and the grammar presented above. Let us start to show how to translate  $P$ 's into  $\varepsilon$ 's, by defining a map  $(-)^{\dagger}$  by induction on the structure of  $P$ :

$$\begin{array}{ll} (\mathbf{0})^{\dagger} &= \underline{\emptyset} & (a.P)^{\dagger} &= r[a(\{P^{\dagger}\})] \\ (P_1 + P_2)^{\dagger} &= (P_1)^{\dagger} \oplus (P_2)^{\dagger} & (\surd)^{\dagger} &= l[*] \\ (\mu x.P)^{\dagger} &= \mu x.P^{\dagger} & (\delta)^{\dagger} &= r[\underline{\emptyset}] \\ x^{\dagger} &= x \end{array}$$

And now the converse translation:

$$\begin{array}{ll} (\underline{\emptyset})^{\ddagger} &= \mathbf{0} & (l[*])^{\ddagger} &= \surd \\ (\varepsilon_1 \oplus \varepsilon_2)^{\ddagger} &= (\varepsilon_1)^{\ddagger} + (\varepsilon_2)^{\ddagger} & (r[\underline{\emptyset}])^{\ddagger} &= \delta \\ (\mu x.\varepsilon)^{\ddagger} &= \mu x.\varepsilon^{\ddagger} & (r[\varepsilon_2 \oplus \varepsilon_2'])^{\ddagger} &= (r[\varepsilon_2])^{\ddagger} + (r[\varepsilon_2'])^{\ddagger} \\ x^{\ddagger} &= x & (r[a(\underline{\emptyset})])^{\ddagger} &= \delta \\ (l[\underline{\emptyset}])^{\ddagger} &= \surd & (r[a(\varepsilon_1' \oplus \varepsilon_2')])^{\ddagger} &= (r[a(\varepsilon_1')])^{\ddagger} + (r[a(\varepsilon_2')])^{\ddagger} \\ (l[\varepsilon_1 \oplus \varepsilon_1'])^{\ddagger} &= (l[\varepsilon_1])^{\ddagger} + (l[\varepsilon_1'])^{\ddagger} & (r[a(\{\varepsilon\})])^{\ddagger} &= a.\varepsilon^{\ddagger} \end{array}$$

One can prove that if  $P_1 \equiv P_2$  (using the equations above) then  $(P_1)^{\dagger} \equiv (P_2)^{\dagger}$  (using the automatically derived equations for the functor) and also that  $\varepsilon_1 \equiv \varepsilon_2$  implies  $(\varepsilon_1)^{\ddagger} \equiv (\varepsilon_2)^{\ddagger}$ .

**Automata on guarded strings.** It has recently been shown [71] that automata on guarded strings (acceptors of the join irreducible elements of the free Kleene algebra with tests on generators  $\Sigma, T$ ) are coalgebras for the functor  $\mathbf{B} \times \text{Id}^{At \times \Sigma}$ , where  $At$  is the set of atoms, *i.e.* minimal nonzero elements of the free Boolean algebra  $\mathbf{B}$  generated by  $T$  and  $\Sigma$  is a set of actions. Applying our framework to this functor yields a language that is equivalent to the closed and guarded expressions generated by the following grammar, where  $b \in \mathbf{B}$  and  $a \in \Sigma$ :

$$P ::= \mathbf{0} \mid \langle b \rangle \mid P + P \mid b \rightarrow a.P \mid \mu x.P \mid x$$

accompanied by the equations (omitting the congruence and  $\alpha$ -equivalence rules)

$$\begin{array}{ll} P_1 + P_2 \equiv P_2 + P_1 & P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3 \\ P + P \equiv P & P + \mathbf{0} \equiv P \\ \langle b_1 \rangle + \langle b_2 \rangle \equiv \langle b_1 \vee_{\mathbf{B}} b_2 \rangle & \mathbf{0} \equiv \langle \perp_{\mathbf{B}} \rangle \\ (b \rightarrow a.\mathbf{0}) \equiv \mathbf{0} & (\perp_{\mathbf{B}} \rightarrow a.P) \equiv \mathbf{0} \\ (b \rightarrow a.P_2) + (b \rightarrow a.P_2) \equiv b \rightarrow a.(P_1 + P_2) & (b_1 \rightarrow a.P) + (b_2 \rightarrow a.P) \equiv (b_1 \vee_{\mathbf{B}} b_2) \rightarrow a.P \\ P[\mu x.P/x] \equiv \mu x.P & P[Q/x] \equiv Q \Rightarrow (\mu x.P) \equiv Q \end{array}$$

We will not present a full comparison of this syntax to the one of Kleene algebra with tests [71] (and propositional Hoare triples). The differences between our syntax and that of KAT are similar to the ones between regular expressions and the language  $\text{Exp}_{\mathcal{D}}$  for the functor representing deterministic automata (see Definition 5.2.5). Similarly to the LTS example one can define maps between the beautified syntax and the automatically generated one and prove its correctness.

## 5.5 Polynomial and finitary coalgebras

The functors we considered above allowed us to modularly derive languages and axiomatizations for a large class of coalgebras. If we consider the subset of *NDF* without the  $\mathcal{P}_\omega$  functor, the class of coalgebras for these functors almost coincides with polynomial coalgebras (that is, coalgebras for a polynomial functor). The only difference comes from the use of join-semilattices for constant functors and  $\oplus$  instead of the ordinary coproduct, which played an important role in order for us to be able to have underspecification and overspecification. We will now show how to derive expressions and axiomatizations directly for polynomial coalgebras, where no underspecification or overspecification is allowed.

Before we show the formal definition, let us provide some intuition. The main changes<sup>4</sup>, compared to the previous sections, would be not to have  $\emptyset$  and  $\oplus$  and consider an expression  $\langle -, - \rangle$  for the product instead of the two expressions  $l\langle - \rangle$  and  $r\langle - \rangle$  which we considered and an expression  $\langle a_1(-), a_2(-), \dots, a_n(-) \rangle$  for the exponential (with  $A = \{a_1, \dots, a_n\}$ ). As an example, take the functor  $\mathcal{D}(X) = 2 \times X^A$  of deterministic automata. The expressions corresponding to this functor would then be the set of closed and guarded expressions given by the following BNF:

$$\varepsilon ::= x \mid \mu x. \varepsilon \mid \langle 0, \langle a_1(\varepsilon), a_2(\varepsilon), \dots, a_n(\varepsilon) \rangle \rangle \mid \langle 1, \langle a_1(\varepsilon), a_2(\varepsilon), \dots, a_n(\varepsilon) \rangle \rangle$$

This syntax can be perceived as an explicit and complete description of the automaton. This means that underspecification is nonexistent and the compactness of regular expressions is lost. As an example of the verbosity present in this new language, take  $A = \{a, b, c\}$  and consider the language that accepts words with only  $a$ 's and has at last one  $a$  (described by  $aa^*$  in Kleene's regular expressions). In the language  $\text{Exp}_{\mathcal{D}}$  it would be written as  $\mu x. a(l\langle 1 \rangle \oplus x)$ . Using the approach described above it would be encoded as the expression

$$\mu x. \langle 0, \langle a(\langle 1, \langle a(x), b(\text{empty}), c(\text{empty}) \rangle \rangle), b(\text{empty}), c(\text{empty}) \rangle \rangle$$

where  $\text{empty} = \mu y. \langle 0, \langle a(y), b(y), c(y) \rangle \rangle$  is the expression denoting the empty language. The approach we presented before, by allowing underspecification, provides a more user-friendly syntax and stays close to the know syntaxes for deterministic automata and LTS.

In what follows we will formally present a language for polynomial coalgebras. We start by introducing the definition of polynomial functor, which we take from [4].

**5.5.1 DEFINITION (Polynomial Functor).** Sums of the Cartesian power functors are called polynomial functors:

$$\mathbf{P}_\Sigma(X) = \coprod_{\sigma \in \Sigma} X^{ar(\sigma)}$$

Here,  $\coprod$  stands for ordinary coproduct and the indexing set  $\Sigma$  is a signature, that is a possibly infinite collection of symbols  $\sigma$ , each of which is equipped with a finite cardinal  $ar(\sigma)$ , called the arity of  $\sigma$ . ♣

<sup>4</sup>This syntax was suggested to us by B. Klin, during CONCUR'09.

**5.5.2 DEFINITION** (Expressions and axioms for polynomial functors). Let  $\mathbf{P}_\Sigma$  be a polynomial functor. The set  $\text{Exp}_{\mathbf{P}_\Sigma}$  of expressions for  $\mathbf{P}_\Sigma$  is given by the closed and guarded expressions generated by the following BNF, where  $\sigma \in \Sigma$  and  $x \in V$ , for  $V$  a set of fixed point variables:

$$\varepsilon_i ::= x \mid \mu x. \varepsilon \mid \sigma(\varepsilon_1, \dots, \varepsilon_{\text{ar}(\sigma)})$$

accompanied by the equations:

$$\begin{aligned} \gamma[\mu x. \gamma/x] &\equiv \mu x. \gamma && \text{(FP)} \\ \gamma[\varepsilon/x] &\equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon && \text{(Unique)} \\ \varepsilon_1 \equiv \varepsilon_2 &\Rightarrow \varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x], \quad \text{if } x \text{ is free in } \varepsilon && \text{(Cong)} \\ \mu x. \gamma &\equiv \mu y. \gamma[y/x], \quad \text{if } y \text{ is not free in } \gamma && \text{(\alpha - equiv)} \end{aligned}$$

♣

Providing the set  $\text{Exp}_{\mathbf{P}_\Sigma}$  with a coalgebraic structure is achieved using induction on the number of unguarded occurrences of nested fixed points:

$$\begin{aligned} \delta : \text{Exp}_{\mathbf{P}_\Sigma} &\rightarrow \prod_{\sigma \in \Sigma} (\text{Exp}_{\mathbf{P}_\Sigma})^{\text{ar}(\sigma)} \\ \delta(\mu x. \varepsilon) &= \delta(\varepsilon[\mu x. \varepsilon/x]) \\ \delta(\sigma(\varepsilon_1, \dots, \varepsilon_{\text{ar}(\sigma)})) &= \kappa_\sigma(\langle \varepsilon_1, \dots, \varepsilon_{\text{ar}(\sigma)} \rangle) \end{aligned}$$

We are ready to state and prove Kleene's theorem.

**5.5.3 THEOREM** (Kleene's theorem for polynomial functors). *Let  $\mathbf{P}_\Sigma$  be a polynomial functor.*

1. *For every locally finite coalgebra  $(S, g : S \rightarrow \mathbf{P}_\Sigma(S))$  and for every  $s \in S$  there exists an expression  $\varepsilon \in \text{Exp}_{\mathbf{P}_\Sigma}$  such that  $\varepsilon \sim s$ .*
2. *For every expression  $\varepsilon \in \text{Exp}_{\mathbf{P}_\Sigma}$  there is a finite coalgebra  $(S, g : S \rightarrow \mathbf{P}_\Sigma(S))$  with  $s \in S$  such that  $s \sim \varepsilon$ .*

PROOF. Point 1. amounts to solve a system of equations. Let  $\langle s \rangle = \{s_1, \dots, s_n\}$ . We associate with each  $s_i \in \langle s \rangle$  an expression  $\langle\langle s_i \rangle\rangle = A_i^n$ , where  $A_i^n$  is defined inductively as in the proof of 5.2.12, with  $A_i^{k+1} = A_i^k \{A_{k+1}^k/x_{k+1}\}$  and  $A_i^0 = A_i$  given by

$$A_i = \mu x_{s_i}. \sigma(x_{s'_1}, \dots, x_{s'_{\text{ar}(\sigma)}}), \quad g(s_i) = \kappa_\sigma(s'_1, \dots, s'_{\text{ar}(\sigma)})$$

It remains to prove that  $s_i \sim \varepsilon_i$ , for all  $s_i \in \langle s \rangle$ . We observe that

$$R = \{ \langle s_i, \langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle \}$$

is a bisimulation, since for  $g(s_i) = \kappa_\sigma(s'_1, \dots, s'_{ar(\sigma)})$ , we have

$$\begin{aligned}
& \delta(\langle\langle s_i \rangle\rangle) \\
&= \delta((\mu x_i. \sigma(x_{s'_1}, \dots, x_{s'_{ar(\sigma)}}))\{A_1^0/x_1\} \dots \{A_n^{n-1}/x_n\}) \quad (\text{def. of } \varepsilon_i) \\
&= \delta(\mu x_i. \sigma(x_{s'_1}, \dots, x_{s'_{ar(\sigma)}})\{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}) \\
&= \delta(\sigma(x_{s'_1}, \dots, x_{s'_{ar(\sigma)}})\{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}[A_i^n/x_i]) \quad (\text{def. of } \delta) \\
&= \delta(\sigma(x_{s'_1}, \dots, x_{s'_{ar(\sigma)}})\{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\}\{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}\{A_i^n/x_i\}) \\
&= \delta(\sigma(x_{s'_1}, \dots, x_{s'_{ar(\sigma)}})\{A_1^0/x_1\} \dots \{A_{i-1}^{i-2}/x_{i-1}\}\{A_i^n/x_i\}\{A_{i+1}^i/x_{i+1}\} \dots \{A_n^{n-1}/x_n\}) \\
&= \kappa_\sigma(\langle\langle s'_1 \rangle\rangle, \dots, \langle\langle s'_{ar(\sigma)} \rangle\rangle)
\end{aligned}$$

For point 2, we observe that the subcoalgebra  $\langle \varepsilon \rangle$ , for any  $\varepsilon \in \text{Exp}_{\mathbb{P}_\Sigma}$  is finite, since the set  $cl(\varepsilon)$  containing all sub-formulas and unfoldings of fixed points of  $\varepsilon$ , which is finite, is a subcoalgebra of  $(\text{Exp}_{\mathbb{P}_\Sigma}, \delta)$ . The fact that in this point, contrary to what happened in Theorem 5.2.14, we do not need to quotient the set of expressions is a direct consequence of the absence of underspecification or, more concretely, of the expressions  $\underline{\emptyset}$  and  $\oplus$ .  $\square$

The proof of soundness and completeness would follow a similar strategy as in the previous section and we will omit it here.

In order to be able to compare the language introduced in this section with the language obtained in our previous approach, we have to define an infinitary version of the operator  $\diamond$  and extend the framework accordingly. We start by defining the aforementioned operator on sets:  $\diamond_{i \in I} X_i = (\prod_{i \in I} X_i) \cup \{\perp, \top\}$  and the corresponding functor, for which we shall use the same symbol, is defined pointwise in the same way as for  $\diamond$ . Note that  $\diamond$  is a special case of this operator (resp. functor) for  $I$  a two element set. In fact, for simplicity, we shall only consider this operator for index sets  $I$  with two or more elements.

There is a natural transformation between polynomial functors and the class of non-deterministic functors extended with  $\diamond$ : every polynomial functor  $\mathbb{P}_\Sigma(X)$  is mapped to  $\overline{\mathbb{P}}_\Sigma(X) = \diamond_{\sigma \in \Sigma} X^{ar(\sigma)}$ .

Next, we slightly alter the definition of expressions. Instead of the expressions  $l[-]$  and  $r[-]$  we had before for  $\diamond$  we add an expression  $i[-]$  for each  $i \in I$  and the expected typing rule:

$$\frac{\vdash \varepsilon: \mathcal{F}_j \triangleleft \mathcal{G} \quad j \in I}{\vdash j[\varepsilon]: \diamond_{i \in I} \mathcal{F}_i \triangleleft \mathcal{G}}$$

All the other elements in our story are adjusted in the expected way. We show what happens in the axiomatization. For  $\diamond$  we had the rules

$$l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2] \quad r[\varepsilon_1 \oplus \varepsilon_2] \equiv r[\varepsilon_1] \oplus r[\varepsilon_2] \quad l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\underline{\emptyset}] \oplus r[\underline{\emptyset}]$$

which are now replaced by

$$i[\varepsilon_1] \oplus i[\varepsilon_2] \equiv i[\varepsilon_1 \oplus \varepsilon_2] \quad i[\varepsilon_1] \oplus j[\varepsilon_2] \equiv k[\underline{\emptyset}] \oplus l[\underline{\emptyset}], \quad i \neq j, k \neq l$$

It is natural to ask what is the relation between the sets of expressions  $\text{Exp}_{\mathbf{P}_\Sigma}$  and  $\text{Exp}_{\overline{\mathbf{P}}_\Sigma}$ . The set  $\text{Exp}_{\mathbf{P}_\Sigma}$  is bijective to the subset of  $\text{Exp}_{\overline{\mathbf{P}}_\Sigma}$  containing only fully specified expressions, that is expressions  $\varepsilon$  for which the subcoalgebra  $\langle \varepsilon \rangle$  does not contain any state for which  $\delta_{\overline{\mathbf{P}}_\Sigma}$  evaluates to  $\perp$  and  $\top$ . This condition is purely semantical and we were not able to find a purely syntactic restriction that would capture it.

We repeat the exercise above for finitary functors. A finitary functor  $\mathbf{F}$  is a functor that is a quotient of a polynomial functor, *i.e.* there exists a natural transformation  $\eta: \mathbf{P}_\Sigma \rightarrow \mathbf{F}$ , whose components  $\eta_X: \mathbf{P}_\Sigma(X) \rightarrow \mathbf{F}(X)$  are epimorphisms. We define  $\text{Exp}_{\mathbf{F}} = \text{Exp}_{\mathbf{P}_\Sigma}$ .

**5.5.4 THEOREM** (Kleene's theorem for finitary functors). *Let  $\mathbf{F}$  be a finitary functor.*

1. *Let  $(S, f)$  be a locally-finite  $\mathbf{F}$ -coalgebra. Then, for any  $s \in S$ , there exists an expression  $\langle\langle s \rangle\rangle \in \text{Exp}_{\mathbf{F}}$  such that  $s \sim \langle\langle s \rangle\rangle$ .*
2. *Let  $\varepsilon \in \text{Exp}_{\mathbf{F}}$ . Then, there exists a finite  $\mathbf{F}$ -coalgebra  $(S, f)$  with  $s \in S$  such that  $s \sim \varepsilon$ .*

PROOF. Let  $\mathbf{F}$  be a finitary functor (quotient of a polynomial functor  $\mathbf{P}_\Sigma$ ).

① Let  $(S, f)$  be a locally finite  $\mathbf{F}$ -coalgebra and let  $s \in S$ . We denote by  $T = \{s_1, \dots, s_n\}$  the state space of the subcoalgebra  $\langle s \rangle$  (with  $s_1 = s$ ). We then have that there exists an  $f^\sharp$  making the following diagram commute:

$$\begin{array}{ccccc} T & \xrightarrow{id} & T & \longrightarrow & S \\ f^\sharp \downarrow & & \downarrow f & & \downarrow f \\ \mathbf{P}_\Sigma(T) & \xrightarrow{\eta_s} & \mathbf{F}(T) & \longrightarrow & \mathbf{F}(S) \end{array}$$

We then build  $\langle\langle s \rangle\rangle$  with respect to  $f^\sharp$  just as in Theorem 5.2.12 (note that  $(T, f^\sharp)$  is finite) and the result follows because  $\langle\langle s \rangle\rangle \sim_{\mathbf{F}} s \leftarrow \langle\langle s \rangle\rangle \sim_{\mathbf{P}_\Sigma} s$  (consequence of naturality).

② Let  $\varepsilon \in \text{Exp}_{\mathbf{F}}$ . By Theorem 5.2.14, there exists a finite  $\mathbf{P}_\Sigma$ -coalgebra  $(S, f)$  with  $s \in S$  such that  $s \sim_{\mathbf{P}_\Sigma} \varepsilon$ . Thus, we take  $(S, \eta_S \circ f)$  and we have a finite  $\mathbf{F}$ -coalgebra with  $s \in S$  such that  $\varepsilon \sim_{\mathbf{F}} s$ .  $\square$

For the axiomatization a bit more ingenuity is required. One needs to derive which extra axioms are induced by the epimorphism and then prove that they are sound and complete.

For instance, the finite powerset functor (which we included in the syntax of non-deterministic functors) is the classical example of a finitary functor. It is the quotient of the polynomial functor  $\mathbf{P}_\Sigma(X) = 1 + X + X^2 + \dots$  (this represents lists of length  $n$ ) by identifying lists that contain precisely the same elements (that is, eliminating repeated elements and abstracting from the ordering).

The syntax for  $\text{Exp}_{\mathbf{P}_\Sigma}$  is the set of closed and guarded expressions given by the following BNF:

$$\varepsilon ::= x \mid \mu x. \varepsilon \mid i(\varepsilon_1, \dots, \varepsilon_i), \quad i \in \mathbb{N}$$

together with the axioms for the fixed point, ( $\alpha$ -equiv) and (Cong).



Taking into account the restriction mentioned we would have to include the extra axioms:

$$\begin{aligned} i(\varepsilon_1, \dots, \varepsilon_i) &\equiv i(\varepsilon'_1, \dots, \varepsilon'_i) \text{ if } \{\varepsilon_1, \dots, \varepsilon_i\} = \{\varepsilon'_1, \dots, \varepsilon'_i\} \\ i(\varepsilon_1, \varepsilon_2, \dots, \varepsilon_i) &\equiv (i-1)(\varepsilon_1, \varepsilon_3, \dots, \varepsilon_i) \text{ if } \varepsilon_1 \equiv \varepsilon_2 \end{aligned}$$

In this case, one can see that this set of axioms is sound and complete, by simply proving, for  $\mathbf{P}_\Sigma(X) = 1 + X + X^2 + \dots$ ,  $\text{Exp}_{\mathbf{P}_\Sigma}/\equiv \cong \text{Exp}_{\mathcal{P}_\omega}/\equiv$  (since we already had a language and sound and complete axiomatization for the  $\mathcal{P}_\omega$  functor). The restricted syntax and axioms needs to be derived for each concrete finitary functor. Finding a uniform way of defining such restricted syntax/axioms and also uniformly proving soundness and completeness is a challenging problem and it is left as future work.

## 5.6 Discussion

We presented a systematic way of deriving, from the type of a system, a language of (generalized) regular expressions and a sound and complete axiomatization thereof. We presented the analogue of Kleene's theorem, proving the correspondence of the behaviors captured by the expressions and the systems under consideration. The whole approach was illustrated with five examples: deterministic finite automata, partial deterministic automata, non-deterministic automata, labeled transition systems and automata on guarded strings. Moreover, all the results presented in the previous chapter for Mealy machines can be recovered as a particular instance of the present framework.

The language of generalized regular expressions we associated with each functor modulo the axioms is closely related to the work on iterative theories [5, 6]: it is an initial iterative algebra. This also shows the connection of our work with the work by Bloom and Ésik on iterative algebras/theories [20]. It would be interesting to investigate the connections with iterative algebras further.

In [61], a bialgebraic review of deterministic automata and regular expressions was presented. One of the main results of [61] was a description of the free algebra and Brzowski coalgebra structure on regular expressions as a bialgebra with respect to a GSOS law. We expect that this extends to our framework, but fully working this out is left as future work.

In this chapter, we studied coalgebras for **Set** functors. It is an important and challenging question to extend our results to other categories. Following our work, S. Milius [82] has showed how to derive a language and sound and complete axiomatization for the functor  $\mathbb{R} \times \text{Id}$  in the category of vector spaces and linear maps. It would also be interesting to extend Milius results for other functors as well as study functors over other categories, such as metric spaces [73, 114, 115].

The connection between regular expressions and coalgebras was first explored by Rutten in [95], as we discussed in the introduction of this thesis. In the present chapter, the set of expressions for the functor  $\mathcal{F}(S) = 2 \times S^A$  differs from the classical definition in that we do not have Kleene star and full concatenation (sequential composition) but, instead, the least fixed point operator and action prefixing. Modulo that difference, the definition of a coalgebra structure on the set of expressions in both [95]

and the present chapter is essentially the same. All in all, one can therefore say that standard regular expressions and their treatment in [95] can be viewed as a special instance of the present approach. This is also the case for the generalization of the results in [95] to automata on guarded strings [71]. Finally, this chapter extends the results of the previous chapter, where a sound and complete specification language was presented. All the results therein can be recovered as a special instance of the framework of this chapter by considering the functor  $(B \times \text{Id})^A$ , where  $A$  is a finite input alphabet and  $B$  is a finite semilattice for the output alphabet.

In the last few years several proposals of specification languages for coalgebras appeared [24, 25, 36, 50, 60, 75, 86, 94, 103]. We discussed in the introduction of this thesis the main similarities and differences with the existing approaches.

All the results presented in this chapter can be extended in order to accommodate systems with *quantities*, such as probability or costs [21], which we will do in the next chapter. The main technical challenge is that quantitative systems have an inherently non-idempotent behavior and thus the proof of Kleene's theorem and the axiomatization require extra care. This extension allows for the derivation of specification languages and axiomatizations for a wide variety of systems, which include weighted automata, simple probabilistic systems (also known as Markov chains) and systems with mixed probability and non-determinism (such as Segala systems). Here is when generality of our approach really brings new results. For instance, we have derived a language and an axiomatization for the so-called stratified systems. The language is equivalent to the one presented in [116], but no axiomatization was known.

The derivation of the syntax and axioms associated with each non-deterministic functor has been implemented in the coinductive prover CIRC [78]. This allows for automatic reasoning about the equivalence of expressions specifying systems.

In the previous chapter, we introduced for coalgebras of a large but restricted class of functors, a language of regular expressions; a corresponding generalization of Kleene's Theorem; and a sound and complete axiomatization with respect to bisimilarity. We derived both the language of expressions and their axiomatization, in a modular fashion, from the functor defining the type of the system, by induction on the structure of the functors.

In recent years, much attention has been devoted to the analysis of probabilistic behaviors, which occur for instance in randomized, fault-tolerant systems. Several different types of systems were proposed: reactive [76, 92], generative [48], stratified [109, 117], alternating [55, 118], (simple) Segala systems [105, 106], bundle [40] and Pnueli-Zuck [91], among others. For some of these systems, expressions were defined for the specification of their behaviors, as well as axioms to reason about their behavioral equivalence. Examples include [1, 13, 15, 41, 42, 62, 77, 85, 110].

The results of the previous chapter apply to the class of non-deterministic functors, which is general enough to include the examples of deterministic automata and labeled transition systems, as well as many other systems such as Mealy and Moore machines. However, probabilistic systems, weighted automata [43, 104] etc. *cannot* be described by non-deterministic functors. It is aim of the present chapter to identify a class of functors (a) that is general enough to include these and more generally a large class of *quantitative systems*; and (b) to which the methodology developed in the previous chapter can be extended.

To this end, we give a non-trivial extension of the class of non-deterministic functors by adding a functor type that allows the transitions of our systems to take values in a *monoid* structure of quantitative values. This new class, which we shall call quantitative functors, now includes all the types of probabilistic systems mentioned above.

At the same time, we show how to extend our earlier approach to the new setting. As it turns out, the main technical challenge is due to the fact that the behavior of quantitative systems is inherently *non-idempotent*. As an example consider the expression  $1/2 \cdot \varepsilon \oplus 1/2 \cdot \varepsilon'$  representing a probabilistic system that either behaves as  $\varepsilon$  with prob-

ability  $1/2$  or behaves as  $\varepsilon'$  with the same probability. When  $\varepsilon$  is equivalent to  $\varepsilon'$ , then the system is equivalent to  $1 \cdot \varepsilon$  rather than  $1/2 \cdot \varepsilon$ . This is problematic because idempotency played a crucial role in our previous results to ensure that expressions denote finite-state behaviors.

We will show how the lack of idempotency in the extended class of functors can be circumvented by a clever use of the monoid structure. This will allow us to derive for each functor in our new extended class everything we were after: a language of regular expressions; a corresponding Kleene's Theorem; and a sound and complete axiomatization for the corresponding notion of behavioral equivalence.

In order to show the effectiveness and the generality of our approach, we apply it to four types of systems: weighted automata; and simple Segala, stratified and Pnueli-Zuck systems. For simple Segala systems, we recover the language and axiomatization presented in [42]. For weighted automata and stratified systems, languages have been defined in [30] and [117] but, to the best of our knowledge, no axiomatization was ever given. Applying our method, we obtain the same languages and, more interestingly, we obtain novel axiomatizations. We also present a completely new framework to reason about Pnueli-Zuck systems. Table 6.1 summarizes the main results of this chapter: the language and axiomatizations derived for several quantitative systems.

**Organization of the chapter.** In Section 6.1 we introduce the functor that will allow us to model quantitative systems: the monoidal exponentiation functor. Section 6.2 shows how to extend the framework presented in the previous chapter to quantitative systems: we associate with every quantitative functor  $\mathcal{H}$  a language of expressions  $\text{Exp}_{\mathcal{H}}$ , we prove a Kleene like theorem and we introduce a sound and complete axiomatization with respect to the behavioral equivalence induced by  $\mathcal{H}$ . Section 6.3 paves the way for the derivation of expressions and axioms for probabilistic systems, which we present in Section 6.4. In Section 6.5, we present a variation on the definition of the monoidal exponentiation functor and show the advantages and disadvantages for the construction of the framework. Section 6.6 presents concluding remarks and directions for future research.

<u>Weighted automata</u> – $\mathcal{H}(S) = \mathbb{S} \times (\mathbb{S}^S)^A$	
$\varepsilon ::= \underline{0} \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \mid x \mid s \mid a(s \cdot \varepsilon)$	where $s \in \mathbb{S}$ and $a \in A$
$(\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \equiv \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3)$ $a(s \cdot \varepsilon) \oplus a(s' \cdot \varepsilon) \equiv a((s + s') \cdot \varepsilon)$ $\varepsilon[\mu x. \varepsilon / x] \equiv \mu x. \varepsilon$	$\varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$ $s \oplus s' \equiv s + s'$ $\gamma[\varepsilon / x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon$
$\varepsilon \oplus \underline{0} \equiv \varepsilon$ $a(\underline{0} \cdot \varepsilon) \equiv \underline{0}$ $0 \equiv \underline{0}$	
<u>Segala systems</u> – $\mathcal{H}(S) = \mathcal{P}_\omega(\mathcal{D}_\omega(S))^A$	
$\varepsilon ::= \underline{0} \mid \varepsilon \boxplus \varepsilon \mid \mu x. \varepsilon \mid x \mid a(\{\varepsilon'\})$	where $a \in A, p_i \in (0, 1]$ and $\sum_{i \in 1 \dots n} p_i = 1$
$\varepsilon' ::= \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i$	
$(\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 \equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3)$ $(\varepsilon'_1 \boxplus \varepsilon'_2) \boxplus \varepsilon'_3 \equiv \varepsilon'_1 \boxplus (\varepsilon'_2 \boxplus \varepsilon'_3)$ $\varepsilon[\mu x. \varepsilon / x] \equiv \mu x. \varepsilon$	$\varepsilon_1 \boxplus \varepsilon_2 \equiv \varepsilon_2 \boxplus \varepsilon_1$ $\varepsilon'_1 \boxplus \varepsilon'_2 \equiv \varepsilon'_2 \boxplus \varepsilon'_1$ $\gamma[\varepsilon / x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon$
$\varepsilon \boxplus \underline{0} \equiv \varepsilon$ $(p_1 \cdot \varepsilon) \boxplus (p_2 \cdot \varepsilon) \equiv (p_1 + p_2) \cdot \varepsilon$	
<u>Stratified systems</u> – $\mathcal{H}(S) = \mathcal{D}_\omega(S) + (\mathbb{B} \times S) + 1$	
$\varepsilon ::= \mu x. \varepsilon \mid x \mid \langle b, \varepsilon \rangle \mid \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i \mid \downarrow$	where $b \in \mathbb{B}, p_i \in (0, 1]$ and $\sum_{i \in 1 \dots n} p_i = 1$
$(\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \equiv \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3)$ $\varepsilon[\mu x. \varepsilon / x] \equiv \mu x. \varepsilon$	$\varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$ $\gamma[\varepsilon / x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon$
$(p_1 \cdot \varepsilon) \oplus (p_2 \cdot \varepsilon) \equiv (p_1 + p_2) \cdot \varepsilon$	
<u>Pnueli-Zuck systems</u> – $\mathcal{H}(S) = \mathcal{P}_\omega \mathcal{D}_\omega(\mathcal{P}_\omega(S))^A$	
$\varepsilon ::= \underline{0} \mid \varepsilon \boxplus \varepsilon \mid \mu x. \varepsilon \mid x \mid \{\varepsilon'\}$	where $a \in A, p_i \in (0, 1]$ and $\sum_{i \in 1 \dots n} p_i = 1$
$\varepsilon' ::= \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon'_i$	
$\varepsilon'' ::= \underline{0} \mid \varepsilon'' \boxplus \varepsilon'' \mid a(\{\varepsilon\})$	
$(\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 \equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3)$ $(\varepsilon'_1 \boxplus \varepsilon'_2) \boxplus \varepsilon'_3 \equiv \varepsilon'_1 \boxplus (\varepsilon'_2 \boxplus \varepsilon'_3)$ $(\varepsilon''_1 \boxplus \varepsilon''_2) \boxplus \varepsilon''_3 \equiv \varepsilon''_1 \boxplus (\varepsilon''_2 \boxplus \varepsilon''_3)$ $\varepsilon[\mu x. \varepsilon / x] \equiv \mu x. \varepsilon$	$\varepsilon_1 \boxplus \varepsilon_2 \equiv \varepsilon_2 \boxplus \varepsilon_1$ $\varepsilon'_1 \boxplus \varepsilon'_2 \equiv \varepsilon'_2 \boxplus \varepsilon'_1$ $\varepsilon''_1 \boxplus \varepsilon''_2 \equiv \varepsilon''_2 \boxplus \varepsilon''_1$ $\gamma[\varepsilon / x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon$
$\varepsilon \boxplus \underline{0} \equiv \varepsilon$ $(p_1 \cdot \varepsilon'') \boxplus (p_2 \cdot \varepsilon'') \equiv (p_1 + p_2) \cdot \varepsilon''$ $\varepsilon'' \boxplus \underline{0} \equiv \varepsilon''$ $\varepsilon'' \boxplus \varepsilon'' \equiv \varepsilon''$	

Table 6.1: All the (valid) expressions are closed and guarded. The congruence and the  $\alpha$ -equivalence axioms are implicitly assumed for all the systems. The symbols 0 and + denote, in the case of weighted automata, the empty element and the binary operator of the commutative monoid  $\mathbb{S}$  while, for the other systems, they denote the ordinary 0 and sum of real numbers. We write  $\bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i$  for  $p_1 \cdot \varepsilon_1 \oplus \dots \oplus p_n \cdot \varepsilon_n$ .

## 6.1 The monoidal exponentiation functor

In the previous chapter we introduced non-deterministic functors and a language of expressions for specifying coalgebras. Coalgebras for non-deterministic functors cover many interesting types of systems, such as deterministic automata and Mealy machines, but not quantitative systems. For this reason, we recall the definition of the *monoidal exponentiation functor* [51], which will allow us to define coalgebras representing quantitative systems. In the next section, we will provide expressions and an axiomatization for these.

A *monoid*  $\mathbb{M}$  is an algebraic structure consisting of a set with an associative binary operation  $+$  and a neutral element  $0$  for that operation. A *commutative monoid* is a monoid where  $+$  is also commutative. Examples of commutative monoids include  $\mathbf{2}$ , the two-element  $\{0, 1\}$  Boolean algebra with logical “or”, and the set  $\mathbb{R}$  of real numbers with addition (we will use  $\mathbb{R}$  to denote both the monoid and the carrier set). A property that will play a crucial role in the rest of the paper is *idempotency*: a monoid is idempotent, if the associated binary operation  $+$  is idempotent. For example, the monoid  $\mathbf{2}$  is idempotent, while  $\mathbb{R}$  is not. Note that an idempotent commutative monoid is a join-semilattice.

Given a function  $\varphi$  from a set  $S$  to a monoid  $\mathbb{M}$ , we define *support of  $\varphi$*  as the set  $\{s \in S \mid \varphi(s) \neq 0\}$ .

**6.1.1 DEFINITION (Monoidal exponentiation functor).** Let  $\mathbb{M}$  be a commutative monoid. The monoidal exponentiation functor  $\mathbb{M}_\omega^- : \mathbf{Set} \rightarrow \mathbf{Set}$  is defined as follows. For each set  $S$ ,  $\mathbb{M}_\omega^S$  is the set of functions from  $S$  to  $\mathbb{M}$  with finite support. For each function  $h : S \rightarrow T$ ,  $\mathbb{M}_\omega^h : \mathbb{M}_\omega^S \rightarrow \mathbb{M}_\omega^T$  is the function mapping each  $\varphi \in \mathbb{M}_\omega^S$  into  $\varphi^h \in \mathbb{M}_\omega^T$  defined, for every  $t \in T$ , as

$$\varphi^h(t) = \sum_{s' \in h^{-1}(t)} \varphi(s')$$

♣

Throughout this chapter we will omit the subscript  $\omega$  and use  $\mathbb{M}^-$  to denote the monoidal exponentiation functor. Note that the (finite) powerset functor  $\mathcal{P}_\omega(-)$  coincides with  $\mathbf{2}_\omega^-$ . The  $\mathcal{P}_\omega$  functor is often used to represent non-deterministic systems. For example, LTS’s (with labels over  $A$ ) are coalgebras for the functor  $\mathcal{P}_\omega(-)^A$ .

**6.1.2 PROPOSITION.** *The functor  $\mathbb{M}^-$  is bounded.*

PROOF. Using [53, Theorem 4.7] it is enough to prove that there exists a natural surjection  $\eta$  from a functor  $B \times (-)^A$  to  $\mathbb{M}^-$ , for some sets  $B$  and  $A$ .

We take  $A = \mathbb{N}$  and  $B = \mathbb{M}^\mathbb{N}$ , where  $\mathbb{N}$  denotes the set of all natural numbers and we define for every set  $X$  the function  $\eta_X : \mathbb{M}^\mathbb{N} \times X^\mathbb{N} \rightarrow \mathbb{M}^X$  as

$$\eta_X(\varphi, f)(x) = \sum_{n \in f^{-1}(x)} \varphi(n)$$

Because  $\varphi$  has finite support, the function  $\eta_X$  is surjective. It remains to prove that it is natural. Take  $h: X \rightarrow Y$ . We shall prove that the following diagram commutes

$$\begin{array}{ccc} \mathbb{M}^{\mathbb{N}} \times X^{\mathbb{N}} & \xrightarrow{id \times h^{\mathbb{N}}} & \mathbb{M}^{\mathbb{N}} \times Y^{\mathbb{N}} \\ \eta_X \downarrow & & \downarrow \eta_Y \\ \mathbb{M}^X & \xrightarrow{\mathbb{M}^h} & \mathbb{M}^Y \end{array}$$

that is  $\mathbb{M}^h \circ \eta_X = \eta_Y \circ (id \times h^{\mathbb{N}})$ .

$$\begin{aligned} (\mathbb{M}^h \circ \eta_X)(\varphi, f) &= \sum_{x \in h^{-1}(y)} \eta_X(\varphi, f)(x) && \text{(def. } \mathbb{M}^h \text{ applied to } \eta_X(\varphi, f)) \\ &= \sum_{x \in h^{-1}(y)} \sum_{n \in f^{-1}(x)} \varphi(n) && \text{(def. } \eta_X) \\ &= \sum_{n \in (h \circ f)^{-1}(y)} \varphi(n) && \text{(} f \text{ and } h \text{ are functions)} \\ &= \eta_Y(\varphi, h \circ f) && \text{(def. } \eta_Y) \\ &= (\eta_Y \circ (id \times h^{\mathbb{N}}))(\varphi, f) \end{aligned}$$

□

**6.1.3 COROLLARY.** *The functor  $\mathbb{M}^-$  has a final coalgebra.*

PROOF. By [52, Theorem 7.2], the fact that  $\mathbb{M}^-$  is bounded is enough to guarantee the existence of a final coalgebra. □

Recall that  $\mathbb{M}^-$  does not preserve weak-pullbacks [51], but it preserves arbitrary intersections [51, Corollary 5.4], which we need to define smallest subcoalgebras. The fact that  $\mathbb{M}^-$  does not preserve weak-pullbacks has as consequence that the notions of bisimilarity and observational equivalence might not coincide. Due to this fact, the soundness and completeness results of the axiomatization we shall later introduce will be formulated using the notion of behavioral equivalence.

We finish this section with an example of quantitative systems-weighted automata-modeled as coalgebras of a functor which contains the monoidal exponentiation as a subfunctor.

**Weighted Automata.** Weighted automata [43, 104] are transition systems labeled over a set  $A$  and with weights in a semiring  $\mathbb{S}$ . Moreover, each state is equipped with an output value<sup>1</sup> in  $\mathbb{S}$ . A *semiring*  $\mathbb{S}$  is a tuple  $\langle \mathbb{S}, +, \times, 0, 1 \rangle$  where  $\langle \mathbb{S}, +, 0 \rangle$  is a commutative monoid and  $\langle \mathbb{S}, \times, 1 \rangle$  is a monoid satisfying certain distributive laws. Examples of semirings include the real numbers  $\mathbb{R}$ , with usual addition and multiplication, and the Boolean semiring  $\mathbf{2}$  with disjunction and conjunction.

<sup>1</sup>In the original formulation also an input value is considered. To simplify the presentation and following [31] we omit it.

From a coalgebraic perspective, weighted automata are coalgebras of the functor  $\mathcal{W} = \mathbb{S} \times (\mathbb{S}^{\text{Id}})^A$ , where we write again  $\mathbb{S}$  to denote the commutative monoid of the semiring  $\mathbb{S}$ . More concretely, a coalgebra for  $\mathbb{S} \times (\mathbb{S}^{\text{Id}})^A$  is a pair  $(S, \langle o, T \rangle)$ , where  $S$  is a set of states,  $o: S \rightarrow \mathbb{S}$  is the function that associates an output weight to each state  $s \in S$  and  $T: S \rightarrow (\mathbb{S}^S)^A$  is the transition relation that associates a weight to each transition. We will use the following notation in the representation of weighted automata:

$$\begin{array}{c} \textcircled{s} \xrightarrow{a,w} \textcircled{s'} \\ \downarrow \quad \downarrow \\ o_s \quad o_{s'} \end{array} \Leftrightarrow T(s)(a)(s') = w \text{ and } o(s) = o_s \text{ and } o(s') = o_{s'}$$

If the set of states  $S$  and the alphabet  $A$  are finite, weighted automata can be conveniently represented in the following way. Let  $S = \{s_1, \dots, s_n\}$  be the set of states and  $A = \{a_1, \dots, a_m\}$  the input alphabet. The output function  $o$  can be seen as a vector with  $n$  entries

$$o = \begin{pmatrix} o(s_1) \\ \vdots \\ o(s_n) \end{pmatrix}$$

and the transition function  $T$  is a set of  $m$  matrices (of dimension  $n \times n$ )

$$T_{a_i} = \begin{pmatrix} t_{11} & \dots & t_{1n} \\ \vdots & & \vdots \\ t_{n1} & \dots & t_{nn} \end{pmatrix} \text{ with } t_{jk} = t(s_j)(a_i)(s_k)$$

This representation has advantages in the definition of homomorphism between two weighted automata. Composition of homomorphisms can be expressed as matrix multiplication, making it easier to check the commutativity of the diagram below (recall that the definition of the monoidal exponentiation function on arrows (Definition 6.1.1) is not very simple).

Let  $(S, \langle o, T \rangle)$  and  $(S', \langle o', T' \rangle)$  be two weighted automata. A homomorphism between these automata is a function  $h: S \rightarrow S'$  which makes the following diagram commute

$$\begin{array}{ccc} S & \xrightarrow{h} & S' \\ \langle o, T \rangle \downarrow & & \downarrow \langle o', T' \rangle \\ \mathbb{S} \times (\mathbb{S}^S)^A & \xrightarrow{id \times (\mathbb{S}^h)^A} & \mathbb{S} \times (\mathbb{S}^{S'})^A \end{array}$$

Now, representing  $h: S \rightarrow S'$ , with  $S = \{s_1, \dots, s_n\}$  and  $S' = \{s'_1, \dots, s'_m\}$  as a matrix with dimensions  $n \times m$  in the following way

$$h = \begin{pmatrix} h_{11} & \dots & h_{1m} \\ \vdots & & \vdots \\ h_{n1} & \dots & h_{nm} \end{pmatrix} \text{ with } h_{jk} = \begin{cases} 1 & \text{if } h(s_j) = s'_k \\ 0 & \text{otherwise} \end{cases}$$

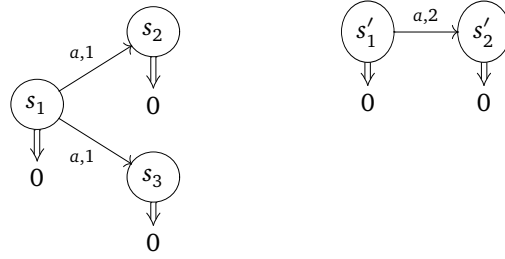


one can formulate the commutativity condition of the diagram above –  $(id \times (\mathbb{S}^h)^A) \circ \langle o, T \rangle = \langle o', T' \rangle \circ h$  – as the following matrix equalities:

$$o = h \times o' \text{ and } \forall_{a \in A} T_a \times h = h \times T'_a$$

Here, note that  $((\mathbb{S}^h)^A \circ T_a)(s_i)(s'_j) = (T_a \times h)(i, j)$ ,  $T'_a \circ h = h \times T'_a$  and  $o' \circ h = h \times o'$  (we are using the same letters to denote the functions, on the left side of the equations, and their representation as matrices, on the right side).

For a concrete example, let  $\mathbb{S} = \mathbb{R}$ , let  $A = \{a\}$  and consider the two weighted automata depicted below.



$$o = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad T_a = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad o' = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad T'_a = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}$$

Now consider the morphism  $h: S \rightarrow S'$  which maps  $s_1$  to  $s'_1$  and both  $s_2$  and  $s_3$  to  $s'_2$ , that is, it corresponds to the matrix

$$h = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

We now compute

$$h \times o' = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = o \text{ and } T_a \times h = \begin{pmatrix} 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = h \times T'_a$$

and we can conclude that  $h$  is a coalgebra homomorphism.

It is worth recalling that coalgebra homomorphisms always map states into bisimilar states ([96, Lemma 5.3]). Thus, since  $h$  is a  $\mathbb{R} \times (\mathbb{R}^{\text{ld}})^A$ -homomorphism,  $s_1$  is bisimilar to  $s'_1$  and  $s_2, s_3$  are bisimilar to  $s'_2$ .

Note that the multiplicative monoid  $\langle \mathbb{S}, \times, 1 \rangle$  plays no role in the coalgebraic definition of weighted automata. Indeed, in [43, 104], it is used only to define the weight of a sequence of transitions. Bisimilarity for weighted automata has been studied in [30] and it coincides with the coalgebraic notion of behavioral equivalence for the functor  $\mathbb{S} \times (\mathbb{S}^{\text{ld}})^A$  (Definition 2.2.5).

**6.1.4 PROPOSITION.** *Behavioral equivalence for  $\mathbb{S} \times (\mathbb{S}^{\text{ld}})^A$  coincides with the weighted automata bisimilarity defined in [30].*

PROOF. The definition of homomorphism which we stated above using matrix multiplication coincides with the definition of functional simulation [30, Definition 3.1]. Then, by [30, Corollary 3.6], two weighted automata  $(S, \langle o, T \rangle)$  and  $(S', \langle o', T' \rangle)$  are bisimilar if and only if there exists a weighted automata  $(Q, \langle o_1, T_1 \rangle)$  such that there exist surjective functional simulations  $h: S \rightarrow Q$  and  $h': S' \rightarrow Q$ . In coalgebraic terms, this means that  $h$  and  $h'$  form a cospan of coalgebra homomorphisms. Thus, for any  $s \in S$  and  $s' \in S'$ , if they are bisimilar according to [30], that is if  $h(s) = h'(s')$ , then we have that  $\mathbf{beh}_Q(h(s)) = \mathbf{beh}_Q(h'(s'))$  which, by uniqueness of the map into the final coalgebra, implies that  $\mathbf{beh}_S(s) = \mathbf{beh}_{S'}(s')$ . Hence, the states  $s$  and  $s'$  are behaviorally equivalent. The converse implication follows using a similar reasoning and we omit it here.  $\square$

## 6.2 A non-idempotent algebra for quantitative regular behaviors

In this section, we will extend the framework presented in the previous chapter in order to deal with quantitative systems, as described in the previous section. We will start by defining an appropriate class of functors  $\mathcal{H}$ , proceed with presenting the language  $\text{Exp}_{\mathcal{H}}$  of expressions associated with  $\mathcal{H}$  together with a Kleene like theorem and finally we introduce an axiomatization of  $\text{Exp}_{\mathcal{H}}$  and prove it sound and complete with respect to behavioral equivalence.

**6.2.1 DEFINITION.** The set  $QF$  of *quantitative functors* on  $\mathbf{Set}$  is defined inductively by:

$$QF \ni \mathcal{H} ::= \mathcal{G} \mid \mathbb{M}^{\mathcal{H}} \mid (\mathbb{M}^{\mathcal{H}})^A \mid \mathbb{M}_1^{\mathcal{H}c_1} \times \mathbb{M}_2^{\mathcal{H}c_2} \mid \mathbb{M}_1^{\mathcal{H}c_1} \oplus \mathbb{M}_2^{\mathcal{H}c_2}$$

where  $\mathcal{G}$  is a non-deterministic functor,  $\mathbb{M}$  is a commutative monoid and  $A$  is a finite set.  $\clubsuit$

Note that we do not allow *mixed* functors, such as  $\mathcal{G} + \mathbb{M}^{\mathcal{H}}$  or  $\mathcal{G} \times \mathbb{M}^{\mathcal{H}}$ . The reason for this restriction will become clear later in this section when we discuss the proof of Kleene's theorem. In Section 6.3, we will show how to deal with such mixed functors. Also, recall that the class of non-deterministic functors includes the  $\mathcal{P}_\omega$  functor (which is isomorphic to  $2^{\text{ld}}$ ): this will not be a source of problems as we show in Example 6.2.7.

We need now to extend several definitions presented in the previous chapter. The definition of the ingredient associated with a functor is extended in the expected way, as we show next.

**6.2.2 DEFINITION.** Let  $\triangleleft \subseteq QF \times QF$  be the least reflexive and transitive relation on quantitative functors such that

$$\begin{aligned} \mathcal{H}_1 \triangleleft \mathcal{H}_1 \times \mathcal{H}_2, \quad \mathcal{H}_2 \triangleleft \mathcal{H}_1 \times \mathcal{H}_2, \quad \mathcal{H}_1 \triangleleft \mathcal{H}_1 \oplus \mathcal{H}_2, \quad \mathcal{H}_2 \triangleleft \mathcal{H}_1 \oplus \mathcal{H}_2, \\ \mathcal{H} \triangleleft \mathcal{H}^A, \quad \mathcal{H} \triangleleft \mathcal{P}_\omega \mathcal{H}, \quad \mathcal{H} \triangleleft \mathbb{M}^{\mathcal{H}} \end{aligned}$$



All the other definitions we presented in the previous chapter need now to be extended to quantitative functors. We start by observing that taking the current definitions and replacing the subscript  $\mathcal{F} \triangleleft \mathcal{G}$  with  $\mathcal{F} \triangleleft \mathcal{H}$  does most of the work. In fact, having that, we just need to extend all the definitions for the case  $\mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{H}$ .

We start by introducing a new expression  $m \cdot \varepsilon$ , which we highlight in the definition, with  $m \in \mathbb{M}$ , extending the set of untyped expressions.

**6.2.3 DEFINITION** (Expressions for quantitative functors). Let  $A$  be a finite set,  $B$  a finite join-semilattice,  $\mathbb{M}$  a commutative monoid and  $X$  a set of fixed point variables. The set  $\text{Exp}$  of all *expressions* is given by the following grammar, where  $a \in A$ ,  $b \in B$ ,  $m \in \mathbb{M}$  and  $x \in X$ :

$$\varepsilon ::= \underline{0} \mid x \mid \varepsilon \oplus \varepsilon \mid \mu x. \gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid \{\varepsilon\} \mid m \cdot \varepsilon$$

where  $\gamma$  is a *guarded expression* given by:

$$\gamma ::= \underline{0} \mid \gamma \oplus \gamma \mid \mu x. \gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid \{\varepsilon\} \mid m \cdot \varepsilon$$



The intuition behind the new expression  $m \cdot \varepsilon$  is that there is a transition between the current state and the state specified by  $\varepsilon$  with weight  $m$ .

The type system will have one extra rule, which we highlight in the definition.

**6.2.4 DEFINITION** (Type system). We now define a typing relation  $\vdash \subseteq \text{Exp} \times QF \times QF$  that will associate an expression  $\varepsilon$  with two quantitative functors  $\mathcal{F}$  and  $\mathcal{H}$ , which are related by the ingredient relation ( $\mathcal{F}$  is an ingredient of  $\mathcal{H}$ ). We shall write  $\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{H}$  for  $\langle \varepsilon, \mathcal{F}, \mathcal{H} \rangle \in \vdash$ . The rules that define  $\vdash$  are the following:

$$\begin{array}{c} \frac{}{\vdash \underline{0} : \mathcal{F} \triangleleft \mathcal{H}} \quad \frac{}{\vdash b : B \triangleleft \mathcal{H}} \quad \frac{}{\vdash x : \mathcal{H} \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{H} \triangleleft \mathcal{H}}{\vdash \mu x. \varepsilon : \mathcal{H} \triangleleft \mathcal{H}} \\ \frac{\vdash \varepsilon_1 : \mathcal{F} \triangleleft \mathcal{H} \quad \vdash \varepsilon_2 : \mathcal{F} \triangleleft \mathcal{H}}{\vdash \varepsilon_1 \oplus \varepsilon_2 : \mathcal{F} \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{H} \triangleleft \mathcal{H}}{\vdash \varepsilon : \text{Id} \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{H}}{\vdash \{\varepsilon\} : \mathcal{P}_\omega \mathcal{F} \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{H}}{\vdash a(\varepsilon) : \mathcal{F}^A \triangleleft \mathcal{H}} \\ \frac{\vdash \varepsilon : \mathcal{F}_1 \triangleleft \mathcal{H}}{\vdash l(\varepsilon) : \mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{F}_2 \triangleleft \mathcal{H}}{\vdash r(\varepsilon) : \mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{F}_1 \triangleleft \mathcal{H}}{\vdash l[\varepsilon] : \mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}} \quad \frac{\vdash \varepsilon : \mathcal{F}_2 \triangleleft \mathcal{H}}{\vdash r[\varepsilon] : \mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}} \\ \frac{\varepsilon : \mathcal{F} \triangleleft \mathcal{H}}{m \cdot \varepsilon : \mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{H}} \end{array}$$



As before, we define  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$  by:

$$\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} = \{\varepsilon \in \text{Exp}^c \mid \vdash \varepsilon : \mathcal{F} \triangleleft \mathcal{H}\}.$$

The set  $\text{Exp}_{\mathcal{H}}$  of well-typed  $\mathcal{H}$ -expressions equals  $\text{Exp}_{\mathcal{H} \triangleleft \mathcal{H}}$ .

Next, we provide the set  $\text{Exp}_{\mathcal{H}}$  with a coalgebraic structure. We define a function  $\delta_{\mathcal{F} \triangleleft \mathcal{H}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{H}})$  and then set  $\delta_{\mathcal{H}} = \delta_{\mathcal{H} \triangleleft \mathcal{H}}$ . We show the definition of  $\delta_{\mathcal{F} \triangleleft \mathcal{H}}$  as well as of the auxiliary constant  $\text{Empty}_{\mathcal{F} \triangleleft \mathcal{H}}$  and function  $\text{Plus}_{\mathcal{F} \triangleleft \mathcal{H}}$ . As before we highlight the new part of the definition when compared with the definition for non-deterministic functors.

**6.2.5 DEFINITION.** For every  $\mathcal{H} \in QF$  and for every  $\mathcal{F}$  with  $\mathcal{F} \triangleleft \mathcal{H}$ :

- (i) we define a constant  $\text{Empty}_{\mathcal{F} \triangleleft \mathcal{H}} \in \mathcal{F}(\text{Exp}_{\mathcal{H}})$  by induction on the syntactic structure of  $\mathcal{F}$ :

$$\begin{array}{ll}
 \text{Empty}_{\text{Id} \triangleleft \mathcal{H}} & = \emptyset \\
 \text{Empty}_{\text{B} \triangleleft \mathcal{H}} & = \perp_{\text{B}} \\
 \text{Empty}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}} & = \langle \text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{H}}, \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{H}} \rangle \\
 \text{Empty}_{\mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{H}} & = \lambda c. 0
 \end{array}
 \qquad
 \begin{array}{ll}
 \text{Empty}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}} & = \perp \\
 \text{Empty}_{\mathcal{F}^A \triangleleft \mathcal{H}} & = \lambda a. \text{Empty}_{\mathcal{F} \triangleleft \mathcal{H}} \\
 \text{Empty}_{\mathbb{P}_\omega \mathcal{F} \triangleleft \mathcal{H}} & = \emptyset
 \end{array}$$

- (ii) we define a function  $\text{Plus}_{\mathcal{F} \triangleleft \mathcal{H}} : \mathcal{F}(\text{Exp}_{\mathcal{H}}) \times \mathcal{F}(\text{Exp}_{\mathcal{H}}) \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{H}})$  by induction on the syntactic structure of  $\mathcal{F}$ :

$$\begin{array}{ll}
 \text{Plus}_{\text{Id} \triangleleft \mathcal{H}}(\varepsilon_1, \varepsilon_2) & = \varepsilon_1 \oplus \varepsilon_2 \\
 \text{Plus}_{\text{B} \triangleleft \mathcal{H}}(b_1, b_2) & = b_1 \vee_{\text{B}} b_2 \\
 \text{Plus}_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}}(\langle \varepsilon_1, \varepsilon_2 \rangle, \langle \varepsilon_3, \varepsilon_4 \rangle) & = \langle \text{Plus}_{\mathcal{F}_1 \triangleleft \mathcal{H}}(\varepsilon_1, \varepsilon_3), \text{Plus}_{\mathcal{F}_2 \triangleleft \mathcal{H}}(\varepsilon_2, \varepsilon_4) \rangle \\
 \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(\kappa_i(\varepsilon_1), \kappa_i(\varepsilon_2)) & = \kappa_i(\text{Plus}_{\mathcal{F}_i \triangleleft \mathcal{H}}(\varepsilon_1, \varepsilon_2)), \quad i \in \{1, 2\} \\
 \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(\kappa_i(\varepsilon_1), \kappa_j(\varepsilon_2)) & = \top \quad i, j \in \{1, 2\} \text{ and } i \neq j \\
 \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(x, \top) & = \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(\top, x) = \top \\
 \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(x, \perp) & = \text{Plus}_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(\perp, x) = x \\
 \text{Plus}_{\mathcal{F}^A \triangleleft \mathcal{H}}(f, g) & = \lambda a. \text{Plus}_{\mathcal{F} \triangleleft \mathcal{H}}(f(a), g(a)) \\
 \text{Plus}_{\mathbb{P}_\omega \mathcal{F} \triangleleft \mathcal{H}}(s_1, s_2) & = s_1 \cup s_2 \\
 \text{Plus}_{\mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{H}}(f, g) & = \lambda c. f(c) + g(c)
 \end{array}$$

- (iii) we define a function  $\delta_{\mathcal{F} \triangleleft \mathcal{H}} : \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{H}})$ , by induction on the product of types of expressions and expressions (using the order defined in equation (5.1), extended with the clause  $N(m \cdot \varepsilon) = 0$ ). For every ingredient  $\mathcal{F}$  of a non-deterministic functor  $\mathcal{H}$  and an expression  $\varepsilon \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$ , we define  $\delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon)$  as

follows:

$$\begin{aligned}
\delta_{\mathcal{F} \triangleleft \mathcal{H}}(\emptyset) &= \text{Empty}_{\mathcal{F} \triangleleft \mathcal{H}} \\
\delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon_1 \oplus \varepsilon_2) &= \text{Plus}_{\mathcal{F} \triangleleft \mathcal{H}}(\delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon_1), \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon_2)) \\
\delta_{\mathcal{H} \triangleleft \mathcal{H}}(\mu x. \varepsilon) &= \delta_{\mathcal{H} \triangleleft \mathcal{H}}(\varepsilon[\mu x. \varepsilon/x]) \\
\delta_{\text{Id} \triangleleft \mathcal{H}}(\varepsilon) &= \varepsilon \text{ for } \mathcal{H} \neq \text{Id} \\
\delta_{\text{B} \triangleleft \mathcal{H}}(b) &= b \\
\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}}(l(\varepsilon)) &= \langle \delta_{\mathcal{F}_1 \triangleleft \mathcal{H}}(\varepsilon), \text{Empty}_{\mathcal{F}_2 \triangleleft \mathcal{H}} \rangle \\
\delta_{\mathcal{F}_1 \times \mathcal{F}_2 \triangleleft \mathcal{H}}(r(\varepsilon)) &= \langle \text{Empty}_{\mathcal{F}_1 \triangleleft \mathcal{H}}, \delta_{\mathcal{F}_2 \triangleleft \mathcal{H}}(\varepsilon) \rangle \\
\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(l[\varepsilon]) &= \kappa_1(\delta_{\mathcal{F}_1 \triangleleft \mathcal{H}}(\varepsilon)) \\
\delta_{\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleleft \mathcal{H}}(r[\varepsilon]) &= \kappa_2(\delta_{\mathcal{F}_2 \triangleleft \mathcal{H}}(\varepsilon)) \\
\delta_{\mathcal{F}^A \triangleleft \mathcal{H}}(a(\varepsilon)) &= \lambda a'. \begin{cases} \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon) & \text{if } a = a' \\ \text{Empty}_{\mathcal{F} \triangleleft \mathcal{H}} & \text{otherwise} \end{cases} \\
\delta_{\mathcal{P}, \mathcal{F} \triangleleft \mathcal{G}}(\{\varepsilon\}) &= \{\delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon)\} \\
\delta_{\mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{H}}(m \cdot \varepsilon) &= \lambda c. \begin{cases} m & \text{if } \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon) = c \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

♣

The function  $\delta_{\mathcal{H}} = \delta_{\mathcal{H} \triangleleft \mathcal{H}}$  provides an operational semantics for the expressions. We will soon illustrate this for the case of expressions for weighted automata (Example 6.2.8).

Finally, we introduce an equational system for expressions of type  $\mathcal{F} \triangleleft \mathcal{H}$ . We define the relation  $\equiv \subseteq \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} \times \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$ , written infix, as the least equivalence relation containing the following identities:

$$\begin{aligned}
\varepsilon \oplus \varepsilon &\equiv \varepsilon, & \varepsilon \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} & \quad (\text{Idempotency}) \\
\varepsilon_1 \oplus \varepsilon_2 &\equiv \varepsilon_2 \oplus \varepsilon_1 & & \quad (\text{Commutativity}) \quad \gamma[\mu x. \gamma/x] \equiv \mu x. \gamma \quad (\text{FP}) \\
\varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) &\equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 & & \quad (\text{Associativity}) \quad \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x. \gamma \equiv \varepsilon \quad (\text{Unique}) \\
\emptyset \oplus \varepsilon &\equiv \varepsilon & & \quad (\text{Empty})
\end{aligned}$$

$$\begin{array}{llll}
\emptyset \equiv \perp_{\text{B}} & (\text{B} - \emptyset) & b_1 \oplus b_2 \equiv b_1 \vee_{\text{B}} b_2 & (\text{B} - \oplus) \\
l(\emptyset) \equiv \emptyset & (\times - \emptyset - L) & l(\varepsilon_1 \oplus \varepsilon_2) \equiv l(\varepsilon_1) \oplus l(\varepsilon_2) & (\times - \oplus - L) \\
r(\emptyset) \equiv \emptyset & (\times - \emptyset - R) & r(\varepsilon_1 \oplus \varepsilon_2) \equiv r(\varepsilon_1) \oplus r(\varepsilon_2) & (\times - \oplus - R) \\
a(\emptyset) \equiv \emptyset & (-^A - \emptyset) & a(\varepsilon_1 \oplus \varepsilon_2) \equiv a(\varepsilon_1) \oplus a(\varepsilon_2) & (-^A - \oplus) \\
(0 \cdot \varepsilon) \equiv \emptyset & (\mathbb{M}^- - \emptyset) & (m \cdot \varepsilon) \oplus (m' \cdot \varepsilon) \equiv (m + m') \cdot \varepsilon & (\mathbb{M}^- - \oplus) \\
& & l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2] & (+ - \oplus - L) \\
& & r[\varepsilon_1 \oplus \varepsilon_2] \equiv r[\varepsilon_1] \oplus r[\varepsilon_2] & (+ - \oplus - R) \\
& & l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset] & (+ - \oplus - \top)
\end{array}$$

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x] \quad \text{if } x \text{ is free in } \varepsilon \quad (\text{Cong})$$

$$\mu x. \gamma \equiv \mu y. \gamma[y/x] \quad \text{if } y \text{ is not free in } \gamma \quad (\alpha - \text{equiv})$$

Note that (*Idempotency*) only holds for  $\varepsilon \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ , where  $\mathcal{G}$  is a non-deterministic functor. The reason why it cannot hold for the remaining functors comes from the fact that a monoid is, in general, not idempotent. Thus, (*Idempotency*) would conflict with the axiom  $(\mathbb{M}^- - \oplus)$ , which allows us to derive, for instance,  $(2 \cdot \underline{0}) \oplus (2 \cdot \underline{0}) \equiv 4 \cdot \underline{0}$ . In the case of an idempotent commutative monoid  $\mathbb{M}$ , (*Idempotency*) follows from the axiom  $(\mathbb{M}^- - \oplus)$ .

**6.2.6 LEMMA.** *Let  $\mathbb{M}$  be an idempotent commutative monoid. For every expression  $\varepsilon \in \text{Exp}_{\mathbb{M}^{\mathcal{F}} \triangleleft \mathcal{G}}$ , one has  $\varepsilon \oplus \varepsilon \equiv \varepsilon$ .*

PROOF. By induction on the product of types of expressions and expressions (using the order defined in equation (5.1)). Everything follows easily by induction. The most interesting case is  $\varepsilon = p \cdot \varepsilon_1$ . Then, by  $(\mathbb{M}^- - \oplus)$ ,  $(p \cdot \varepsilon_1) \oplus (p \cdot \varepsilon_1) \equiv (p + p) \cdot \varepsilon_1$  and, since the monoid is idempotent one has  $(p + p) \cdot \varepsilon_1 = p \cdot \varepsilon_1$ .  $\square$

**6.2.7 EXAMPLE** (Expressions for  $\mathcal{P}_\omega(\text{Id})$  and  $\mathbf{2}^{\text{Id}}$ ). The functor  $\mathcal{P}_\omega(\text{Id})$ , which we explicitly include in our syntax of quantitative functors (since it is a non-deterministic functor), is isomorphic to the functor  $\mathbf{2}^{\text{Id}}$ , an instance of the monoidal exponentiation functor. We shall next show that, as expected,  $\text{Exp}_{\mathcal{P}_\omega(\text{Id})/\equiv} \cong \text{Exp}_{\mathbf{2}^{\text{Id}}/\equiv}$ . The expressions for  $\mathcal{P}_\omega(\text{Id})$  are given by the closed and guarded expressions defined in the following BNF

$$\varepsilon ::= \underline{0} \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \mid x \mid \{\varepsilon\}$$

The axioms which apply for these expressions are the axioms for fixed points plus the axioms (*Associativity*), (*Commutativity*), (*Idempotency*) and (*Empty*).

For  $\mathbf{2}^{\text{Id}}$ , the expressions are given by the closed and guarded expressions defined in the following BNF

$$\varepsilon ::= \underline{0} \mid \varepsilon \oplus \varepsilon \mid \mu x. \varepsilon \mid x \mid 1 \cdot \varepsilon \mid 0 \cdot \varepsilon$$

The axiomatization contains the axioms for fixed points plus the axioms (*Associativity*), (*Commutativity*), (*Empty*),  $0 \cdot \varepsilon \equiv \underline{0}$  and  $p \cdot \varepsilon \oplus p' \cdot \varepsilon \equiv (p + p') \cdot \varepsilon$ . Because  $\mathbf{2}$  is an idempotent monoid, the last axiom can be replaced, for  $p = p'$ , by the (*Idempotency*) axiom (Lemma 6.2.6). For  $p \neq p'$ , note that  $0 \cdot \varepsilon \equiv \underline{0}$  applies and, using the fact that  $1 + 0 = 0$ , the axiom  $p \cdot \varepsilon \oplus p' \cdot \varepsilon \equiv (p + p') \cdot \varepsilon$  can be completely eliminated. This, together with the one but last axiom, yields that  $\text{Exp}_{\mathcal{P}_\omega(\text{Id})/\equiv} \cong \text{Exp}_{\mathbf{2}^{\text{Id}}/\equiv}$ .  $\spadesuit$

**6.2.8 EXAMPLE** (Expressions for weighted automata). The syntax canonically derived from our typing system for the functor  $\mathcal{W} = \mathbb{S} \times (\mathbb{S}^{\text{Id}})^A$ <sup>2</sup> is given by the closed and guarded expressions defined by the following BNF:

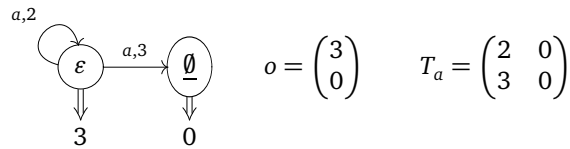
$$\begin{aligned} \varepsilon &::= \underline{0} \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x. \varepsilon \mid l \langle s \rangle \mid r \langle \varepsilon' \rangle \\ \varepsilon' &::= \underline{0} \mid \varepsilon' \oplus \varepsilon' \mid a \langle \varepsilon'' \rangle \\ \varepsilon'' &::= \underline{0} \mid \varepsilon'' \oplus \varepsilon'' \mid s \cdot \varepsilon \end{aligned}$$

<sup>2</sup>To be completely precise (in order for  $\mathcal{W}$  to be a quantitative functor) here the leftmost  $\mathbb{S}$  should be written as  $\mathbb{S}^{\{\ast\}}$ . However, it is easy to see that  $\text{Exp}_{\mathbb{S}^{\{\ast\}}/\equiv} \cong \mathbb{S}$  and so we will omit this detail from now on.

where  $s \in \mathbb{S}$  and  $a \in A$ . The operational semantics of these expressions is given by the function  $\delta_{\mathcal{W} \triangleleft \mathcal{W}}$  (hereafter denoted by  $\delta_{\mathcal{W}}$ ) which is an instance of the general definition of  $\delta_{\mathcal{F} \triangleleft \mathcal{H}}$  above. It is given by:

$$\begin{aligned}
\delta_{\mathcal{W}}(\emptyset) &= \langle 0, \lambda a. \lambda \varepsilon. 0 \rangle \\
\delta_{\mathcal{W}}(\varepsilon_1 \oplus \varepsilon_2) &= \langle s_1 + s_2, \lambda a. \lambda \varepsilon. (f(a)(\varepsilon) + g(s)(\varepsilon)) \rangle \\
&\quad \text{where } \langle s_1, f \rangle = \delta_{\mathcal{W}}(\varepsilon_1) \text{ and } \langle s_2, g \rangle = \delta_{\mathcal{W}}(\varepsilon_2) \\
\delta_{\mathcal{W}}(\mu x. \varepsilon) &= \delta_{\mathcal{W}}(\varepsilon[\mu x. \varepsilon/x]) \\
\delta_{\mathcal{W}}(l\langle s \rangle) &= \langle s, \lambda a. \lambda \varepsilon. 0 \rangle \\
\delta_{\mathcal{W}}(r\langle \varepsilon' \rangle) &= \langle 0, \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon') \rangle \\
\\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\emptyset) &= \lambda a. \lambda \varepsilon. 0 \\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_1 \oplus \varepsilon_2) &= \lambda a. \lambda \varepsilon. (f(a)(\varepsilon) + g(s)(\varepsilon)) \\
&\quad \text{where } f = \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_1) \text{ and } g = \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_2) \\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(a\langle \varepsilon'' \rangle) &= \lambda a'. \begin{cases} \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon'') & \text{if } a = a' \\ \lambda \varepsilon. 0 & \text{otherwise} \end{cases} \\
\\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\emptyset) &= \lambda \varepsilon. 0 \\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_1 \oplus \varepsilon_2) &= \lambda \varepsilon. (f(\varepsilon) + g(\varepsilon)) \\
&\quad \text{where } f = \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_1) \text{ and } g = \delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(\varepsilon_2) \\
\delta_{(\mathbb{S}^{\text{id}})^A \triangleleft \mathcal{W}}(s \cdot \varepsilon) &= \lambda \varepsilon'. \begin{cases} s & \text{if } \varepsilon = \varepsilon' \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

The function  $\delta_{\mathcal{W}}$  assigns to each expression  $\varepsilon$  a pair  $\langle s, t \rangle$ , consisting of an output weight  $s \in \mathbb{S}$  and a function  $t : A \rightarrow \mathbb{S}^{\text{Exp}_{\mathcal{W}}}$ . For a concrete example, let  $\mathbb{S} = \mathbb{R}$ ,  $A = \{a\}$ , and consider  $\varepsilon = \mu x. r\langle a(2 \cdot x \oplus 3 \cdot \emptyset) \rangle \oplus l\langle 1 \rangle \oplus l\langle 2 \rangle$ . The semantics of this expression, obtained by  $\delta_{\mathcal{W}}$  is described by the weighted automaton below.



In Table 6.1 a more concise syntax for expressions for weighted automata is presented (together with an axiomatization). It is interesting to remark that this syntax is a subset of the one proposed in [31] (there a parallel composition was also considered), but the axiomatization is new.

In order to see that the concise syntax and axiomatization are correct, one has to write translation maps between both syntaxes and then prove that if two expressions are provably equivalent in one syntax then their translations are provably equivalent as well.

We show next the aforementioned translations and then we do not show the full proof here but we will illustrate one case.

First, we translate the syntax presented in Table 6.1 into the canonically derived syntax presented above.

$$\begin{array}{ll} (\underline{\emptyset})^\dagger &= \underline{\emptyset} & s^\dagger &= l\langle s \rangle \\ (\varepsilon_1 \oplus \varepsilon_2)^\dagger &= (\varepsilon_1)^\dagger \oplus (\varepsilon_2)^\dagger & (a(s \cdot \varepsilon))^\dagger &= r\langle a(s \cdot \varepsilon^\dagger) \rangle \\ (\mu x. \varepsilon)^\dagger &= \mu x. \varepsilon^\dagger & x^\dagger &= x \end{array}$$

And now the converse translation:

$$\begin{array}{ll} (\underline{\emptyset})^\ddagger &= \underline{\emptyset} & (r(\underline{\emptyset}))^\ddagger &= \underline{\emptyset} \\ (\varepsilon_1 \oplus \varepsilon_2)^\ddagger &= (\varepsilon_1)^\ddagger \oplus (\varepsilon_2)^\ddagger & (r(\varepsilon'_1 \oplus \varepsilon'_2))^\ddagger &= (r(\varepsilon'_1))^\ddagger \oplus (r(\varepsilon'_2))^\ddagger \\ (\mu x. \varepsilon)^\ddagger &= \mu x. \varepsilon^\ddagger & (r(a(\underline{\emptyset})))^\ddagger &= \underline{\emptyset} \\ x^\ddagger &= x & (r(a(\varepsilon'_1 \oplus \varepsilon'_2)))^\ddagger &= (r(a(\varepsilon'_1)))^\ddagger \oplus (r(a(\varepsilon'_2)))^\ddagger \\ (l\langle s \rangle)^\ddagger &= s & (r(a(s \cdot \varepsilon)))^\ddagger &= a(s \cdot \varepsilon^\ddagger) \end{array}$$

Let us next show an example of the correctness of the syntax and axioms presented in Table 6.1. Take, from Table 6.1, the axiom  $a(0 \cdot \varepsilon) \equiv \underline{\emptyset}$ . We need to prove that  $(a(0 \cdot \varepsilon))^\dagger \equiv \underline{\emptyset}^\dagger$ , using the canonically derived axioms for  $\text{Exp}_{\mathcal{W}}$ . The left expression would be translated to  $r\langle a(0 \cdot \varepsilon^\dagger) \rangle$ , whereas  $\underline{\emptyset}$  would be translated to  $\underline{\emptyset}$ . Next, using the axioms of  $\text{Exp}_{\mathcal{W}}$  one derives  $r\langle a(0 \cdot \varepsilon^\dagger) \rangle \equiv r\langle a(\underline{\emptyset}) \rangle \equiv r(\underline{\emptyset}) \equiv \underline{\emptyset}$ , as expected.  $\spadesuit$

We are now ready to formulate the analogue of Kleene's theorem for quantitative systems.

**6.2.9 THEOREM** (Kleene's theorem for quantitative functors). *Let  $\mathcal{H}$  be a quantitative functor.*

1. *For every locally finite  $\mathcal{H}$ -coalgebra  $(S, h)$  and for every  $s \in S$  there exists an expression  $\varepsilon_s \in \text{Exp}_{\mathcal{H}}$  such that  $s \sim \varepsilon_s$ .*
2. *For every  $\varepsilon \in \text{Exp}_{\mathcal{H}}$ , there exists a finite  $\mathcal{H}$ -coalgebra  $(S, h)$  with  $s \in S$  such that  $s \sim \varepsilon$ .*

**PROOF.** Let  $\mathcal{H}$  be a quantitative functor. The proof of this theorem follows the same structure as the proof of 5.2.12 and 5.2.14, the corresponding theorems for non-deterministic functors.

*Proof of item 1.* Let  $s \in S$  and let  $\langle s \rangle = \{s_1, \dots, s_n\}$  with  $s_1 = s$ . We construct, for every state  $s_i \in \langle s \rangle$ , an expression  $\langle\langle s_i \rangle\rangle$  such that  $s_i \sim \langle\langle s_i \rangle\rangle$ .

If  $\mathcal{H} = \text{Id}$ , we set, for every  $i$ ,  $\langle\langle s_i \rangle\rangle = \underline{\emptyset}$ . It is easy to see that  $\{\langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle\}$  is a bisimulation and, thus, we have that  $s \sim \langle\langle s \rangle\rangle$ .

For  $\mathcal{H} \neq \text{Id}$ , we proceed in the following way. Let, for every  $i$ ,  $A_i = \mu x_i. \gamma_{h(s_i)}^{\mathcal{H}}$  where, for  $\mathcal{F} \triangleleft \mathcal{H}$  and  $c \in \mathcal{F}(s)$ , the expression  $\gamma_c^{\mathcal{F}} \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$  is defined by induction on the



structure of  $\mathcal{F}$ :

$$\begin{aligned}
\gamma_{s_i}^{\text{Id}} &= x_i & \gamma_b^{\text{B}} &= b & \gamma_{(c,c')}^{\mathcal{F}_1 \times \mathcal{F}_2} &= l\langle \gamma_c^{\mathcal{F}_1} \rangle \oplus r\langle \varepsilon_{c'}^{\mathcal{F}_2} \rangle & \gamma_f^{\mathcal{F}^A} &= \bigoplus_{a \in A} a(\gamma_{f(a)}^{\mathcal{F}}) \\
\gamma_{\kappa_1(c)}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= l[\gamma_c^{\mathcal{F}_1}] & \gamma_{\kappa_2(c)}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= r[\gamma_c^{\mathcal{F}_2}] & \gamma_{\perp}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= \underline{\emptyset} & \gamma_{\top}^{\mathcal{F}_1 \oplus \mathcal{F}_2} &= l[\underline{\emptyset}] \oplus r[\underline{\emptyset}] \\
\gamma_C^{\mathcal{P}_\omega \mathcal{F}} &= \begin{cases} \bigoplus_{c \in C} \{\gamma_c^{\mathcal{F}}\} & \text{if } C \neq \emptyset \\ \underline{\emptyset} & \text{otherwise} \end{cases} & \gamma_f^{M^{\mathcal{J}c_1}} &= \bigoplus_{\substack{c \in \mathcal{J}c_1(\langle s \rangle) \\ f(c) \neq 0}} f(c) \cdot \gamma_c^{\mathcal{J}c_1}
\end{aligned}$$

Now, let  $A_i^0 = A_i$ , define  $A_i^{k+1} = A_i^k \{A_{k+1}^k / x_{k+1}\}$  and then set  $\langle\langle s_i \rangle\rangle = A_i^n$ . Here,  $A\{A'/x\}$  denotes syntactic replacement (that is, substitution without renaming of bound variables in  $A$  which are also free variables in  $A'$ ).

Observe that the term

$$A_i^n = (\mu x_i. \gamma_{h(s_i)}^{\mathcal{J}c}) \{A_1^0 / x_1\} \dots \{A_n^{n-1} / x_n\}$$

is a closed term because, for every  $j = 1, \dots, n$ , the term  $A_j^{j-1}$  contains at most  $n - j$  free variables in the set  $\{x_{j+1}, \dots, x_n\}$ .

It remains to prove that  $s_i \sim \langle\langle s_i \rangle\rangle$ . We show that  $R = \{s_i, \langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle\}$  is a bisimulation. For that, we first define, for  $\mathcal{F} \triangleleft \mathcal{H}$  and  $c \in \mathcal{F}(s)$ ,  $\xi_c^{\mathcal{F}} = \gamma_c^{\mathcal{F}} \{A_1^0 / x_1\} \dots \{A_n^{n-1} / x_n\}$  and the relation

$$R_{\mathcal{F} \triangleleft \mathcal{H}} = \{c, \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\xi_c^{\mathcal{F}}) \mid c \in \mathcal{F}(s)\}.$$

Then, we prove that ①  $R_{\mathcal{F} \triangleleft \mathcal{H}} = \overline{\mathcal{F}(R)}$  and ②  $\langle h(s_i), \delta_{\mathcal{H}}(\langle\langle s_i \rangle\rangle) \rangle \in R_{\mathcal{H} \triangleleft \mathcal{H}}$ . We will show here the proof for the case  $\mathcal{F} = \mathbb{M}^{\mathcal{J}c_1}$ . The case when  $\mathcal{F}$  is a non-deterministic functor has been proved in Theorem 5.2.12. The remaining cases rely directly on  $\mathbb{M}^{\mathcal{J}c_1}$  and we omit them here.

$$\begin{aligned}
& \textcircled{1} \langle f, g \rangle \in \overline{\mathbb{M}^{\mathcal{J}c_1}(R)} \\
& \Leftrightarrow \exists \varphi: \mathcal{J}c_1(R) \rightarrow \mathbb{M} \quad \mathbb{M}^{\mathcal{J}c_1}(\pi_1)(\varphi) = f \text{ and } \mathbb{M}^{\mathcal{J}c_1}(\pi_2)(\varphi) = g && \text{(def. } \overline{\mathbb{M}^{\mathcal{J}c_1}(R)} \text{)} \\
& \Leftrightarrow f(u) = \sum_{\langle u, y \rangle \in \overline{\mathcal{J}c_1}(R)} \varphi(\langle u, y \rangle) \text{ and } g(v) = \sum_{\langle x, v \rangle \in \overline{\mathcal{J}c_1}(R)} \varphi(\langle x, v \rangle) && \text{(def. } \mathbb{M}^{\mathcal{J}c_1} \text{ on arrows)} \\
& \Leftrightarrow f(u) = \sum_{\langle u, y \rangle \in R_{\mathcal{J}c_1 \triangleleft \mathcal{H}}} \varphi(\langle u, y \rangle) \text{ and } g(v) = \sum_{\langle x, v \rangle \in R_{\mathcal{J}c_1 \triangleleft \mathcal{H}}} \varphi(\langle x, v \rangle) && \text{(ind. hyp.)} \\
& \Leftrightarrow f(u) = \varphi(\langle u, \delta(\xi_u^{\mathcal{J}c_1}) \rangle) \text{ and } g(v) = \sum_{v = \delta(\xi_x^{\mathcal{J}c_1})} \varphi(\langle x, \delta(\xi_x^{\mathcal{J}c_1}) \rangle) && \text{(def. } R_{\mathcal{J}c_1 \triangleleft \mathcal{H}} \text{)} \\
& \Leftrightarrow f(u) = \varphi(\langle u, \delta(\xi_u^{\mathcal{J}c_1}) \rangle) \text{ and } g(v) = \sum_{v = \delta(\xi_x^{\mathcal{J}c_1})} f(x) && \text{(def. } f \text{)} \\
& \Leftrightarrow f \in \mathbb{M}^{\mathcal{J}c_1}(\langle s \rangle) \text{ and } g = \delta_{\mathbb{M}^{\mathcal{J}c_1} \triangleleft \mathcal{H}}(\bigoplus_{f(x) \neq 0} f(x) \cdot \xi_x^{\mathcal{J}c_1}) && \text{(def. } \delta_{\mathbb{M}^{\mathcal{J}c_1} \triangleleft \mathcal{H}} \text{)} \\
& \Leftrightarrow f \in \mathbb{M}^{\mathcal{J}c_1}(\langle s \rangle) \text{ and } g = \delta_{\mathbb{M}^{\mathcal{J}c_1} \triangleleft \mathcal{H}}(\xi_f^{\mathbb{M}^{\mathcal{J}c_1}}) && \text{(def. } \xi_f^{\mathbb{M}^{\mathcal{J}c_1}} \text{)} \\
& \Leftrightarrow \langle f, g \rangle \in R_{\mathbb{M}^{\mathcal{J}c_1} \triangleleft \mathcal{H}}
\end{aligned}$$

The proof of ② is exactly as in the case of non-deterministic functors and thus we omit it here.

*Proof of item 2.* We want to show that for every expression  $\varepsilon \in \text{Exp}_{\mathcal{H}}$  there is a finite  $\mathcal{H}$ -coalgebra  $(S, f)$  with  $s \in S$  such that  $s \sim \varepsilon$ . We construct such coalgebra in the following way.

Again, we only show the proof for  $\mathcal{H} = \mathbb{M}^{\mathcal{H}c_1}$ . The case when  $\mathcal{H}$  is a non-deterministic functor has been proved in Theorem 5.2.14 and the other cases  $(\mathbb{M}^{\mathcal{H}c} \times \mathbb{M}^{\mathcal{H}c}, (\mathbb{M}^{\mathcal{H}c})^A$  and  $\mathbb{M}^{\mathcal{H}c} \oplus \mathbb{M}^{\mathcal{H}c})$  follow directly from the  $\mathcal{H} = \mathbb{M}^{\mathcal{H}c_1}$ , which we shall prove next.

For  $\varepsilon \in \text{Exp}_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}$ , we set  $(S, h) = \langle \varepsilon \rangle$  (recall that  $\varepsilon$  is the subcoalgebra generated by  $\varepsilon$ ). We now just have to prove that  $S$  is finite. In fact, we shall prove that  $S \subseteq cl(\varepsilon)$ , where  $cl(\varepsilon)$  denotes the smallest subset containing all subformulas of  $\varepsilon$  and the unfoldings of  $\mu$  (sub)formulas, that is, the smallest subset satisfying:

$$\begin{array}{ll} cl(\emptyset) & = \{\emptyset\} \\ cl(\varepsilon_1 \oplus \varepsilon_2) & = \{\varepsilon_1 \oplus \varepsilon_2\} \cup cl(\varepsilon_1) \cup cl(\varepsilon_2) \\ cl(\mu x. \varepsilon_1) & = \{\mu x. \varepsilon_1\} \cup cl(\varepsilon_1[\mu x. \varepsilon_1/x]) \\ cl(l\langle \varepsilon_1 \rangle) & = \{l\langle \varepsilon_1 \rangle\} \cup cl(\varepsilon_1) \\ cl(r\langle \varepsilon_1 \rangle) & = \{r\langle \varepsilon_1 \rangle\} \cup cl(\varepsilon_1) \\ cl(l[\varepsilon_1]) & = \{l[\varepsilon_1]\} \cup cl(\varepsilon_1) \\ cl(r[\varepsilon_1]) & = \{r[\varepsilon_1]\} \cup cl(\varepsilon_1) \\ cl(a(\varepsilon_1)) & = \{a(\varepsilon_1)\} \cup cl(\varepsilon_1) \\ cl(\{\varepsilon_1\}) & = \{\{\varepsilon_1\}\} \cup cl(\varepsilon_1) \\ cl(m \cdot \varepsilon_1) & = \{m \cdot \varepsilon_1\} \cup cl(\varepsilon_1) \end{array}$$

Note that the set  $cl(\varepsilon)$  is finite (the number of different unfoldings of  $\mu$ -expressions is finite).

To show that  $S \subseteq cl(\varepsilon)$  ( $S$  is the state space of  $\langle \varepsilon \rangle$ ), it is enough to show that, for any  $c \in \mathcal{H}_1(\text{Exp}_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c})$ ,  $\delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\varepsilon)(c) \neq 0 \Rightarrow c \in \mathcal{H}_1(cl(\varepsilon))$ .

It is an easy proof by induction on the product of types of expressions and expressions (using the order defined in equation (5.1), extended with the clause  $N(m \cdot \varepsilon) = 0$ ).

We exemplify the cases  $\varepsilon = \varepsilon_1 \oplus \varepsilon_2$

$$\begin{array}{ll} \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\varepsilon_1 \oplus \varepsilon_2)(c) \neq 0 \\ \Leftrightarrow \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\varepsilon_1)(c) \neq 0 \text{ or } \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\varepsilon_2)(c) \neq 0 & \text{(def. } \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}) \\ \Rightarrow c \in \mathcal{H}_1(cl(\varepsilon_1)) \text{ or } c \in \mathcal{H}_1(cl(\varepsilon_2)) & \text{(ind. hyp.)} \\ \Rightarrow c \in \mathcal{H}_1(cl(\varepsilon_1 \oplus \varepsilon_2)) & \text{(def. } cl) \end{array}$$

and  $\varepsilon = \mu x. \varepsilon_1$

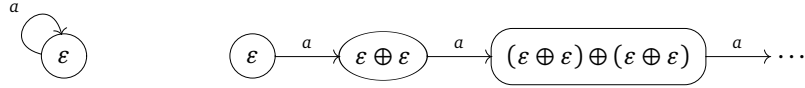
$$\begin{array}{ll} \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\mu x. \varepsilon_1)(c) \neq 0 \\ \Leftrightarrow \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\varepsilon_1[\mu x. \varepsilon_1/x])(c) \neq 0 & \text{(def. } \delta_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}c}(\mu x. \varepsilon_1)) \\ \Rightarrow c \in \mathcal{H}_1(cl(\varepsilon_1[\mu x. \varepsilon_1/x])) & \text{(ind. hyp.)} \\ \Rightarrow c \in \mathcal{H}_1(cl(\mu x. \varepsilon_1)) & (\mathcal{H}_1(cl(\varepsilon_1[\mu x. \varepsilon_1/x])) \subseteq \mathcal{H}_1(cl(\mu x. \varepsilon_1))) \end{array}$$

□

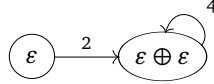
We can now explain the technical reason why we consider, in this section, only functors that are not mixed.

In the case of a non-deterministic functor  $\mathcal{G}$ , the proof of item 2. above requires considering subcoalgebras modulo (*Associativity*), (*Commutativity*) and (*Idempotency*) (ACI).

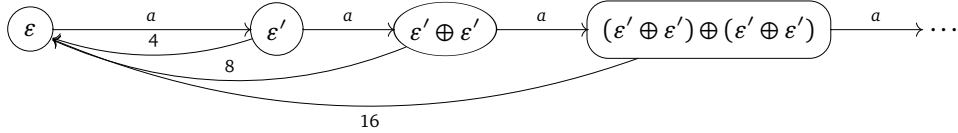
Consider for instance the expression  $\varepsilon = \mu x.r(a(x \oplus x))$  of type  $D = 2 \times \text{Id}^A$ . The subcoalgebras generated with and without applying ACI are the following:



In the case of  $\mathbb{M}^{\mathcal{H}^c}$  (or  $\mathbb{M}^{\mathcal{H}^c} \times \mathbb{M}^{\mathcal{H}^c}$ ,  $(\mathbb{M}^{\mathcal{H}^c})^A$  and  $\mathbb{M}^{\mathcal{H}^c} \oplus \mathbb{M}^{\mathcal{H}^c}$ ), the idempotency axiom does not hold anymore. However, surprisingly enough, in these cases proving the finiteness of the subcoalgebra  $\langle \varepsilon \rangle$  is not problematic. The key observation is that the monoid structure will be able to avoid the infinite scenario described above. What happens is concisely captured by the following example. Take the expression  $\varepsilon = \mu x.2 \cdot (x \oplus x)$  for the functor  $\mathbb{R}^{\text{Id}}$ . Then, the subcoalgebra generated by  $\varepsilon$  is depicted in the following picture:



The syntactic restriction that excludes mixed functors is needed because of the following problem. Take as an example the functor  $\mathbb{M}^{\text{Id}} \times \text{Id}^A$ . A well-typed expression for this functor would be  $\varepsilon = \mu x.r(a(x \oplus x \oplus l(2 \cdot x) \oplus l(2 \cdot x)))$ . It is clear that we cannot apply idempotency in the subexpression  $x \oplus x \oplus l(2 \cdot x) \oplus l(2 \cdot x)$  and hence the subcoalgebra generated by  $\varepsilon$  will be infinite:



with  $\varepsilon' = \varepsilon \oplus \varepsilon \oplus l(2 \cdot \varepsilon) \oplus l(2 \cdot \varepsilon)$ . We will show in the next section how to overcome this problem.

Let us summarize what we have achieved so far: we have presented a framework that allows, for each quantitative functor  $\mathcal{H} \in QF$ , the derivation of a language  $\text{Exp}_{\mathcal{H}^c}$ . Moreover, Theorem 6.2.9 guarantees that for each expression  $\varepsilon \in \text{Exp}_{\mathcal{H}^c}$ , there exists a finite  $\mathcal{H}$ -coalgebra  $(S, h)$  that contains a state  $s \in S$  bisimilar to  $\varepsilon$  and, conversely, for each locally finite  $\mathcal{H}$ -coalgebra  $(S, h)$  and for every state in  $s$  there is an expression  $\varepsilon_s \in \text{Exp}_{\mathcal{H}^c}$  bisimilar to  $s$ .

Next, we show that the axiomatization is sound and complete with respect to behavioral equivalence.

### Soundness and completeness

The proof of soundness and completeness follows exactly the same structure as the one presented in the previous chapter for non-deterministic functors. We will recall all the steps here, but will only show the proof of each theorem and lemma for the new case of the monoidal exponentiation functor. It is important to remark that

both the soundness and the completeness results will be formulated using behavioral equivalence rather than bisimilarity, as in the previous chapter.

The relation  $\equiv$  gives rise to the equivalence map  $[-]: \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} \rightarrow \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} / \equiv$ , defined by  $[\varepsilon] = \{\varepsilon' \mid \varepsilon \equiv \varepsilon'\}$ . The following diagram summarizes the maps we have defined so far:

$$\begin{array}{ccc} \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} & \xrightarrow{[-]} & \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} / \equiv \\ \delta_{\mathcal{F} \triangleleft \mathcal{H}} \downarrow & & \\ \mathcal{F}(\text{Exp}_{\mathcal{H}}) & \xrightarrow{\mathcal{F}[-]} & \mathcal{F}(\text{Exp}_{\mathcal{H}} / \equiv) \end{array}$$

In order to complete the diagram, we next prove that the relation  $\equiv$  is contained in the kernel of  $\mathcal{F}[-] \circ \delta_{\mathcal{F} \triangleleft \mathcal{H}}$ <sup>3</sup>. This will guarantee, by Theorem 2.2.7, the existence of a well-defined function  $\partial_{\mathcal{F} \triangleleft \mathcal{H}}: \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} / \equiv \rightarrow \mathcal{F}(\text{Exp}_{\mathcal{H}} / \equiv)$  which, when  $\mathcal{F} = \mathcal{H}$ , provides  $\text{Exp}_{\mathcal{H}} / \equiv$  with a coalgebraic structure  $\partial_{\mathcal{H}}: \text{Exp}_{\mathcal{H}} / \equiv \rightarrow \mathcal{H}(\text{Exp}_{\mathcal{H}} / \equiv)$  (as before we write  $\partial_{\mathcal{H}}$  for  $\partial_{\mathcal{H} \triangleleft \mathcal{H}}$ ) and which makes  $[-]$  a homomorphism of coalgebras.

**6.2.10 LEMMA.** *Let  $\mathcal{H}$  and  $\mathcal{F}$  be quantitative functors, with  $\mathcal{F} \triangleleft \mathcal{H}$ . For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$  with  $\varepsilon_1 \equiv \varepsilon_2$ ,*

$$(\mathcal{F}[-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon_1) = (\mathcal{F}[-]) \circ \delta_{\mathcal{F} \triangleleft \mathcal{H}}(\varepsilon_2)$$

PROOF. By induction on the length of derivations of  $\equiv$ .

We just show the proof for the axioms  $0 \cdot \varepsilon = \emptyset$  and  $(m \cdot \varepsilon) \oplus (m' \cdot \varepsilon) = (m + m') \cdot \varepsilon$ .

$$\begin{aligned} \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}(0 \cdot \varepsilon) = \lambda c. 0 &= \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}(\emptyset) \\ \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}((m \cdot \varepsilon) \oplus (m' \cdot \varepsilon)) &= \lambda c. \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}(m \cdot \varepsilon)(c) + \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}(m' \cdot \varepsilon)(c) \\ &= \lambda c. \begin{cases} m + m' & \text{if } \delta_{\mathcal{H} \triangleleft \mathcal{H}}(\varepsilon) = c \\ 0 & \text{otherwise} \end{cases} \\ &= \delta_{\mathbb{M}^{\mathcal{H} \triangleleft \mathcal{H}}}((m + m') \cdot \varepsilon) \end{aligned}$$

□

Thus, we have now provided the set  $\text{Exp}_{\mathcal{H}} / \equiv$  with a coalgebraic structure: we have defined a function  $\partial_{\mathcal{H}}: \text{Exp}_{\mathcal{H}} / \equiv \rightarrow \mathcal{H}(\text{Exp}_{\mathcal{H}} / \equiv)$ , with  $\partial_{\mathcal{H}}([\varepsilon]) = (\mathcal{H}[-]) \circ \delta_{\mathcal{H}}(\varepsilon)$ .

At this point we can prove soundness, since it is a direct consequence of the fact that the equivalence map  $[-]$  is a coalgebra homomorphism.

**6.2.11 THEOREM (Soundness).** *Let  $\mathcal{H}$  be a quantitative functor. For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{H}}$ ,*

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon_1 \sim_b \varepsilon_2$$

<sup>3</sup>This is equivalent to proving that  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}} / \equiv$ , together with  $[-]$ , is the coequalizer of the projection morphisms from  $\equiv$  to  $\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}$ .

PROOF. Let  $\mathcal{H}$  be a quantitative functor, let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{H}}$  and suppose that  $\varepsilon_1 \equiv \varepsilon_2$ . Then,  $[\varepsilon_1] = [\varepsilon_2]$  and, thus

$$\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([\varepsilon_1]) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([\varepsilon_2])$$

where  $\mathbf{beh}_S$  denotes, for any  $\mathcal{H}$ -coalgebra  $(S, f)$ , the unique map into the final coalgebra. The uniqueness of the map into the final coalgebra and the fact that  $[-]$  is a coalgebra homomorphism implies that  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv} \circ [-] = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}}$  which then yields

$$\mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_2)$$

Hence,  $\varepsilon_1 \sim_b \varepsilon_2$ .  $\square$

For completeness, we proceed as before (with the difference that now we use behavioral equivalence  $\sim_b$  instead of bisimilarity  $\sim$ ). Let us recall the key ingredients of the proof. The goal is to prove that  $\varepsilon_1 \sim_b \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$ . First, note that we have

$$\varepsilon_1 \sim_b \varepsilon_2 \Leftrightarrow \mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_2) \Leftrightarrow \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([\varepsilon_1]) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([\varepsilon_2]) \quad (6.1)$$

We then prove that  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}$  is injective, which is a sufficient condition to guarantee that  $\varepsilon_1 \equiv \varepsilon_2$  (since it implies, together with (6.1), that  $[\varepsilon_1] = [\varepsilon_2]$ ).

First, we factorize  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}$  into an epimorphism and a monomorphism [96, Theorem 7.1] as shown in the following diagram (where  $I = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}(\text{Exp}_{\mathcal{H}}/\equiv)$ ):

$$\begin{array}{ccccc} & & \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv} & & \\ & & \text{---} & & \\ \text{Exp}_{\mathcal{H}}/\equiv & \xrightarrow{e} & I & \xrightarrow{m} & \Omega_{\mathcal{H}} \\ & \downarrow \partial_{\mathcal{H}} & \downarrow \overline{\omega}_{\mathcal{H}} & & \downarrow \omega_{\mathcal{H}} \\ \mathcal{H}(\text{Exp}_{\mathcal{H}}/\equiv) & \longrightarrow & \mathcal{H}(I) & \longrightarrow & \mathcal{H}\Omega_{\mathcal{H}} \end{array} \quad (6.2)$$

Then, we prove that (1)  $(\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$  is a locally finite coalgebra (Lemma 6.2.12) and (2) both coalgebras  $(\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$  and  $(I, \overline{\omega}_{\mathcal{H}})$  are final in the category of locally finite  $\mathcal{H}$ -coalgebras (Lemmas 6.2.15 and 6.2.16, respectively). Since final coalgebras are unique up to isomorphism, it follows that  $e: \text{Exp}_{\mathcal{H}}/\equiv \rightarrow I$  is in fact an isomorphism and therefore  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}$  is injective, which will give us completeness.

We now proceed with presenting and proving the extra lemmas needed in order to prove completeness. We start by showing that the coalgebra  $(\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$  is locally finite (note that this implies that  $(I, \overline{\omega}_{\mathcal{H}})$  is also locally finite) and that  $\partial_{\mathcal{H}}$  is an isomorphism.

**6.2.12 LEMMA.** *The coalgebra  $(\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$  is a locally finite coalgebra. Moreover,  $\partial_{\mathcal{H}}$  is an isomorphism.*

PROOF. Locally finiteness is a direct consequence of the generalized Kleene's theorem (Theorem 6.2.9). In the proof of Theorem 6.2.9 we showed that given  $\varepsilon \in \text{Exp}_{\mathcal{H}}$ ,

for  $\mathcal{H} = \mathbb{M}^{\mathcal{H}_1}$ ,  $\mathcal{H} = \mathbb{M}^{\mathcal{H}_1} \times \mathbb{M}^{\mathcal{H}_2}$ ,  $\mathcal{H} = (\mathbb{M}^{\mathcal{H}_1})^A$  or  $\mathcal{H} = \mathbb{M}^{\mathcal{H}_1} \boxplus \mathbb{M}^{\mathcal{H}_2}$ , the subcoalgebra  $\langle \varepsilon \rangle$  is finite. In case  $\mathcal{H}$  is a non-deterministic functor, we proved that the subcoalgebra  $\langle [\varepsilon]_{ACIE} \rangle$  is finite. Thus, the subcoalgebra  $\langle [\varepsilon] \rangle$  is always finite (since  $\text{Exp}_{\mathcal{H}/\equiv}$  is a quotient of both  $\text{Exp}_{\mathcal{H}}$  and  $\text{Exp}_{\mathcal{H}/\equiv_{ACIE}}$ ). Recall that *ACIE* abbreviates the axioms (*Associativity*), (*Commutativity*), (*Idempotency*) and (*Empty*) and  $\equiv_{ACIE}$  denotes equivalence under these axioms only.

To see that  $\partial_{\mathcal{H}}$  is an isomorphism, first define, for every  $\mathcal{F} \triangleleft \mathcal{H}$ ,

$$\partial_{\mathcal{F} \triangleleft \mathcal{H}}^{-1}(c) = [\bar{\gamma}_c^{\mathcal{F}}] \quad (6.3)$$

where  $\bar{\gamma}_c^{\mathcal{F}}$  is defined, for  $\mathcal{F} \neq \text{Id}$ , as  $\gamma_c^{\mathcal{F}}$  in the proof of Theorem 6.2.9, and for  $\mathcal{F} = \text{Id}$  as  $\bar{\gamma}_{[\varepsilon]}^{\text{Id}} = \varepsilon$ . Then, we prove that the function  $\partial_{\mathcal{F} \triangleleft \mathcal{H}}^{-1}$  has indeed the desired properties ①  $\partial_{\mathcal{F} \triangleleft \mathcal{H}}^{-1} \circ \partial_{\mathcal{F} \triangleleft \mathcal{H}} = \text{id}_{\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}/\equiv}$  and ②  $\partial_{\mathcal{F} \triangleleft \mathcal{H}} \circ \partial_{\mathcal{F} \triangleleft \mathcal{H}}^{-1} = \text{id}_{\mathcal{F}(\text{Exp}_{\mathcal{F} \triangleleft \mathcal{H}}/\equiv)}$ . Instantiating  $\mathcal{F} = \mathcal{H}$  one derives that  $\delta_{\mathcal{H}}$  is an isomorphism. It is enough to prove for ① that  $\bar{\gamma}_{\partial_{\mathcal{F} \triangleleft \mathcal{H}}([\varepsilon])}^{\mathcal{F}} \equiv \varepsilon$  and for ② that  $\partial_{\mathcal{F} \triangleleft \mathcal{H}}([\bar{\gamma}_c^{\mathcal{F}}]) = c$ . We just illustrate the new cases when compared to Theorem 5.3.4.

① By induction on the product of types of expressions and expressions (using the order defined in equation (5.1)).

$$\begin{aligned} & \bar{\gamma}_{\partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([m \cdot \varepsilon])}^{\mathbb{M}^{\mathcal{H}_1}} \\ &= \bigoplus \{ \partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([m \cdot \varepsilon])(c) \cdot \bar{\gamma}_c^{\mathcal{H}_1} \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}/\equiv}), \partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([m \cdot \varepsilon])(c) \neq 0 \} \\ &= m \cdot \bar{\gamma}_{\partial_{\mathcal{H}_1 \triangleleft \mathcal{H}}([\varepsilon])}^{\mathcal{H}_1} \\ &\equiv m \cdot \varepsilon \end{aligned}$$

In the last step, we used the induction hypothesis, whereas in the one but last step we used the fact that  $\partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([m \cdot \varepsilon])(c) \neq 0 \Leftrightarrow c = \partial_{\mathcal{H}_1 \triangleleft \mathcal{H}}([\varepsilon])$ .

② By induction on the structure of  $\mathcal{F}$ .

$$\begin{aligned} \partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([\bar{\gamma}_f^{\mathbb{M}^{\mathcal{H}_1}}]) &= \partial_{\mathbb{M}^{\mathcal{H}_1} \triangleleft \mathcal{H}}([\bigoplus \{ f(c) \cdot \bar{\gamma}_c^{\mathcal{H}_1} \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}/\equiv}), f(c) \neq 0 \}]) \\ &= \lambda c'. \sum \{ f(c) \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}/\equiv}) \text{ and } c' = \partial_{\mathcal{H}_1 \triangleleft \mathcal{H}}(\bar{\gamma}_c^{\mathcal{H}_1}) \} \\ &\stackrel{IH}{=} \lambda c'. \sum \{ f(c) \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}/\equiv}) \text{ and } c' = c \} \\ &= f \end{aligned}$$

□

We now present the analogue of the following useful and intuitive equality on regular expressions, which we had already presented in the previous chapter for non-deterministic functors (Lemma 5.3.5). Given a deterministic automaton  $\langle o, t \rangle: S \rightarrow 2 \times S^A$  and a state  $s \in S$ , the associated regular expression  $r_s$  can be written as

$$r_s = o(s) + \sum_{a \in A} a \cdot r_{t(s)(a)} \quad (6.4)$$

using the axioms of Kleene algebra [29, Theorem 4.4].

**6.2.13 LEMMA.** *Let  $(S, h)$  be a locally finite  $\mathcal{H}$ -coalgebra, with  $\mathcal{H} \neq \text{Id}$ , and let  $s \in S$ , with  $\langle s \rangle = \{s_1, \dots, s_n\}$  (where  $s_1 = s$ ). Then:*

$$\langle\langle s_i \rangle\rangle \equiv \gamma_{g(s_i)}^{\mathcal{H}} \{\langle\langle s_1 \rangle\rangle / x_1\} \dots \{\langle\langle s_n \rangle\rangle / x_n\} \quad (6.5)$$

PROOF. Same as in Lemma 5.3.5.  $\square$

The above equality is used to prove that there exists a coalgebra homomorphism between any locally finite coalgebra  $(S, h)$  and  $(\text{Exp}_{\mathcal{H}} / \equiv, \partial_{\mathcal{H}})$ .

**6.2.14 LEMMA.** *Let  $(S, h)$  be a locally finite  $\mathcal{H}$ -coalgebra. There exists a coalgebra homomorphism  $\lceil - \rceil : S \rightarrow \text{Exp}_{\mathcal{H}} / \equiv$ .*

PROOF. We define  $\lceil - \rceil = [-] \circ \langle\langle - \rangle\rangle$ , where  $\langle\langle - \rangle\rangle$  is as in the proof of Theorem 6.2.9, associating to a state  $s$  of a locally finite coalgebra an expression  $\langle\langle s \rangle\rangle$  with  $s \sim \langle\langle s \rangle\rangle$ . To prove that  $\lceil - \rceil$  is a homomorphism we need to verify that  $(\mathcal{H}\lceil - \rceil) \circ h = \partial_{\mathcal{H}} \circ \lceil - \rceil$ . If  $\mathcal{H} = \text{Id}$ , then  $(\mathcal{H}\lceil - \rceil) \circ g(s_i) = [\emptyset] = \partial_{\mathcal{H}}(\lceil s_i \rceil)$ . For  $\mathcal{H} \neq \text{Id}$  we calculate, using Lemma 6.2.13:

$$\partial_{\mathcal{H}} \circ \lceil s_i \rceil = \partial_{\mathcal{H}}(\lceil \gamma_{g(s_i)}^{\mathcal{H}} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n] \rceil)$$

and we then prove the more general equality, for  $\mathcal{F} \triangleleft \mathcal{H}$  and  $c \in \mathcal{F}\langle s \rangle$ :

$$\partial_{\mathcal{F} \triangleleft \mathcal{H}}(\lceil \gamma_c^{\mathcal{F}} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n] \rceil) = \mathcal{F}\lceil - \rceil(c) \quad (6.6)$$

The intended equality then follows by taking  $\mathcal{F} = \mathcal{H}$  and  $c = g(s_i)$ . Let us prove the equation (6.6) by induction on  $\mathcal{F}$ . We only show the case  $\mathcal{F} = \mathbb{M}^{\mathcal{H}c_1}$ .

$$\begin{aligned} & \partial_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}}(\lceil \gamma_f^{\mathbb{M}^{\mathcal{H}c_1}} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n] \rceil) \\ &= \partial_{\mathbb{M}^{\mathcal{H}c_1} \triangleleft \mathcal{H}}(\lceil \bigoplus \{f(c) \cdot \gamma_c^{\mathcal{H}c_1} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n] \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}} / \equiv), f(c) \neq 0\} \rceil) \\ &= \lambda c'. \sum \{f(c) \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}} / \equiv) \text{ and } c' = \partial_{\mathcal{H}c_1 \triangleleft \mathcal{H}}(\gamma_c^{\mathcal{H}c_1} [\langle\langle s_1 \rangle\rangle / x_1] \dots [\langle\langle s_n \rangle\rangle / x_n])\} \\ &\stackrel{\text{IH}}{=} \lambda c'. \sum \{f(c) \mid c \in \mathcal{H}_1(\text{Exp}_{\mathcal{H}} / \equiv) \text{ and } c' = (\mathcal{H}_1\lceil - \rceil)(c)\} \\ &= \mathbb{M}^{\mathcal{H}c_1}(\lceil - \rceil)(f) \end{aligned}$$

$\square$

We can now prove that the coalgebras  $(\text{Exp}_{\mathcal{H}} / \equiv, \partial_{\mathcal{H}})$  and  $(I, \overline{\omega}_{\mathcal{H}})$  are both final in the category of locally finite  $\mathcal{H}$ -coalgebras.

**6.2.15 LEMMA.** *The coalgebra  $(I, \overline{\omega}_{\mathcal{H}})$  is final in the category  $\text{Coalg}(\mathcal{H})_{\text{LF}}$ .*

PROOF. We want to show that for any locally finite coalgebra  $(S, h)$ , there exists a *unique* homomorphism  $(S, h) \rightarrow (I, \overline{\omega}_{\mathcal{H}})$ . Lemma 6.2.14, where  $\lceil - \rceil : S \rightarrow \text{Exp}_{\mathcal{H}} / \equiv$  is defined, guarantees the existence. Postcomposing this homomorphism with  $e$  (defined above, in diagram 6.2) we get a coalgebra homomorphism  $e \circ \lceil - \rceil : S \rightarrow I$ . If there is another homomorphism  $f : S \rightarrow I$ , then by postcomposition with the inclusion  $m : I \hookrightarrow \Omega$  we get two homomorphisms  $(m \circ f$  and  $m \circ e \circ \lceil - \rceil)$  into the final  $\mathcal{H}$ -coalgebra. Thus,  $f$  and  $e \circ \lceil - \rceil$  must be equal.  $\square$

**6.2.16 LEMMA.** *The coalgebra  $(\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$  is final in the category  $\text{Coalg}(\mathcal{H})_{\text{LF}}$ .*

PROOF. We want to show that for any locally finite coalgebra  $(S, h)$ , there exists a *unique* homomorphism  $(S, h) \rightarrow (\text{Exp}_{\mathcal{H}}/\equiv, \partial_{\mathcal{H}})$ . We only need to prove uniqueness, since the existence is guaranteed by Lemma 6.2.14, where  $[-]: S \rightarrow \text{Exp}_{\mathcal{H}}/\equiv$  is defined.

Suppose we have another homomorphism  $f: S \rightarrow \text{Exp}_{\mathcal{H}}/\equiv$ . Then, we shall prove that  $f = [-]$ . First, observe that because  $f$  is a homomorphism the following holds for every  $s \in S$ :

$$f(s) \equiv \partial_{\mathcal{H}}^{-1} \circ \mathcal{H}f \circ h(s) \equiv \gamma_{g(s)}^{\mathcal{H}} [f(s_1)/x_1] \dots [f(s_n)/x_n] \quad (6.7)$$

where  $\langle s \rangle = \{s_1, \dots, s_n\}$ , with  $s_1 = s$  (recall that  $\partial_{\mathcal{H}}^{-1}$  was defined in (6.3) and note that  $\overline{\gamma}_{\mathcal{H}f \circ h(s)}^{\mathcal{H}} = \gamma_{h(s)}^{\mathcal{H}} [f(s_i)/x_i]$ ).

We now have to prove that  $f(s_i) = [s_i]$ , for all  $i = 1, \dots, n$ . The proof, which relies mainly on uniqueness of fixed points, is exactly as in Theorem 5.3.8 and we omit it here.  $\square$

At this point, because final objects are unique up-to isomorphism, we know that  $e: \text{Exp}_{\mathcal{H}}/\equiv \rightarrow I$  is an isomorphism and hence we can conclude that the map  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}$  is injective, since it factorizes into an isomorphism followed by a mono. This fact is the last ingredient we need to prove completeness.

**6.2.17 THEOREM (Completeness).** *Let  $\mathcal{H}$  be a quantitative functor. For all  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{H}}$ ,*

$$\varepsilon_1 \sim_b \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$$

PROOF. Let  $\mathcal{H}$  be a quantitative functor, let  $\varepsilon_1, \varepsilon_2 \in \text{Exp}_{\mathcal{H}}$  and suppose that  $\varepsilon_1 \sim_b \varepsilon_2$ , that is,  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_1) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}}(\varepsilon_2)$ . We reason as in equation (6.1). Since the equivalence class map  $[-]$  is a homomorphism, it holds that  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([ \varepsilon_1 ]) = \mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}([ \varepsilon_2 ])$ . Now, because  $\mathbf{beh}_{\text{Exp}_{\mathcal{H}}/\equiv}$  is injective we have that  $[ \varepsilon_1 ] = [ \varepsilon_2 ]$ . Hence,  $\varepsilon_1 \equiv \varepsilon_2$ .  $\square$

### 6.3 Extending the class of functors

In the previous section, we introduced regular expressions for the class of quantitative functors. In this section, by employing standard results from the theory of coalgebras, we show how to use such regular expressions to describe the coalgebras of many more functors, including the mixed functors we mentioned in Section 6.2.

Given two endofunctors  $\mathcal{F}$  and  $\mathcal{G}$  on **Set**, a *natural transformation*  $\alpha: \mathcal{F} \Rightarrow \mathcal{G}$  is a family of functions  $\alpha_S: \mathcal{F}(S) \rightarrow \mathcal{G}(S)$  (for all sets  $S$ ), such that for all functions  $h: T \rightarrow U$ ,  $\alpha_U \circ \mathcal{F}(h) = \mathcal{G}(h) \circ \alpha_T$ . If all the  $\alpha_S$  are injective, then we say that  $\alpha$  is injective.

**6.3.1 PROPOSITION.** *An injective natural transformation  $\alpha: \mathcal{F} \Rightarrow \mathcal{G}$  induces a functor  $\alpha \circ (-): \text{Coalg}(\mathcal{F})_{\text{LF}} \rightarrow \text{Coalg}(\mathcal{G})_{\text{LF}}$  that preserves and reflects behavioral equivalence.*



PROOF. It is shown in [17, 96] that an injective  $\alpha: \mathcal{F} \Rightarrow \mathcal{G}$  induces a functor  $\alpha \circ (-): \text{Coalg}(\mathcal{F}) \rightarrow \text{Coalg}(\mathcal{G})$  that preserves and reflects behavioral equivalence. Thus we have only to prove that  $\alpha \circ (-)$  maps locally finite  $\mathcal{F}$ -coalgebras into locally finite  $\mathcal{G}$ -coalgebras.

Recall that  $\alpha \circ (-)$  maps each  $\mathcal{F}$ -coalgebra  $(S, f)$  into the  $\mathcal{G}$ -coalgebra  $(S, \alpha \circ f)$ , and each  $\mathcal{F}$ -homomorphism into itself. We prove that if  $(S, f)$  is locally finite, then also  $(S, \alpha \circ f)$  is locally finite.

An  $\mathcal{F}$ -coalgebra  $(S, f)$  is locally finite if for all  $s \in S$ , there exists a finite set  $V \subseteq S$ , such that  $s \in V$  and  $V$  is a subsystem of  $S$ , that is there exists a function  $\nu: V \rightarrow \mathcal{F}(V)$ , such that inclusion  $i: V \rightarrow S$  is an  $\mathcal{F}$ -homomorphism between  $(V, \nu)$  and  $(S, f)$ . At this point, note that if  $i: V \rightarrow S$  is an  $\mathcal{F}$ -homomorphism between  $(V, \nu)$  and  $(S, f)$ , then it is also a  $\mathcal{G}$ -homomorphism between  $(V, \alpha \circ \nu)$  and  $(S, \alpha \circ f)$ . Thus,  $(S, \alpha \circ f)$  is locally finite.  $\square$

This result allows us to extend our framework to many other functors, as we shall explain next. Consider a functor  $\mathcal{F}$  which is not quantitative and suppose there exists an injective natural transformation  $\alpha$  from  $\mathcal{F}$  into some quantitative functor  $\mathcal{H}$ . A (locally finite)  $\mathcal{F}$ -coalgebra can be translated into a (locally finite)  $\mathcal{H}$ -coalgebra via the functor  $\alpha \circ (-)$  and then it can be characterized by using expressions in  $\text{Exp}_{\mathcal{H}}$ . The axiomatization for  $\text{Exp}_{\mathcal{H}}$  is still sound and complete for  $\mathcal{F}$ -coalgebras, since the functor  $\alpha \circ (-)$  preserves and reflects behavioral equivalence.

However, note that (half of) Kleene's theorem does not hold anymore, because not all the expressions in  $\text{Exp}_{\mathcal{H}}$  denote  $\mathcal{F}$ -behaviors or, more precisely, not all expressions in  $\text{Exp}_{\mathcal{H}}$  are equivalent to  $\mathcal{H}$ -coalgebras that are in the image of  $\alpha \circ (-)$ . Thus, in order to retrieve Kleene's theorem, one has to exclude such expressions. In many situations, this is feasible by simply imposing some syntactic constraints on  $\text{Exp}_{\mathcal{H}}$ .

Let us illustrate this by means of an example. First, we recall the definition of the probability functor (which will play a key role in the next section).

**6.3.2 DEFINITION** (Probability functor). A probability distribution over a set  $S$  is a function  $d: S \rightarrow [0, 1]$  such that  $\sum_{s \in S} d(s) = 1$ . The probability functor  $\mathcal{D}_\omega: \mathbf{Set} \rightarrow \mathbf{Set}$  is defined as follows. For all sets  $S$ ,  $\mathcal{D}_\omega(S)$  is the set of probability distributions over  $S$  with finite support. For all functions  $h: S \rightarrow T$ ,  $\mathcal{D}_\omega(h)$  maps each  $d \in \mathcal{D}_\omega(S)$  into  $d^h$  (Definition 6.1.1).  $\clubsuit$

Note that for any set  $S$ ,  $\mathcal{D}_\omega(S) \subseteq \mathbb{R}^S$  since probability distributions are also functions from  $S$  to  $\mathbb{R}$ . Let  $\iota$  be the family of inclusions  $\iota_S: \mathcal{D}_\omega(S) \rightarrow \mathbb{R}^S$ . It is easy to see that  $\iota$  is a natural transformation between  $\mathcal{D}_\omega$  and  $\mathbb{R}^{\text{Id}}$  (the two functors are defined in the same way on arrows). Thus, in order to specify  $\mathcal{D}_\omega$ -coalgebras, we will use  $\varepsilon \in \text{Exp}_{\mathbb{R}^{\text{Id}}}$ . These are the closed and guarded expressions given by the following BNF, where  $r \in \mathbb{R}$  and  $x \in X$  ( $X$  a set of fixed point variables)

$$\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x. \varepsilon \mid r \cdot \varepsilon$$

This language is enough to specify all  $\mathcal{D}_\omega$ -behaviors, but it also allows us to specify  $\mathbb{R}^{\text{Id}}$ -behaviors that are not  $\mathcal{D}_\omega$ -behaviors, such as for example,  $\mu x. 2 \cdot x$  and  $\mu x. 0 \cdot x$ . In

order to obtain a language  $\text{Exp}_{\mathcal{D}_\omega}$  that specifies all and only the regular  $\mathcal{D}_\omega$ -behaviors, it suffices to change the BNF above as follows:

$$\varepsilon ::= x \mid \mu x. \varepsilon \mid \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i \quad \text{for } p_i \in (0, 1] \text{ such that } \sum_{i \in 1 \dots n} p_i = 1 \quad (6.8)$$

where  $\bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i$  denotes  $p_1 \cdot \varepsilon_1 \oplus \dots \oplus p_n \cdot \varepsilon_n$ .

Next, we prove Kleene's theorem for this restricted syntax. Note that the procedure of appropriately restricting the syntax usually requires some ingenuity. We shall see that in many concrete cases, as for instance  $\mathcal{D}_\omega$  above, it is fairly intuitive which restriction to choose. Also, although we cannot provide a uniform proof of Kleene's theorem, the proof for each concrete example is a slight adaptation of the more general one (Theorem 6.2.9).

### 6.3.3 THEOREM (Kleene's Theorem for the probability functor).

1. For every locally finite  $\mathcal{D}_\omega$ -coalgebra  $(S, d)$  and for every  $s \in S$  there exists an expression  $\varepsilon_s \in \text{Exp}_{\mathcal{D}_\omega}$  such that  $s \sim \varepsilon_s$ .
2. For every  $\varepsilon \in \text{Exp}_{\mathcal{D}_\omega}$ , there exists a finite  $\mathcal{D}_\omega$ -coalgebra  $(S, d)$  with  $s \in S$  such that  $s \sim \varepsilon$ .

PROOF. Let  $s \in S$  and let  $\langle s \rangle = \{s_1, \dots, s_n\}$  with  $s_1 = s$ . We construct, for every state  $s_i \in \langle s \rangle$ , an expression  $\langle\langle s_i \rangle\rangle$  such that  $s_i \sim \langle\langle s_i \rangle\rangle$ .

Let, for every  $i$ ,  $A_i = \mu x_i. \bigoplus_{d(s_i)(s_j) \neq 0} d(s_i)(s_j) \cdot x_j$ .

Now, let  $A_i^0 = A_i$ , define  $A_i^{k+1} = A_i^k \{A_{k+1}^k / x_{k+1}\}$  and then set  $\langle\langle s_i \rangle\rangle = A_i^n$ . Observe that the term

$$A_i^n = (\mu x_i. \bigoplus_{d(s_i)(s_j) \neq 0} d(s_i)(s_j) \cdot x_j) \{A_1^0 / x_1\} \dots \{A_n^{n-1} / x_n\}$$

is a closed term and  $\sum_{d(s_i)(s_j) \neq 0} d(s_i)(s_j) = 1$ . Thus,  $A_i^n \in \text{Exp}_{\mathcal{D}_\omega}$ .

It remains to prove that  $s_i \sim \langle\langle s_i \rangle\rangle$ . We show that  $R = \{\langle s_i, \langle\langle s_i \rangle\rangle \mid s_i \in \langle s \rangle\}$  is a bisimulation. For that we define a function  $\xi: R \rightarrow \mathcal{D}_\omega(R)$  as  $\xi(\langle s_i, - \rangle)(\langle s_j, - \rangle) = d(s_i)(s_j)$  and we observe that the projection maps  $\pi_1$  and  $\pi_2$  are coalgebra homomorphisms, that is, the following diagram commutes.

$$\begin{array}{ccc} \langle s \rangle & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & \{A_i^n \mid s_i \in \langle s \rangle\} \\ d \downarrow & (1) & \downarrow \xi & (2) & \downarrow \delta_{\text{Exp}_{\mathcal{D}_\omega}} \\ \mathcal{D}_\omega(\langle s \rangle) & \xleftarrow{\mathcal{D}_\omega(\pi_1)} & \mathcal{D}_\omega(R) & \xrightarrow{\mathcal{D}_\omega(\pi_2)} & \mathcal{D}_\omega(\{A_i^n \mid s_i \in \langle s \rangle\}) \end{array}$$

$$\mathcal{D}_\omega(\pi_1)(\xi(\langle s_i, A_i^n \rangle))(s_j) = \sum_{\langle s_j, x \rangle \in R} \xi(\langle s_i, A_i^n \rangle)(\langle s_j, x \rangle) = d(s_i)(s_j) \quad (1)$$

$$\mathcal{D}_\omega(\pi_2)(\xi(\langle s_i, A_i^n \rangle))(A_j^n) = \sum_{\langle x, A_j^n \rangle \in R} \xi(\langle s_i, A_i^n \rangle)(\langle x, A_j^n \rangle) = d(s_i)(s_j) = \delta_{\text{Exp}_{\mathcal{D}_\omega}}(A_i^n)(A_j^n) \quad (2)$$

For the second part of the theorem, We need to show that for every expression  $\varepsilon \in \text{Exp}_{\mathcal{D}_\omega}$  there is a finite  $\mathcal{D}_\omega$ -coalgebra  $(S, d)$  with  $s \in S$  such that  $s \sim \varepsilon$ . We take  $(S, d) = \langle \varepsilon \rangle$  and we observe that  $S$  is finite, because  $S \subseteq cl(\varepsilon)$  (the proof of this inclusion is as in Theorem 6.2.9).  $\square$

The axiomatization of  $\text{Exp}_{\mathcal{D}_\omega}$  is a subset of the one for  $\text{Exp}_{\mathbb{R}^{\text{ld}}}$ , since some axioms, such as  $\underline{0} \equiv 0 \cdot \varepsilon$ , have no meaning in the restricted syntax. In this case the axiomatization for  $\text{Exp}_{\mathcal{D}_\omega}$  would contain the axioms for the fixed point, plus (*Associativity*), (*Commutativity*) and  $p \cdot \varepsilon \oplus p' \cdot \varepsilon = (p + p') \cdot \varepsilon$ . The soundness of this axiomatization comes for free from the soundness in  $\mathbb{R}^{\text{ld}}$ , because behavioral equivalence in  $\mathbb{R}^{\text{ld}}$  implies behavioral equivalence in  $\mathcal{D}_\omega$ . For completeness we would have to prove that, for any two expressions in the new syntax, if they are provably equivalent using all the axioms of  $\text{Exp}_{\mathbb{R}^{\text{ld}}}$  then they must be provably equivalent using only the restricted set of axioms. We omit the proof here. Note that it is usually obvious which axioms one needs to keep for the restricted syntax.

For another example, consider the functors  $\text{Id}$  and  $\mathcal{P}_\omega(\text{Id})$ . Let  $\tau$  be the family of functions  $\tau_s: S \rightarrow \mathcal{P}_\omega(S)$  mapping each  $s \in S$  in the singleton set  $\{s\}$ . It is easy to see that  $\tau$  is an injective natural transformation. With the above observation, we can also get regular expressions for the functor  $\mathbb{M}^{\text{ld}} \times \text{Id}^A$  which, as discussed in Section 6.2, does not belong to our class of quantitative functors. Indeed, by extending  $\tau$ , we can construct an injective natural transformation  $\mathbb{M}^{\text{ld}} \times \text{Id}^A \Rightarrow \mathbb{M}^{\text{ld}} \times \mathcal{P}_\omega(\text{Id})^A$ .

In the same way, we can construct an injective natural transformation from the functor  $\mathcal{D}_\omega(\text{Id}) + (A \times \text{Id}) + 1$  (which is the type of stratified systems, which we shall use as an example in the next section) into  $\mathbb{R}^{\text{ld}} + (A \times \mathcal{P}_\omega(\text{Id})) + 1$ . Since the latter is a quantitative functor, we can use its expressions and axiomatization for stratified systems. But since not all its expressions define stratified behaviors, we again will have to restrict the syntax.

## 6.4 Probabilistic systems

Many different types of probabilistic systems have been defined in the literature: examples include reactive, generative, stratified, alternating, (simple) Segala, bundle and Pnueli-Zuck. Each type corresponds to a functor, and the systems of a certain type are coalgebras of the corresponding functor. A systematic study of all these systems as coalgebras was made in [17]. In particular, Figure 1 of [17] provides a full correspondence between types of systems and functors. By employing this correspondence and the results of the previous section, we can use our framework in order to derive regular expressions and axiomatizations for all these types of probabilistic systems.

In order to show the effectiveness of our approach, we have derived expressions and axioms for three different types of probabilistic systems: simple Segala, stratified, and Pnueli-Zuck.

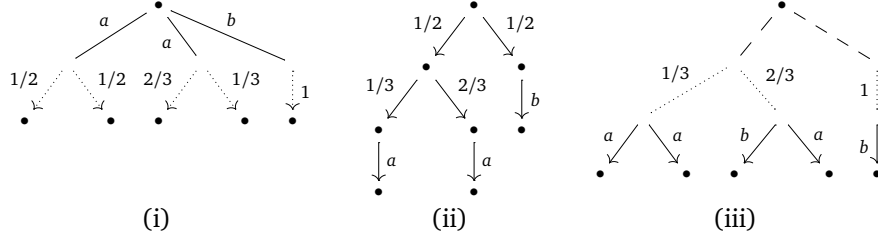


Figure 6.1: (i) A simple Segala system, (ii) a stratified system and (iii) a Pnueli-Zuck system

**Simple Segala systems.** Simple Segala systems are transition systems where both probability and non-determinism are present. They are coalgebras for the functor  $\mathcal{P}_\omega(\mathcal{D}_\omega(\text{Id}))^A$ . Each labeled transition leads, non-deterministically, to a probability distribution of states instead of a single state. An example is shown in Figure 6.1(i). We recall the expressions and axioms for simple Segala systems as shown in Table 6.1.

$$\begin{aligned} \varepsilon &::= \emptyset \mid \varepsilon \boxplus \varepsilon \mid \mu x. \varepsilon \mid x \mid a(\{\varepsilon'\}) & \text{where } a \in A, p_i \in (0, 1] \text{ and } \sum_{i \in 1 \dots n} p_i = 1 \\ \varepsilon' &::= \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon_i \\ (\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 &\equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3) & \varepsilon_1 \boxplus \varepsilon_2 &\equiv \varepsilon_2 \boxplus \varepsilon_1 & \varepsilon \boxplus \emptyset &\equiv \varepsilon & \varepsilon \boxplus \varepsilon &\equiv \varepsilon \\ (\varepsilon'_1 \oplus \varepsilon'_2) \oplus \varepsilon'_3 &\equiv \varepsilon'_1 \oplus (\varepsilon'_2 \oplus \varepsilon'_3) & \varepsilon'_1 \oplus \varepsilon'_2 &\equiv \varepsilon'_2 \oplus \varepsilon'_1 & (p_1 \cdot \varepsilon) \oplus (p_2 \cdot \varepsilon) &\equiv (p_1 + p_2) \cdot \varepsilon \\ \varepsilon[\mu x. \varepsilon/x] &\equiv \mu x. \varepsilon & \gamma[\varepsilon/x] &\equiv \varepsilon \Rightarrow \mu x. \gamma & \equiv \varepsilon \end{aligned}$$

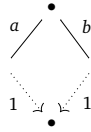
Here, in order to avoid confusion, we use  $\boxplus$  instead of  $\oplus$  for the expressions at the top level, making a clear distinction between the idempotent (non-deterministic) and non-idempotent (probabilistic) sums. The syntax above was obtained from the canonically derived one by applying the restrictions arising from  $\mathcal{D}_\omega$  and also some simplifications (syntactic sugar) which improve readability (in the spirit of what we showed before for  $\text{Exp}_{\mathcal{W}}$ , the expressions for weighted automata).

As we showed in section 6.3, to be completely formal we would have to prove Kleene's Theorem and the completeness of the axiomatization for the restricted syntax (as well as the correctness of the simplifications). The proofs would be based on the ones we showed for the functor  $\mathcal{D}_\omega$  (and the ones for the simplifications similar to what we showed for  $\text{Exp}_{\mathcal{W}}$ ). We omit them here and show instead an example. The expression  $a(\{1/2 \cdot \emptyset \oplus 1/2 \cdot \emptyset\}) \boxplus a(\{1/3 \cdot \emptyset \oplus 2/3 \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\})$  is bisimilar to the top-most state in the simple Segala system depicted in Figure 6.1(i). Using the axiomatization, we

can derive:

$$\begin{aligned}
& a(\{1/2 \cdot \emptyset \oplus 1/2 \cdot \emptyset\}) \boxplus a(\{1/3 \cdot \emptyset \oplus 2/3 \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\}) \\
\equiv & a(\{(1/2 + 1/2) \cdot \emptyset\}) \boxplus a(\{(1/3 + 2/3) \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\}) \\
\equiv & a(\{1 \cdot \emptyset\}) \boxplus a(\{1 \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\}) \\
\equiv & a(\{1 \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\})
\end{aligned}$$

Thus, we can conclude that the system presented in Figure 6.1(i) is bisimilar to the following one:



The language and axiomatization we presented above are the same as the ones presented in [42] (with the slight difference that in [42] a parallel composition operator was also considered). This is of course reassuring for the correctness of the general framework we presented. In the next two examples, we will present new results (that is syntax/axiomatizations which did not exist). This is where the generality starts paying off: not only one recovers known results but also derives new ones, all of this inside the same uniform framework.

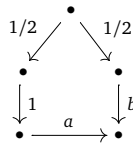
**Stratified systems.** Stratified systems are coalgebras for the functor  $\mathcal{D}_\omega(\text{Id}) + (\mathbb{B} \times \text{Id}) + 1$ . Each state of these systems either performs unlabeled probabilistic transitions or one  $\mathbb{B}$ -labeled transition or it terminates. We first derive expressions and axioms for  $\mathbb{R}^{\text{Id}} + (\mathbb{B} \times \mathcal{P}_\omega(\text{Id})) + 1$  and then we restrict the syntax to characterize only  $\mathcal{D}_\omega(\text{Id}) + (\mathbb{B} \times \text{Id}) + 1$ -behaviors. This, together with the introduction of some syntactic sugar, leads to the following syntax and axioms.

$$\varepsilon ::= \mu x. \varepsilon \mid x \mid \langle b, \varepsilon \rangle \mid \bigoplus_{i=1 \dots n} p_i \cdot \varepsilon_i \mid \downarrow \quad \text{where } b \in \mathbb{B}, p_i \in (0, 1] \text{ and } \sum_{i=1 \dots n} p_i = 1$$

$$\begin{aligned}
(\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 &\equiv \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) & \varepsilon_1 \oplus \varepsilon_2 &\equiv \varepsilon_2 \oplus \varepsilon_1 & (p_1 \cdot \varepsilon) \oplus (p_2 \cdot \varepsilon) &\equiv (p_1 + p_2) \cdot \varepsilon \\
\varepsilon[\mu x. \varepsilon/x] &\equiv \mu x. \varepsilon & \gamma[\varepsilon/x] &\equiv \varepsilon \Rightarrow \mu x. \gamma & \equiv \varepsilon
\end{aligned}$$

Here  $\downarrow$ , which denotes termination, corresponds to the canonically derived expression  $r[r[1]]$ , while  $\langle b, \varepsilon \rangle$  corresponds to  $r[l[l\langle b \rangle \oplus r\langle \{\varepsilon\} \rangle]]$ .

We can use these axioms (together with Kleene's theorem) to reason about the system presented in Figure 6.1(ii). The topmost state of this system is bisimilar to the expression  $1/2 \cdot (1/3 \cdot \langle a, \downarrow \rangle \oplus 2/3 \cdot \langle a, \downarrow \rangle) \oplus 1/2 \cdot \langle b, \downarrow \rangle$ , which in turn is provably equivalent to  $1/2 \cdot (1 \cdot \langle a, \downarrow \rangle) \oplus 1/2 \cdot \langle b, \downarrow \rangle$ . That leads us to conclude that the aforementioned system is equivalent to the following simpler one.



The language of expressions we propose for these systems is a subset of the language originally proposed in [117] (there a parallel composition operator is also considered). More interestingly, there was no axiomatization of the language in [117] and thus the axiomatization we present here is completely new.

**Pnueli-Zuck systems.** These systems are coalgebras for the functor  $\mathcal{P}_\omega \mathcal{D}_\omega(\mathcal{P}_\omega(\text{Id})^A)$ . Intuitively, the ingredient  $\mathcal{P}_\omega(\text{Id})^A$  denotes  $A$ -labeled transitions to other states. Then,  $\mathcal{D}_\omega(\mathcal{P}_\omega(\text{Id})^A)$  corresponds to a probability distribution of labeled transitions and finally each state of a  $\mathcal{P}_\omega \mathcal{D}_\omega(\mathcal{P}_\omega(\text{Id})^A)$ -coalgebra performs a non-deterministic choice amongst probability distributions of labeled transitions. For an example, consider the system depicted in Figure 6.1(iii).

The expressions and axioms for these systems are the following.

$$\begin{aligned} \varepsilon &::= \emptyset \mid \varepsilon \boxplus \varepsilon \mid \mu x. \varepsilon \mid x \mid \{\varepsilon'\} & \text{where } a \in A, p_i \in (0, 1] \text{ and } \sum_{i \in 1 \dots n} p_i = 1 \\ \varepsilon' &::= \bigoplus_{i \in 1 \dots n} p_i \cdot \varepsilon'_i \\ \varepsilon'' &::= \underline{\emptyset} \mid \varepsilon'' \boxplus \varepsilon'' \mid a(\{\varepsilon\}) \end{aligned}$$

$$\begin{aligned} (\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 &\equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3) & \varepsilon_1 \boxplus \varepsilon_2 &\equiv \varepsilon_2 \boxplus \varepsilon_1 & \varepsilon \boxplus \emptyset &\equiv \varepsilon & \varepsilon \boxplus \varepsilon &\equiv \varepsilon \\ (\varepsilon'_1 \oplus \varepsilon'_2) \oplus \varepsilon'_3 &\equiv \varepsilon'_1 \oplus (\varepsilon'_2 \oplus \varepsilon'_3) & \varepsilon'_1 \oplus \varepsilon'_2 &\equiv \varepsilon'_2 \oplus \varepsilon'_1 & (p_1 \cdot \varepsilon'') \oplus (p_2 \cdot \varepsilon'') &\equiv (p_1 + p_2) \cdot \varepsilon'' \\ (\varepsilon''_1 \boxplus \varepsilon''_2) \boxplus \varepsilon''_3 &\equiv \varepsilon''_1 \boxplus (\varepsilon''_2 \boxplus \varepsilon''_3) & \varepsilon''_1 \boxplus \varepsilon''_2 &\equiv \varepsilon''_2 \boxplus \varepsilon''_1 & \underline{\emptyset} \boxplus \varepsilon'' &\equiv \varepsilon'' & \varepsilon'' \boxplus \varepsilon'' &\equiv \varepsilon'' \\ \varepsilon[\mu x. \varepsilon/x] &\equiv \mu x. \varepsilon & \gamma[\varepsilon/x] &\equiv \varepsilon \Rightarrow \mu x. \gamma & \equiv \varepsilon \end{aligned}$$

The expression  $\{1/3 \cdot (a(\{\emptyset\}) \boxplus a(\{\emptyset\})) \oplus 2/3 \cdot (b(\{\emptyset\}) \boxplus a(\{\emptyset\}))\} \boxplus \{1 \cdot b(\{\emptyset\})\}$  specifies the Pnueli-Zuck system in Figure 6.1(iii). Note that we use the same symbol  $\boxplus$  for denoting two different kinds of non-deterministic choice. This is safe, since they satisfy exactly the same axioms.

Both the syntax and the axioms we propose here for these systems are to the best of our knowledge new. In the past, these systems have been studied using a temporal logic [91].

## 6.5 A slight variation on the functor

In this section, we show a slight variation on the definition of the monoidal exponentiation functor which would allow for a cleaner derivation of syntax and axioms for certain functors, among which the probability functor.

In the spirit of [65], where this functor is defined, we shall call it constrained monoidal exponentiation functor.

**6.5.1 DEFINITION** (Constrained monoidal exponentiation functor). Let  $\mathbb{M}$  be a commutative monoid and  $V \subseteq \mathbb{M}$  ( $V$  being the constraint). The constrained monoidal exponentiation functor  $\mathbb{M}_V^- : \mathbf{Set} \rightarrow \mathbf{Set}$  is defined on sets as  $\mathbb{M}_V^S = \{f \in \mathbb{M}_\omega^S \mid \sum_{s \in S} f(s) \in V\}$  and on functions as  $\mathbb{M}_\omega^-$ . ♣

The probability functor  $\mathcal{D}_\omega$  coincides now with  $(\mathbb{R}_0^+)^-_{\{1\}}$  ( $\mathbb{R}_0^+$  denotes the monoid of positive real numbers).

The expressions associated with this functor are the closed and guarded expressions given by the following BNF, where  $x \in X$  and  $m_i \in \mathbb{M}$ ,

$$\text{Exp}_{\mathbb{M}_V^-} \ni \varepsilon ::= x \mid \mu x. \varepsilon \mid \bigoplus_{i \in I} m_i \cdot \varepsilon_i \text{ such that } \sum_{i \in I} m_i \in V$$

We note that instantiating this syntax for  $(\mathbb{R}_0^+)^-_{\{1\}}$ , one gets precisely the syntax we proposed in equation (6.8) for  $\mathcal{D}_\omega$ .

Providing a Kleene like theorem and a sound and complete axiomatization goes precisely as before (for the functor  $\mathbb{M}_\omega^-$ ), with the minor difference that the axiom  $0 \cdot \varepsilon \equiv \emptyset$  has to be replaced by  $0 \cdot \varepsilon \oplus m \cdot \varepsilon' \equiv m \cdot \varepsilon'$ , since  $\emptyset$  is not a valid expression for this functor.

All of this seems to indicate that using the constrained monoidal exponentiation functor would have made it easier to define expressions and axiomatizations for quantitative systems. Although this would have been true for systems such as the simple Segala, it would not completely avoid the use of the technique we described in Section 6.3, which allows us to deal with a large class of functors: not only with  $\mathcal{D}_\omega$  (embedded into  $\mathbb{R}^{\text{d}}$ ) but also with mixed functors, such the one of stratified systems. Moreover, using this functor would require extra care when dealing with the modularity of the axiomatization, which we will illustrate next by means of an example. For these reasons, we decided to present the syntax and axiomatizations of all our running examples as a special instance of the general technique described in Section 6.3.

**6.5.2 EXAMPLE (Reactive probabilistic automata).** Let us consider a very simple version of reactive probabilistic automata, coalgebras for the functor  $\mathcal{D}_\omega(-)^A = ((\mathbb{R}_0^+)^{\text{d}}_{\{1\}})^A$ . The syntax modularly derived for this functor would be

$$\begin{aligned} \varepsilon &::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x. \varepsilon \mid a(\varepsilon') \\ \varepsilon' &::= \bigoplus_{i \in I} p_i \cdot \varepsilon_i \end{aligned} \quad \text{such that } \sum_{i \in I} p_i = 1$$

In the axiomatization, we would (expect to) have the axiom  $a(\varepsilon_1) \oplus a(\varepsilon_2) \equiv a(\varepsilon_1 \oplus \varepsilon_2)$ . But now note how this could lead to an inconsistent specification, since  $\varepsilon_1 \oplus \varepsilon_2$  will not be a valid expression anymore for  $\mathcal{D}_\omega$  (if  $\sum p_i = 1$  in both  $\varepsilon_1$  and  $\varepsilon_2$  then it will be 2 in  $\varepsilon_1 \oplus \varepsilon_2$ !).

In order to keep the axiomatization compositional we would have to require certain conditions on the set  $V$  in the functor  $\mathbb{M}_V^-$ . For instance, one of the possible conditions would be that  $V$  would have to be closed with respect to  $+$ , which would be a too strong condition to model  $\mathcal{D}_\omega$ .  $\spadesuit$

## 6.6 Discussion

In this chapter, we presented a general framework to canonically derive expressions and axioms for quantitative regular behaviors. To illustrate the effectiveness and gen-

erality of our approach we derived expressions and equations for weighted automata, simple Segala, stratified and Pnueli-Zuck systems.

We recovered the syntaxes proposed in [31, 42, 117] for the first three models and the axiomatization of [42]. For weighted automata and stratified systems we derived new axiomatizations and for Pnueli-Zuck systems both a novel language of expressions and axioms. The process calculi in [31, 42, 117] also contained a parallel composition operator and thus they slightly differ from our languages that are more in the spirit of Kleene and Milner's expressions. In order to obtain a language, based on the one we defined in this chapter, which also includes other (user-defined) operators, such as parallel composition, we would like to study the connection with bialgebras and GSOS.

In [15, 41, 110] expressions without parallel composition are studied for probabilistic systems. These provide syntax and axioms for generative systems, Segala systems and alternating systems, respectively. For Segala systems our approach will derive the same language of [41], while the expressions in [110] differ from the ones resulting from our approach, since they use a probabilistic choice operator  $+_p$ . For alternating systems, our approach could bring some new insights, since in [15] only expressions without recursion are considered.

The interplay between non-determinism and probabilities, present in some models, such as Segala or Pnueli-Zuck, is usually a source of challenges when it comes to define a process calculi and axiomatization. We note that in our framework this interplay is just functor composition and thus the derivation of expressions and accompanying axioms follows in the same canonical way as for any other functor.

The derivation of the syntax and axioms associated with each quantitative functor is in the process of being implemented in the coinductive prover CIRC [78]. For the non-deterministic fragment everything can be done automatically, whereas for the functors described in Section 6.3, such as the probability functor, some user input is required, in order to define the syntactic restrictions. This will then allow for automatic reasoning about the equivalence of expressions specifying systems.



There are several ways of extending the work presented in this thesis. In this chapter, we mention some of the most promising topics for future research.

**Additional operators** The languages we associated with each functor are minimal in the sense that they have only the necessary operators to describe regular behaviours. Many process calculi have additional operators, such as parallel composition which, when added freely in the syntax, can give rise to non-regular behaviours (for instance, one could describe context-free languages). This leads to two interesting questions:

1. Is it possible to add operators, such as parallel composition, in a safe manner to the language? That is, can we define and axiomatize additional operators, in a modular fashion, in order to remain in the world of regular behaviours?
2. Is it possible to take all the research in this thesis to a different dimension and consider non-regular behaviours? For instance, can we define a coalgebraic notion of context-free behaviours?

A promising research path in this context is to study the languages associated with each functor from a bialgebraic perspective [61, 113].

**Additional systems** Enriching the class of systems is a natural research direction. Variations on the base category of the functors can lead to the treatment of various systems. Recently, Milius [82] showed how to extend our work for a particular functor in the category of vector spaces, deriving expressions for stream circuits. It is a challenging question whether the language he proposed can be compositionally extended to other functors on vector spaces or even to functors on other categories, such as metric spaces [73, 114, 115]. The latter are of particular interest in the context of quantitative modelling and verification.

**Additional equivalences** The axiomatizations presented in Chapters 4 to 6 are sound and complete with respect to behavioural equivalence. It is an interesting research path to provide sound and complete axiomatizations for other equivalences, such as trace or simulation equivalences. A coalgebraic theory of traces has been presented by Hasuo, Jacobs and Sokolova in [57] and first steps towards having a generic notion of coalgebraic simulation have been taken by Jacobs and Hugues in [59] and Hasuo in [56].

**Coalgebraic  $\mu$ -calculus** The languages presented in this thesis resemble a fragment of the coalgebraic  $\mu$ -calculus [35]. In this thesis, we provided the expressions with a final semantics. Alternatively, one can also associate a modal semantics to the expressions, that is, consider a relation from expressions to the states of a coalgebra instead of a function. We have done this for the particular case of the Mealy expressions [26] and showed how both semantics relate. It is an interesting research question to investigate further the connections with the coalgebraic modal  $\mu$ -calculus.

**Rational behaviours** Expressions to describe rational behaviours of infinite streams have been presented by Rutten in [98] and for infinite binary trees by Silva and Rutten in [107, 108]. Rational streams (respectively trees or, more generally, formal power series) are streams which can be represented by a finite weighted automaton. Instantiating our framework for the stream functor yields a framework where only regular streams, that is streams with a finite number of sub-streams, can be represented. For instance, the stream of natural numbers  $(1, 2, 3, \dots)$  is rational but not regular. Rutten showed recently [100] that rational streams are represented by finitely dimensional linear systems (which are coalgebras over vector spaces). Milius [82] explored this fact to derive expressions for stream circuits, which we mentioned above.

A possible research question is what would be a general coalgebraic notion of rational behaviours and whether it is possible to define rational expressions for a larger class of functors.

**Practical specification of systems** Many operational semantics of languages are given in terms of transition systems. For instance, for the coordination language REO [10], Bonsangue, Clarke and Silva [23] proposed an automata model, which fits in the framework presented in this thesis. It is a promising research direction to investigate whether the expressions associated with the aforementioned automaton type can be used to specify and synthesize REO circuits.

Reasoning (automatically) about specifications of systems is also of uttermost importance. The framework presented in this thesis is currently being implemented in the automatic theorem prover CIRC [22] and we hope to be able to use the tool to automatically decide on the equivalence of specifications.

---

## Bibliography

- [1] Luca Aceto, Zoltán Ésik, and Anna Ingólfssdóttir. Equational axioms for probabilistic bisimilarity. In Hélène Kirchner and Christophe Ringeissen, editors, *AMAST*, volume 2422 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2002. pages 111
- [2] Luca Aceto and Matthew Hennessy. Termination, deadlock, and divergence. *J. ACM*, 39(1):147–187, 1992. pages 90, 102
- [3] Peter Aczel and Nax Paul Mendler. A final coalgebra theorem. In David H. Pitt, David E. Rydeheard, Peter Dybjer, Andrew M. Pitts, and Axel Poigné, editors, *Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 1989. pages 11
- [4] Jirí Adámek, Dominik Lücke, and Stefan Milius. Recursive coalgebras of finitary functors. *ITA*, 41(4):447–462, 2007. pages 104
- [5] Jirí Adámek, Stefan Milius, and Jiri Velebil. Free iterative theories: A coalgebraic view. *Mathematical Structures in Computer Science*, 13(2):259–320, 2003. pages 7, 108
- [6] Jirí Adámek, Stefan Milius, and Jiri Velebil. Iterative algebras at work. *Mathematical Structures in Computer Science*, 16(6):1085–1131, 2006. pages 7, 108
- [7] Roberto M. Amadio, editor. *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*. Springer, 2008. pages 145, 146

- [8] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Computing and Information Science, Cornell University, July 2001. pages 39
- [9] Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. pages 27
- [10] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004. pages 142
- [11] Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *LICS*, pages 160–171, 1997. pages 8
- [12] Eugene Asarin and Catalin Dima. Balanced timed regular expressions. *Electr. Notes Theor. Comput. Sci.*, 68(5), 2002. pages 8
- [13] Jos C. M. Baeten, Jan A. Bergstra, and Scott A. Smolka. Axiomization probabilistic processes: Acp with generative probabilities (extended abstract). In *Cleaveland [38]*, pages 472–485. pages 111
- [14] Jos C. M. Baeten and Jan Willem Klop, editors. *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*. Springer, 1990. pages 148, 151
- [15] Emanuele Bandini and Roberto Segala. Axiomatizations for probabilistic bisimulation. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 370–381. Springer, 2001. pages 111, 140
- [16] Falk Bartels. *On generalized coinduction and probabilistic specification formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004. PhD thesis. pages 28
- [17] Falk Bartels, Ana Sokolova, and Erik P. de Vink. A hierarchy of probabilistic system types. *Theor. Comput. Sci.*, 327(1-2):3–22, 2004. pages 132, 135
- [18] Gérard Berry. The foundations of Esterel. In Plotkin et al. [89], pages 425–454. pages 2
- [19] Gérard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(3):117–126, 1986. pages 2, 27, 28, 29, 30
- [20] S.L. Bloom and Z. Ésik. *Iteration theories: the equational logic of iterative processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993. pages 7, 108

- [21] Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Deriving syntax and axioms for quantitative regular behaviours. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2009. pages 108
- [22] Marcello M. Bonsangue, Georgiana Caltais, Eugen Goriac, Dorel Lucanu, Jan J. M. M. Rutten, and Alexandra Silva. Algebra meets coalgebra: A decision procedure for bisimilarity of generalized regular expressions. draft. pages 142
- [23] Marcello M. Bonsangue, Dave Clarke, and Alexandra Silva. Automata for context-dependent connectors. In John Field and Vasco Thudichum Vasconcelos, editors, *COORDINATION*, volume 5521 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2009. pages 142
- [24] Marcello M. Bonsangue and Alexander Kurz. Duality for logics of transition systems. In Sassone [102], pages 455–469. pages 7, 68, 108
- [25] Marcello M. Bonsangue and Alexander Kurz. Presenting functors by operations and equations. In Luca Aceto and Anna Ingólfssdóttir, editors, *FoSSaCS*, volume 3921 of *LNCS*, pages 172–186. Springer, 2006. pages 7, 68, 108
- [26] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Coalgebraic logic and synthesis of Mealy machines. In Amadio [7], pages 231–245. pages 59, 142
- [27] Patricia Bouyer and Antoine Petit. A Kleene/büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 7(2):167–186, 2002. pages 8
- [28] Janusz A. Brzozowski. A survey of regular expressions and their applications. *IRE Transactions on Electronic Computers*, 11(0):324–335, 1962. pages 2, 48
- [29] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964. pages 2, 7, 15, 19, 23, 33, 54, 62, 68, 74, 77, 98, 130
- [30] Peter Buchholz. Bisimulation relations for weighted automata. *Theor. Comput. Sci.*, 393(1-3):109–123, 2008. pages 112, 117, 118
- [31] Peter Buchholz and Peter Kemper. Quantifying the dynamic behavior of process algebras. In Luca de Alfaro and Stephen Gilmore, editors, *PAPM-PROBMIV*, volume 2165 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2001. pages 115, 123, 140
- [32] J. R. Büchi. On a decision method in restricted second order arithmetic. In *In Proceedings of the International Congress on Logic, Method, and Philosophy of Science*, pages 1–12. Stanford University Press, Stanford, CA, 1962. pages 7

- [33] Jean-Marc Champarnaud, Florent Nicart, and Djelloul Ziadi. From the ZPC structure of a regular expression to its follow automaton. *IJAC*, 16(1):17–34, 2006. pages 30
- [34] Jean-Marc Champarnaud and Djelloul Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theor. Comput. Sci.*, 289(1):137–163, 2002. pages 27
- [35] Corina Cîrstea, Clemens Kupke, and Dirk Pattinson. Exptime tableaux for the coalgebraic micro;-calculus. In Erich Grädel and Reinhard Kahle, editors, *CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2009. pages 142
- [36] Corina Cîrstea and Dirk Pattinson. Modular construction of modal logics. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2004. pages 7, 108
- [37] Edmund M. Clarke, Steven M. German, Yuan Lu, Helmut Veith, and Dong Wang. Executable protocol specification in ESL. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 197–216. Springer, 2000. pages 48, 68
- [38] Rance Cleaveland, editor. *CONCUR '92, Third International Conference on Concurrency Theory, Stony Brook, NY, USA, August 24-27, 1992, Proceedings*, volume 630 of *Lecture Notes in Computer Science*. Springer, 1992. pages 144, 149
- [39] J.H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971. pages 2, 15, 23
- [40] Pedro R. D’Argenio, Holger Hermanns, and Joost-Pieter Katoen. On generative parallel composition. *Electr. Notes Theor. Comput. Sci.*, 22, 1999. pages 111
- [41] Yuxin Deng and Catuscia Palamidessi. Axiomatizations for probabilistic finite-state behaviors. In Sassone [102], pages 110–124. pages 3, 111, 140
- [42] Yuxin Deng, Catuscia Palamidessi, and Jun Pang. Compositional reasoning for probabilistic finite-state behaviors. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles*, volume 3838 of *Lecture Notes in Computer Science*, pages 309–337. Springer, 2005. pages 3, 111, 112, 137, 140
- [43] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005. pages 111, 115, 117
- [44] Manfred Droste and Karin Quaas. A Kleene-Schützenberger theorem for weighted timed automata. In Amadio [7], pages 142–156. pages 8

- [45] C.C. Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*. North-Holland Publishers, 1975. pages 7
- [46] Paul Gastin, Antoine Petit, and Wiesław Zielonka. A Kleene theorem for infinite trace languages. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 1991. pages 8
- [47] Paul Gastin, Antoine Petit, and Wiesław Zielonka. An extension of Kleene’s and Ochmanski’s theorems to infinite traces. *Theor. Comput. Sci.*, 125(2):167–204, 1994. pages 8
- [48] Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B Jones, editors, *Proc. of Working Conference on Programming Concepts and Methods*. IFIP TC 2, 1990. pages 111
- [49] V.M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961. pages 27
- [50] Robert Goldblatt. Equational logic of polynomial coalgebras. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyashev, editors, *Advances in Modal Logic 4*, pages 149–184. King’s College Publications, 2002. pages 7, 108
- [51] H. Peter Gumm and Tobias Schröder. Monoid-labeled transition systems. *Electr. Notes Theor. Comput. Sci.*, 44(1), 2001. pages 114, 115
- [52] H. Peter Gumm and Tobias Schröder. Products of coalgebras. *Algebra Universalis*, 46:163–185, 2001. pages 115
- [53] H. Peter Gumm and Tobias Schröder. Coalgebras of bounded type. *Mathematical Structures in Computer Science*, 12(5):565–578, 2002. pages 11, 114
- [54] Helle Hvid Hansen, David Costa, and Jan J. M. M. Rutten. Synthesis of Mealy machines using derivatives. *Electronic Notes in Theoretical Computer Science*, 164(1):27–45, 2006. pages 68
- [55] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994. pages 111
- [56] Ichiro Hasuo. Generic forward and backward simulations. In Christel Baier and Holger Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2006. pages 142
- [57] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007. pages 142

- [58] Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. Comput.*, 145(2):107–152, 1998. pages 12
- [59] Jesse Hughes and Bart Jacobs. Simulations in coalgebra. *Theor. Comput. Sci.*, 327(1-2):71–108, 2004. pages 142
- [60] Bart Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *ITA*, 35(1):31–59, 2001. pages 7, 108
- [61] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006. pages 7, 15, 28, 36, 37, 108, 141
- [62] Chi-Chang Jou and Scott A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [14], pages 367–383. pages 111
- [63] Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3(4):361–386, 1969. pages 39
- [64] Stephen Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956. pages 2, 47, 68, 69
- [65] Bartek Klin. Structural operational semantics for weighted transition systems. In Jens Palsberg, editor, *Semantics and Algebraic Specification*, volume 5700 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2009. pages 138
- [66] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Logic in Computer Science*, pages 214–225, 1991. pages 2, 7, 35, 47, 69
- [67] Dexter Kozen. *Automata and Computability*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. pages 22, 23, 27, 32
- [68] Dexter Kozen. Myhill-nerode relations on automatic systems and the completeness of Kleene algebra. In Afonso Ferreira and Horst Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2001. pages 15, 35, 36, 37
- [69] Dexter Kozen. Automata on guarded strings and applications. *Matemática Contemporânea*, 24:117–139, 2003. pages 3, 7, 8, 15, 40, 41, 42
- [70] Dexter Kozen. Nonlocal flow of control and Kleene algebra with tests. In *LICS*, pages 105–117. IEEE Computer Society, 2008. pages 15, 39



- [71] Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report <http://hdl.handle.net/1813/10173>, Computing and Information Science, Cornell University, March 2008. pages 41, 103, 108
- [72] Dexter Kozen and Maria-Christina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 568–582. Springer, 2000. pages 39
- [73] Dexter Kozen and Nicholas Ruoizzi. Applications of metric coinduction. *Logical Methods in Computer Science*, 5(3), 2009. pages 108, 141
- [74] Orna Kupferman and Moshe Y. Vardi.  $\mu$ -calculus synthesis. In Mogens Nielsen and Branislav Rován, editors, *MFCS'00*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2000. pages 68
- [75] Clemens Kupke and Yde Venema. Coalgebraic automata theory: Basic results. *Logical Methods in Computer Science*, 4(4), 2008. pages 7, 68, 108
- [76] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991. pages 111
- [77] Kim Guldstrand Larsen and Arne Skou. Compositional verification of probabilistic processes. In Cleaveland [38], pages 456–471. pages 111
- [78] 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. pages 109, 140
- [79] Alan B. Marcovitz. *Introduction to Logic Design*. McGraw-Hill, 2005. pages 47
- [80] Robert McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, 9(0):39–47, 1960. pages 2, 27, 48
- [81] G.H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34:1045–1079, 1955. pages 3
- [82] Stefan Milius. A sound and complete calculus for finite stream circuits. In *LICS*, 2010. To appear. pages 108, 141, 142
- [83] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. pages 3
- [84] Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. pages 3, 69, 73

- [85] Michael W. Mislove, Joël Ouaknine, and James Worrell. Axioms for probability and nondeterminism. *Electr. Notes Theor. Comput. Sci.*, 96:7–28, 2004. pages 111
- [86] Larry Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96, 1999. pages 7, 108
- [87] Edward Ochmanski. Regular behaviour of concurrent systems. *Bulletin of the EATCS*, 27:56–67, 1985. pages 7
- [88] Scott Owens, John H. Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, 2009. pages 27
- [89] Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors. *Proof, Language, and Interaction, Essays in Honour of Robin Milner*. The MIT Press, 2000. pages 144, 151
- [90] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL'89*, pages 179–190, 1989. pages 68
- [91] Amir Pnueli and Lenore D. Zuck. Probabilistic verification by tableaux. In *LICS*, pages 322–331. IEEE Computer Society, 1986. pages 111, 138
- [92] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. pages 111
- [93] Michael O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959. pages 27
- [94] Martin Rößiger. Coalgebras and modal logic. *Electronic Notes in Theoretical Computer Science*, 33, 2000. pages 7, 108
- [95] Jan J. M. M. 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. pages 6, 15, 68, 108
- [96] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. pages 3, 10, 11, 13, 28, 70, 72, 117, 129, 132
- [97] Jan J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003. pages 6, 15, 34, 51
- [98] Jan J. M. M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005. pages 142
- [99] Jan J. M. M. Rutten. Algebraic specification and coalgebraic synthesis of Mealy automata. *Electr. Notes Theor. Comput. Sci.*, 160:305–319, 2006. pages 51, 68

- [100] Jan J. M. M. Rutten. Rational streams coalgebraically. *Logical Methods in Computer Science*, 4(3), 2008. pages 142
- [101] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. pages 2, 21, 47
- [102] Vladimiro Sassone, editor. *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3441 of *Lecture Notes in Computer Science*. Springer, 2005. pages 145, 146
- [103] Lutz Schröder and Dirk Pattinson. Modular algorithms for heterogeneous modal logics. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 459–471. Springer, 2007. pages 7, 108
- [104] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961. pages 3, 8, 111, 115, 117
- [105] Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, MIT, Dept. of EECS, 1995. pages 111
- [106] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994. pages 3, 111
- [107] Alexandra Silva and Jan J. M. M. Rutten. Behavioural differential equations and coinduction for binary trees. In Daniel Leivant and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 4576 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2007. pages 142
- [108] Alexandra Silva and Jan J. M. M. Rutten. A coinductive calculus of binary trees. *Inf. Comput.*, 208(5):578–593, 2010. pages 142
- [109] Scott A. Smolka and Bernhard Steffen. Priority as extremal probability. In Baeten and Klop [14], pages 456–466. pages 111
- [110] Eugene W. Stark and Scott A. Smolka. A complete axiom system for finite-state probabilistic processes. In Plotkin et al. [89], pages 571–596. pages 111, 140
- [111] Ken Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. pages 27
- [112] Simone Tini and Andrea Maggiolo-Schettini. Compositional synthesis of generalized Mealy machines. *Fundamenta Informaticae*, 60(1-4):367–382, 2004. pages 68

- [113] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291, 1997. pages 141
- [114] Daniele Turi and Jan J. M. M. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998. pages 108, 141
- [115] Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006. pages 108, 141
- [116] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995. pages 109
- [117] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995. pages 111, 112, 138, 140
- [118] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338. IEEE, 1985. pages 111