Deriving Syntax and Axioms for Quantitative Regular Behaviours*

Filippo Bonchi[2], Marcello Bonsangue[1,2], Jan Rutten[2,3], and Alexandra Silva[2]

[1] LIACS - Leiden University
[2] Centrum voor Wiskunde en Informatica (CWI)
[3] Vrije Universiteit Amsterdam (VUA)

Abstract. We present a systematic way to generate (1) languages of (generalised) regular expressions, and (2) sound and complete axiomatizations thereof, for a wide variety of quantitative systems. Our quantitative systems include weighted versions of automata and transition systems, in which transitions are assigned a value in a monoid that represents cost, duration, probability, etc. Such systems are represented as coalgebras and (1) and (2) above are derived in a modular fashion from the underlying (functor) type of these coalgebras.

In previous work, we applied a similar approach to a class of systems (without weights) that generalizes both the results of Kleene (on rational languages and DFA's) and Milner (on regular behaviours and finite LTS's), and includes many other systems such as Mealy and Moore machines.

In the present paper, we extend this framework to deal with quantitative systems. As a consequence, our results now include languages and axiomatizations, both existing and new ones, for many different kinds of probabilistic systems.

1 Introduction

Kleene's Theorem [22] gives a fundamental correspondence between *regular expressions* and *deterministic finite automata* (DFA's): each regular expression denotes a language that can be recognized by a DFA and, conversely, the language accepted by a DFA can be specified by a regular expression. Languages denoted by regular expressions are called *regular*. Two regular expressions are (language) equivalent if they denote the same regular language. Salomaa [32] presented a sound and complete axiomatization (later refined by Kozen in [23]) for proving the equivalence of regular expressions.

The above programme was applied by Milner in [26] to process behaviours and labelled transition systems (LTS's). Milner introduced a set of expressions for finite LTS's and proved an analogue of Kleene's Theorem: each expression denotes the behaviour of a finite LTS and, conversely, the behaviour of a finite LTS can be specified by an expression. Milner also provided an axiomatization for his expressions, with the property that two expressions are provably equivalent if and only if they are bisimilar.

Coalgebras provide a general framework for the study of dynamical systems such as DFA's and LTS's. For a functor $G\colon \mathbf{Set} \to \mathbf{Set}$, a G-coalgebra or G-system is a pair

* This work was carried out during the first author's tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The fourth author is partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

M. Bravetti and G. Zavattaro (Eds.): CONCUR 2009, LNCS 5710, pp. 146–162, 2009.

(S, g), consisting of a set S of states and a function $g{:}S \to GS$ defining the "transitions" of the states. We call the functor G the *type* of the system. For instance, DFA's can be readily seen to correspond to coalgebras of the functor $G(S) = 2 \times S^A$ and image-finite LTS's are obtained by $G(S) = \mathcal{P}_\omega(S)^A$, where $\mathcal{P}_\omega$ is finite powerset.

Under mild conditions, functors G have a *final coalgebra* (unique up to isomorphism) into which every G-coalgebra can be mapped via a unique so-called *G-homomor-phism*. The final coalgebra can be viewed as the universe of all possible *G-behaviours*: the unique homomorphism into the final coalgebra maps every state of a coalgebra to a canonical representative of its behaviour. This provides a general notion of behavioural equivalence: two states are equivalent iff they are mapped to the same element of the final coalgebra. In the case of DFA's, two states are equivalent when they accept the same language; for LTS's, behavioural equivalence coincides with bisimilarity.

For coalgebras of a large but restricted class of functors, we introduced in [7] a language of regular expressions; a corresponding generalisation of Kleene's Theorem; and a sound and complete axiomatization for the associated notion of behavioural equivalence. We derived both the language of expressions and their axiomatization, in a modular fashion, from the functor defining the type of the system.

In recent years, much attention has been devoted to the analysis of probabilistic behaviours, which occur for instance in randomized, fault-tolerant systems. Several different types of systems were proposed: reactive [24, 29], generative [16], stratified [36, 38], alternating [18, 39], (simple) Segala [34, 35], bundle [12] and Pnueli-Zuck [28], among others. For some of these systems, expressions were defined for the specification of their behaviours, as well as axioms to reason about their behavioural equivalence. Examples include [1, 2, 4, 13, 14, 21, 25, 27, 37].

Our previous results [7] apply to the class of so-called Kripke-polynomial functors, which is general enough to include the examples of DFA's and LTS's, as well as many other systems such as Mealy and Moore machines. However, probabilistic systems, weighted automata [15, 33], etc. *cannot* be described by Kripke-polynomial functors. It is the aim of the present paper 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 [7] can be extended.

To this end, we give a non-trivial extension of the class of Kripke-polynomial 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. 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 behaviour 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 ε with probability $1/2$ or behaves as ε' with the same probability. When ε is equivalent to ε', 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 behaviours. 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;

Table 1. All the expressions are closed and guarded. The congruence and the α-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, denote the ordinary 0 and sum of real numbers. With a slight abuse of notation, we write $\bigoplus_{i\in 1\cdots n} p_i \cdot \varepsilon_i$ for $p_1 \cdot \varepsilon_1 \oplus \cdots \oplus p_n \cdot \varepsilon_n$.

Weighted automata – $\mathbb{S} \times (\mathbb{S}^{Id})^A$

$\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid s \mid a(s \cdot \varepsilon)$ where $s \in \mathbb{S}$ and $a \in A$

$(\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \equiv \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3)$ $\quad \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$ $\quad \varepsilon \oplus \emptyset \equiv \varepsilon$

$a(s \cdot \varepsilon) \oplus a(s' \cdot \varepsilon) \equiv a((s + s') \cdot \varepsilon)$ $\quad s \oplus s' \equiv s + s'$ $\quad a(0 \cdot \varepsilon) \equiv \emptyset$

$\varepsilon[\mu x.\varepsilon/x] \equiv \mu x.\varepsilon$ $\quad \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$ $\quad 0 \equiv \emptyset$

Stratified systems – $D_\omega(Id) + (B \times Id) + 1$

$\varepsilon ::= \mu x.\varepsilon \mid x \mid \langle b, \varepsilon\rangle \mid \bigoplus_{i\in 1\cdots n} p_i \cdot \varepsilon_i \mid \downarrow$ where $b \in B$, $p_i \in (0, 1]$ and $\sum_{i\in 1\cdots n} p_i = 1$

$(\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \equiv \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3)$ $\quad \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$ $\quad (p_1 \cdot \varepsilon) \oplus (p_2 \cdot \varepsilon) \equiv (p_1 + p_2) \cdot \varepsilon$

$\varepsilon[\mu x.\varepsilon/x] \equiv \mu x.\varepsilon$ $\quad \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$

Segala systems – $\mathcal{P}_\omega(D_\omega(Id))^A$

$\varepsilon ::= \emptyset \mid \varepsilon \boxplus \varepsilon \mid \mu x.\varepsilon \mid x \mid a(\{\varepsilon'\})$ where $a \in A$, $p_i \in (0, 1]$ and $\sum_{i\in 1\cdots n} p_i = 1$

$\varepsilon' ::= \bigoplus_{i\in 1\cdots n} p_i \cdot \varepsilon_i$

$(\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 \equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3)$ $\quad \varepsilon_1 \boxplus \varepsilon_2 \equiv \varepsilon_2 \boxplus \varepsilon_1$ $\quad \varepsilon \boxplus \emptyset \equiv \varepsilon$ $\quad \varepsilon \boxplus \varepsilon \equiv \varepsilon$

$(\varepsilon'_1 \oplus \varepsilon'_2) \oplus \varepsilon'_3 \equiv \varepsilon'_1 \oplus (\varepsilon'_2 \oplus \varepsilon'_3)$ $\quad \varepsilon'_1 \oplus \varepsilon'_2 \equiv \varepsilon'_2 \oplus \varepsilon'_1$ $\quad (p_1 \cdot \varepsilon) \oplus (p_2 \cdot \varepsilon) \equiv (p_1 + p_2) \cdot \varepsilon$

$\varepsilon[\mu x.\varepsilon/x] \equiv \mu x.\varepsilon$ $\quad \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$

Pnueli-Zuck systems – $\mathcal{P}_\omega D_\omega \mathcal{P}_\omega(Id)^A$

$\varepsilon ::= \emptyset \mid \varepsilon \boxplus \varepsilon \mid \mu x.\varepsilon \mid x \mid \{\varepsilon'\}$ where $a \in A$, $p_i \in (0, 1]$ and $\sum_{i\in 1\cdots n} p_i = 1$

$\varepsilon' ::= \bigoplus_{i\in 1\cdots n} p_i \cdot \varepsilon''_i$

$\varepsilon'' ::= \emptyset \mid \varepsilon'' \boxplus \varepsilon'' \mid a(\{\varepsilon\})$

$(\varepsilon_1 \boxplus \varepsilon_2) \boxplus \varepsilon_3 \equiv \varepsilon_1 \boxplus (\varepsilon_2 \boxplus \varepsilon_3)$ $\quad \varepsilon_1 \boxplus \varepsilon_2 \equiv \varepsilon_2 \boxplus \varepsilon_1$ $\quad \varepsilon \boxplus \emptyset \equiv \varepsilon$ $\quad \varepsilon \boxplus \varepsilon \equiv \varepsilon$

$(\varepsilon'_1 \oplus \varepsilon'_2) \oplus \varepsilon'_3 \equiv \varepsilon'_1 \oplus (\varepsilon'_2 \oplus \varepsilon'_3)$ $\quad \varepsilon'_1 \oplus \varepsilon'_2 \equiv \varepsilon'_2 \oplus \varepsilon'_1$ $\quad (p_1 \cdot \varepsilon'') \oplus (p_2 \cdot \varepsilon'') \equiv (p_1 + p_2) \cdot \varepsilon''$

$(\varepsilon''_1 \boxplus \varepsilon''_2) \boxplus \varepsilon''_3 \equiv \varepsilon''_1 \boxplus (\varepsilon''_2 \boxplus \varepsilon''_3)$ $\quad \varepsilon''_1 \boxplus \varepsilon''_2 \equiv \varepsilon''_2 \boxplus \varepsilon''_1$ $\quad \varepsilon'' \boxplus \emptyset \equiv \varepsilon''$ $\quad \varepsilon'' \boxplus \varepsilon'' \equiv \varepsilon''$

$\varepsilon[\mu x.\varepsilon/x] \equiv \mu x.\varepsilon$ $\quad \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$

a corresponding Kleene Theorem; and a sound and complete axiomatization for the corresponding notion of behavioural 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 [14]. For weighted automata and stratified systems, languages have been defined in [9] and [38] 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 1 summarizes our results.

2 Background

In this section, we present the basic definitions for polynomial functors and coalgebras. We recall, from [7], the language of expressions Exp_G associated with a functor G, the analogue of Kleene's theorem and a sound and complete axiomatization of Exp_G.

Let $\mathbf{Set}$ be the category of sets and functions. Sets are denoted by capital letters $X, Y, \ldots$ and functions by lower case $f, g, \ldots$ The collection of functions from a set X to a set Y is denoted by Y^X. We write $g \circ f$ for function composition, when defined. The product of two sets X, Y is written as $X \times Y$, with projection functions $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$. The set 1 is a singleton set written as $1 = \{*\}$. We define $X + Y$ as the set $X \uplus Y \uplus \{\bot, \top\}$, where $\uplus$ is the disjoint union of sets, with injections $X \xrightarrow{\kappa_1} X \uplus Y \xleftarrow{\kappa_2} Y$. Note that the set $X + Y$ is different from the classical coproduct of X and Y, because of the two extra elements $\bot$ and $\top$. These extra elements are used to represent, respectively, underspecification and inconsistency in the specification of systems.

Polynomial functors. In our definition of polynomial 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 endowed with a binary operation $\vee_B$ and a constant $\bot_B \in B$. The operation $\vee_B$ is commutative, associative and idempotent. The element $\bot_B$ is neutral w.r.t. $\vee_B$. Every set S can be transformed into a join-semilattice by taking B to be the set of all finite subsets of S with union as join.

We are now ready to define the class of polynomial functors. They are functors $G : \mathbf{Set} \to \mathbf{Set}$, built inductively from the identity and constants, using $\times$, $+$ and $(-)^A$. Formally, the class PF of *polynomial functors* on $\mathbf{Set}$ is inductively defined by putting:

$$PF \ni G ::= Id \mid B \mid G_1 + G_2 \mid G_1 \times G_2 \mid G^A$$

with B a finite join-semilattice and A a finite set. For a set S, $Id(S) = S$, $B(S) = B$, $(G_1 \times G_2)(S) = G_1(S) \times G_2(S)$, $(G_1 + G_2)(S) = G_1(S) + G_2(S)$ and $G^A(S) = \{f \mid f : A \to G(S)\}$ and, for a function $f : S \to T$, $Gf : GS \to GT$ is defined as usual [31].

Typical examples of polynomial functors are $D = 2 \times Id^A$, $M = (B \times Id)^A$ and $St = A \times Id$. These functors represent, respectively, the type of *deterministic automata*, *Mealy machines*, and *infinite streams*.

Our definition of polynomial functors slightly differs from the one of [19, 30] in the use of a join-semilattice as constant functor and in the definition of $+$. This small variation plays an important technical role in giving a full coalgebraic treatment of the language of expressions which we shall introduce later. The intuition behind these extensions becomes clear if one recalls that the set of classical regular expressions carries a join-semilattice structure. Since ordinary polynomial functors can be naturally embedded into our polynomial functors above (because every set can be naturally embedded in the generated free join semilattice), one can use the results of Section 5 to obtain regular expressions (and axiomatization) for ordinary polynomial functors.

Next, we give the definition of the ingredient relation, which relates a polynomial functor G with its *ingredients*, *i.e.* the functors used in its inductive construction. We shall use this relation later for typing our expressions. Let $\lhd \subseteq PF \times PF$ be the least reflexive and transitive relation, written infix, such that

$$G_1 \lhd G_1 \times G_2, \quad G_2 \lhd G_1 \times G_2, \quad G_1 \lhd G_1 + G_2, \quad G_2 \lhd G_1 + G_2, \quad G \lhd G^A.$$

If $F \lhd G$, then F is said to be an *ingredient* of G. For example, 2, Id, $2\times Id$, and $2\times Id^A$ are the ingredients of the deterministic automata functor D.

Coalgebras. For an endofunctor G on $\mathbf{Set}$, a G-coalgebra is a pair (S, f) consisting of a set of *states* S together with a function $f : S \to GS$. The functor G, together with the function f, determines the *transition structure* of the G-coalgebra [31]. Examples of coalgebras include deterministic automata, Mealy machines and infinite streams, which are, respectively, coalgebras for the functors D, M and St given above.

A *G-homomorphism* from a G-coalgebra (S, f) to a G-coalgebra (T, g) is a function $h : S \to T$ preserving the transition structure, *i.e.*, such that $g \circ h = Gh \circ f$.

A G-coalgebra (Ω, ω) is said to be *final* if for any G-coalgebra (S, f) there exists a unique G-homomorphism $\mathbf{beh}_S : S \to \Omega$. For every polynomial functor G there exists a final G-coalgebra (Ω_G, ω_G) [31]. The notion of finality plays a key role in defining bisimilarity. For G-coalgebras (S, f) and (T, g) and $s \in S$, $t \in T$, we say that s and t are (G-)bisimilar, written $s \sim t$, if and only if $\mathbf{beh}_S(s) = \mathbf{beh}_T(t)$.

Given a G-coalgebra (S, f) and a subset V of S with inclusion map $i : V \to S$ we say that V is a subcoalgebra of S if there exists $g : V \to GV$ such that i is a homomorphism. Given $s \in S$, $\langle s \rangle \subseteq S$ denotes the subcoalgebra generated by s [31], *i.e.* the set consisting of states that are reachable from s. We will write $Coalg_{lf}(G)$ for the category of G-coalgebras that are *locally finite*: objects are G-coalgebras (S, f) such that for each state $s \in S$ the generated subcoalgebra $\langle s \rangle$ is finite; maps are the usual homomorphisms of coalgebras.

2.1 A Language of Expressions for Polynomial Coalgebras

In order to be able to formulate the generalization of our previous work [7], we first have to recall the main definitions and results concerning the language of expressions associated to a polynomial functor G. Note that in [7] we actually treated Kripke polynomial functors, as mentioned also in the present introduction. In order to give a more uniform and concise presentation, we omit in this section the case of the finite powerset $\mathcal{P}_\omega$ (thus, we only present polynomial functors), which can be recovered as a special instance of the monoidal valuation functor (Section 3). 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 polynomial functors. Then, we present the analogue of Kleene's theorem.

Let A be a finite set, B a finite join-semilattice and X a set of fixpoint variables. The set of all *expressions* is given by the following grammar (where $a \in A$, $b \in B$):

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

An expression is *closed* if it has no free occurrences of fixpoint variables x. We denote the set of closed expressions by Exp.

Intuitively, expressions denote elements of final coalgebras. The expressions $\emptyset$, $\varepsilon \oplus \varepsilon$ and $\mu x.\, \varepsilon$ will play a role similar to, respectively, the empty language, the union of languages and the Kleene star in classical regular expressions for deterministic automata. The expressions $l(\varepsilon)$, $r(\varepsilon)$, $l[\varepsilon]$, $r[\varepsilon]$ and $a(\varepsilon)$ denote the left and right hand-side of products and sums and function application, respectively.

Next, we present a typing assignment system that will allow us to associate with each functor G the expressions ε that are valid specifications of G-coalgebras. The typing proceeds following the structure of the expressions and the ingredients of the functors. We type expressions ε using the ingredient relation, for $a \in A$, $b \in B$ and $x \in X$, as follows:

$$\frac{}{\vdash \emptyset : F \lhd G} \qquad \frac{}{\vdash b : B \lhd G} \qquad \frac{}{\vdash x : Id \lhd G} \qquad \frac{\vdash \varepsilon : G \lhd G}{\vdash \mu x.\varepsilon : G \lhd G}$$

$$\frac{\vdash \varepsilon_1 : F \lhd G \quad \vdash \varepsilon_2 : F \lhd G}{\vdash \varepsilon_1 \oplus \varepsilon_2 : F \lhd G} \qquad \frac{\vdash \varepsilon : G \lhd G}{\vdash \varepsilon : Id \lhd G} \qquad \frac{\vdash \varepsilon : F \lhd G}{\vdash a(\varepsilon) : F^A \lhd G}$$

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l(\varepsilon) : F_1 \times F_2 \lhd G} \quad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r(\varepsilon) : F_1 \times F_2 \lhd G} \quad \frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l[\varepsilon] : F_1 + F_2 \lhd G} \quad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r[\varepsilon] : F_1 + F_2 \lhd G}$$

Most of the rules are self-explanatory. The rule involving $Id \lhd G$ reflects the isomorphism of the final coalgebra: $\Omega_G \cong G(\Omega_G)$. It is interesting to note that the rule for the variable x guarantees that occurrences of variables in a fixpoint expression are *guarded*: they occur under the scope of expressions $l(\varepsilon)$, $r(\varepsilon)$, $l[\varepsilon]$, $r[\varepsilon]$ and $a(\varepsilon)$. For further details we refer to [7].

The set of G-expressions of well-typed expressions associated with a polynomial functor G is defined by $Exp_G = Exp_{G \lhd G}$, where, for F an ingredient of G:

$$Exp_{F \lhd G} = \{\varepsilon \in Exp \mid \vdash \varepsilon : F \lhd G\}\,.$$

To illustrate this definition we instantiate it for the functor $D = 2 \times Id^A$.

Example 1 (Deterministic expressions). Let A be a finite set and let X be a set of fixpoint variables. The set Exp_D of well-typed D-expressions is given by the BNF:

$$\varepsilon ::= \emptyset \mid x \mid l(0) \mid l(1) \mid r(a(\varepsilon)) \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon$$

where $a \in A$, $x \in X$, ε is closed and occurrences of fixpoint variables are within the scope of an input action, as can be easily checked by structural induction on the length of the type derivations.

Our derived syntax for this functor differs from classical regular expressions in the use of action prefixing and fixpoint instead of sequential composition and star, respectively. However, as we will soon see (Theorem 1), the expressions in our syntax correspond to deterministic automata and, in that sense, they are equivalent to classical regular expressions.

The language of expressions induces an algebraic description of systems. In [7], we showed that such language is a coalgebra. More precisely, we defined a function $\lambda_{F \lhd G} : Exp_{F \lhd G} \to F(Exp_G)$ and then set $\lambda_G = \lambda_{G \lhd G}$, providing Exp_G with a coalgebraic structure. The function $\lambda_{F \lhd G}$ is defined by double induction on the maximum number of nested unguarded occurrences of μ-expressions in ε and on the length of the proofs for typing expressions. For every ingredient F of a polynomial functor G and $\varepsilon \in Exp_{F \lhd G}$,

Table 2. The function $Plus_{F\triangleleft G} : F(Exp_G) \times F(Exp_G) \to F(Exp_G)$ and the constant $Empty_{F\triangleleft G} \in F(Exp_G)$

$$
\begin{array}{ll}
Empty_{Id\triangleleft G} & = \emptyset \\
Empty_{B\triangleleft G} & = \bot_B \\
Empty_{F_1+F_2\triangleleft G} & = \bot \\
Empty_{F_1\times F_2\triangleleft G} & = \langle Empty_{F_1\triangleleft G}, Empty_{F_2\triangleleft G}\rangle \\
Empty_{F^A\triangleleft G} & = \lambda a. Empty_{F\triangleleft G} \\
 & \\
Plus_{Id\triangleleft G}(\varepsilon_1,\varepsilon_2) & = \varepsilon_1 \oplus \varepsilon_2 \\
Plus_{B\triangleleft G}(b_1,b_2) & = b_1 \vee_B b_2 \\
Plus_{F_1+F_2\triangleleft G}(x,\top) & = Plus_{F_1+F_2\triangleleft G}(\top,x) = \top \\
Plus_{F_1+F_2\triangleleft G}(x,\bot) & = Plus_{F_1+F_2\triangleleft G}(\bot,x) = x \\
Plus_{F_1+F_2\triangleleft G}(\kappa_i(\varepsilon_1),\kappa_i(\varepsilon_2)) & = \kappa_i(Plus_{F_i\triangleleft G}(\varepsilon_1,\varepsilon_2)),\quad i\in\{1,2\} \\
Plus_{F_1+F_2\triangleleft G}(\kappa_i(\varepsilon_1),\kappa_j(\varepsilon_2)) & = \top \text{ for } i,j\in\{1,2\} \text{ and } i\neq j \\
Plus_{F_1\times F_2\triangleleft G}(\langle\varepsilon_1,\varepsilon_2\rangle,\langle\varepsilon_3,\varepsilon_4\rangle) & = \langle Plus_{F_1\triangleleft G}(\varepsilon_1,\varepsilon_3), Plus_{F_2\triangleleft G}(\varepsilon_2,\varepsilon_4)\rangle \\
Plus_{F^A\triangleleft G}(f,g) & = \lambda a.\ Plus_{F\triangleleft G}(f(a),g(a))
\end{array}
$$

the mapping $\lambda_{F\triangleleft G}(\varepsilon)$ is given by :

$$
\begin{array}{ll}
\lambda_{F\triangleleft G}(\emptyset) = Empty_{F\triangleleft G} & \lambda_{F_1\times F_2\triangleleft G}(l(\varepsilon)) = \langle \lambda_{F_1\triangleleft G}(\varepsilon), Empty_{F_2\triangleleft G}\rangle \\
\lambda_{F\triangleleft G}(\varepsilon_1\oplus\varepsilon_2) & \lambda_{F_1\times F_2\triangleleft G}(r(\varepsilon)) = \langle Empty_{F_1\triangleleft G}, \lambda_{F_2\triangleleft G}(\varepsilon)\rangle \\
\quad = Plus_{F\triangleleft G}(\lambda_{F\triangleleft G}(\varepsilon_1),\lambda_{F\triangleleft G}(\varepsilon_2)) & \lambda_{F_1+F_2\triangleleft G}(l[\varepsilon]) = \kappa_1(\lambda_{F_1\triangleleft G}(\varepsilon)) \\
\lambda_{G\triangleleft G}(\mu x.\varepsilon) = \lambda_{G\triangleleft G}(\varepsilon[\mu x.\varepsilon/x]) & \lambda_{F_1+F_2\triangleleft G}(r[\varepsilon]) = \kappa_2(\lambda_{F_2\triangleleft G}(\varepsilon)) \\
\lambda_{Id\triangleleft G}(\varepsilon) = \varepsilon \text{ for } G\neq Id & \lambda_{F^A\triangleleft G}(a(\varepsilon)) = \lambda a'.\begin{cases}\lambda_{F\triangleleft G}(\varepsilon) & a=a'\\ Empty_{F\triangleleft G} & otherwise\end{cases} \\
\lambda_{B\triangleleft G}(b) = b &
\end{array}
$$

Here, $\varepsilon[\mu x.\varepsilon/x]$ denotes syntactic substitution, replacing every free occurrence of x in ε by $\mu x.\varepsilon$. The auxiliary constructs *Empty* and *Plus* are defined in Table 2. Note that we use λ in the right hand side of the equation for $\lambda_{F^A\triangleleft G}(a(\varepsilon))$ to denote lambda abstraction. This overlap of symbols is safe since when we use it in $\lambda_{F\triangleleft G}$ it is always accompanied by the type subscript. It is interesting to remark that λ_G is the generalization of the well-known notion of Brzozowski derivative [8] for regular expressions and, moreover, it provides an operational semantics for expressions.

We now present the generalization of Kleene's theorem.

Theorem 1 ([7, Theorem 4]). *Let G be a polynomial functor.*

1. *For every locally finite G-coalgebra (S,g) and for any $s\in S$ there exists an expression $\varepsilon_s\in Exp_G$ such that $\varepsilon_s\sim s$.*
2. *For every $\varepsilon\in Exp_G$, we can construct a coalgebra (S,g) such that S is finite and there exists $s\in S$ with $\varepsilon\sim s$.*

Note that $\varepsilon_s\sim s$ means that the expression ε_s and the (system with initial) state s have the same behaviour. For instance, for DFA's, this would mean that they denote and accept the same regular language. Similarly for ε and s in item 2..

In [7], we presented a sound and complete axiomatization wrt bisimilarity for Exp_G. We will not recall it here because this axiomatization can be recovered as an instance of the one presented in Section 4.

3 Monoidal Valuation Functor

In the previous section we introduced polynomial functors and a language of expressions for specifying coalgebras. Coalgebras for polynomial functors cover many interesting types of systems, such as deterministic and Mealy automata, but not quantitative systems. For this reason, we recall the definition of the *monoidal valuation functor* [17], 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.

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. Notice that an idempotent commutative monoid is a join-semilattice.

Given a function φ from a set S to a monoid $\mathbb{M}$, we define *support of* φ as the set $\{s \in S \mid \varphi(s) \neq 0\}$.

Definition 1 (Monoidal valuation Functor). *Let $\mathbb{M}$ be a commutative monoid. The monoidal valuation functor $\mathbb{M}_\omega^-$:$\mathbf{Set} \to \mathbf{Set}$ 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 $h : S \to T$, $\mathbb{M}_\omega^h$:$\mathbb{M}_\omega^S \to \mathbb{M}_\omega^T$ 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*

$$\varphi^h(t) = \sum_{s' \in h^{-1}(t)} \varphi(s')$$

Proposition 1. *The functor $\mathbb{M}_\omega^-$ has a final coalgebra.*

Note that the (finite) powerset functor $\mathcal{P}_\omega(-)$ coincides with $\mathbf{2}_\omega^-$. This is often used to represent non-deterministic systems. For example, (image-finite) LTS's (with labels over A) are coalgebras for the functor $\mathcal{P}_\omega(-)^A$. In the following, to simplify the notation we will always write $\mathbb{M}^-$ instead of $\mathbb{M}_\omega^-$.

By combining the monoidal valuation functor with the polynomial functors, we can model quantitative systems as coalgebras. As an example, we mention weighted automata.

Weighted Automata. 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.

Weighted automata [15, 33] are transition systems labelled over a set A and with weights in a semiring $\mathbb{S}$. Moreover, each state is equipped with an output value[1] in $\mathbb{S}$. From a coalgebraic perspective weighted automata are coalgebras for the functor $\mathbb{S} \times (\mathbb{S}^{Id})^A$, where we use $\mathbb{S}$ to denote, the commutative monoid of the semiring $\mathbb{S}$. More concretely, a coalgebra for $\mathbb{S} \times (\mathbb{S}^{Id})^A$ is a pair $(Q, \langle o, t\rangle)$, where Q is a set of states,

[1] In the original formulation also an input value is considered. To simplify the presentation and following [10] we omit it.

$o : Q \to \mathbb{S}$ is the function that associates an output weight to each state $q \in Q$ and $t : Q \to (\mathbb{S}^Q)^A$ is the transition relation that associates a weight to each transition: $q \xrightarrow{a,s} q' \iff t(q)(a)(q') = s$.

Bisimilarity for weighted automata has been studied in [9] and it coincides with the coalgebraic notion of bisimilarity. (For a proof, see [6].)

Proposition 2. *Bisimilarity for $\mathbb{S} \times (\mathbb{S}^{Id})^A$ coincides with the weighted automata bisimilarity defined in [9].*

4 A Non-idempotent Algebra for Quantitative Regular Behaviours

In this section, we will extend the framework presented in Section 2 in order to deal with quantitative systems, as described in the previous section. We will start by defining an appropriate class of functors H, proceed with presenting the language Exp_H of expressions associated with H together with a Kleene like theorem and finally we introduce a sound and complete axiomatization of Exp_H.

Formally, the set QF of *quantitative functors* on $\mathbf{Set}$ is defined inductively by putting:

$$QF \ni H ::= G \mid \mathbb{M}^H \mid (\mathbb{M}^H)^A \mid \mathbb{M}_1^{H_1} \times \mathbb{M}_2^{H_2} \mid \mathbb{M}_1^{H_1} + \mathbb{M}_2^{H_2}$$

where G is a polynomial functor, $\mathbb{M}$ is a commutative monoid and A is a finite set. Note that we do not allow *mixed* functors, such as $G + \mathbb{M}^H$ or $G \times \mathbb{M}^H$. The reason for this restriction will become clear later in this section when we discuss the proof of Kleene's theorem. In Section 5, we will show how to deal with such mixed functors.

Every definition we presented in Section 2 needs now to be extended to quantitative functors. We start by observing that taking the current definitions and replacing the subscript $F \lhd G$ with $F \lhd H$ does most of the work. In fact, having that, we just need to extend all the definitions for the case $\mathbb{M}^F \lhd H$.

We start by introducing a new expression $m \cdot \varepsilon$, with $m \in \mathbb{M}$, extending the set of untyped expressions, which is now given by:

$$\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \mid m \cdot \varepsilon$$

The intuition behind the new expression is that there is a transition between the current state and the state specified by ε with weight m.

The ingredient relation is extended with the rule $H \lhd \mathbb{M}^H$, the type system and $\lambda_{\mathbb{M}^F \lhd H}$ with the following rules:

$$\frac{\varepsilon : F \lhd H}{m \cdot \varepsilon : \mathbb{M}^F \lhd H}$$

$$\begin{aligned} &Empty_{\mathbb{M}^F \lhd H} = \lambda\varepsilon'.0 \\ &Plus_{\mathbb{M}^F \lhd H}(f,g) = \lambda\varepsilon'.f(\varepsilon') + g(\varepsilon') \\ &\lambda_{\mathbb{M}^F \lhd H}(m \cdot \varepsilon) = \lambda\varepsilon'.\begin{cases} m \text{ if } \lambda_{F \lhd H}(\varepsilon) = \varepsilon' \\ 0 \quad otherwise \end{cases} \end{aligned}$$

where 0 and $+$ are the neutral element and the binary operation of $\mathbb{M}$. Recall that the function $\lambda_H = \lambda_{H \lhd H}$ provides an operational semantics for the expressions. We will soon illustrate this for the case of expressions for weighted automata (Example 2).

Finally, we introduce an equational system for expressions of type $F \lhd H$. We will use the symbol $\equiv \subseteq Exp_{F \lhd H} \times Exp_{F \lhd H}$, omitting the subscript $F \lhd H$, for the least relation satisfying the following:

$(Idempotency)\quad \varepsilon \oplus \varepsilon \equiv \varepsilon, \quad \varepsilon \in Exp_{F \lhd G}$
$(Commutativity)\ \varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_2 \oplus \varepsilon_1$
$(Associativity)\quad \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) \equiv (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3$
$(Empty)\qquad \emptyset \oplus \varepsilon \equiv \varepsilon$

$(FP)\qquad \gamma[\mu x.\gamma/x] \equiv \mu x.\gamma$
$(Unique)\ \gamma[\varepsilon/x] \equiv \varepsilon \Rightarrow \mu x.\gamma \equiv \varepsilon$

$(B-\emptyset)\qquad \emptyset \equiv \bot_B$
$(\times-\emptyset-L)\ \ l(\emptyset) \equiv \emptyset$
$(\times-\emptyset-R)\ \ r(\emptyset) \equiv \emptyset$
$(-^A-\emptyset)\qquad a(\emptyset) \equiv \emptyset$
$(\mathbb{M}^- - \emptyset)\qquad (0 \cdot \varepsilon) \equiv \emptyset$
$(+-\oplus-L)\ \ l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2]$
$(\alpha-equiv)\quad \mu x.\gamma \equiv \mu y.\gamma[y/x]$ if y not free in γ

$(B-\oplus)\qquad b_1 \oplus b_2 \equiv b_1 \vee_B b_2$
$(\times-\oplus-L)\ \ l(\varepsilon_1 \oplus \varepsilon_2) \equiv l(\varepsilon_1) \oplus l(\varepsilon_2)$
$(\times-\oplus-R)\ \ r(\varepsilon_1 \oplus \varepsilon_2) \equiv r(\varepsilon_1) \oplus r(\varepsilon_2)$
$(-^A-\oplus)\qquad a(\varepsilon_1 \oplus \varepsilon_2) \equiv a(\varepsilon_1) \oplus a(\varepsilon_2)$
$(\mathbb{M}^- - \oplus)\qquad (m \cdot \varepsilon) \oplus (m' \cdot \varepsilon) \equiv (m + m') \cdot \varepsilon$
$(+-\oplus-R)\ \ r[\varepsilon_1 \oplus \varepsilon_2] \equiv r[\varepsilon_1] \oplus r[\varepsilon_2]$
$(+-\oplus-\top)\ \ l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset]$

$(Cong)$ If $\varepsilon \equiv \varepsilon'$ then $\varepsilon \oplus \varepsilon_1 \equiv \varepsilon' \oplus \varepsilon_1$, $\mu x.\varepsilon \equiv \mu x.\varepsilon'$, $l(\varepsilon) \equiv l(\varepsilon')$, $r(\varepsilon) \equiv r(\varepsilon')$, $l[\varepsilon] \equiv l[\varepsilon']$, $r[\varepsilon] \equiv r[\varepsilon']$, $a(\varepsilon) \equiv a(\varepsilon')$, and $m \cdot \varepsilon \equiv m \cdot \varepsilon'$.

We shall write $Exp/_{\equiv}$ for the set of expressions modulo $\equiv$.

Note that $(Idempotency)$ only holds for $\varepsilon \in Exp_{F \lhd G}$. 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, $(2 \cdot \emptyset) \oplus (2 \cdot \emptyset) \equiv 4 \cdot \emptyset$. In the case of an idempotent commutative monoid $\mathbb{M}$, $(Idempotency)$ follows from the axiom $(\mathbb{M}^- - \oplus)$.

Lemma 1. *Let $\mathbb{M}$ be an idempotent commutative monoid. For every expression $\varepsilon \in Exp_{\mathbb{M}^F \lhd H}$, one has $\varepsilon \oplus \varepsilon \equiv \varepsilon$.*

Example 2 (Expressions for weighted automata). The syntax automatically derived from our typing system for the functor $W = \mathbb{S} \times (\mathbb{S}^{Id})^A$ is the following.

$$\begin{array}{l} \varepsilon ::= \ \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid l(s) \mid r(\varepsilon') \\ \varepsilon' ::= \ \emptyset \mid \varepsilon' \oplus \varepsilon' \mid a(\varepsilon'') \\ \varepsilon'' ::= \ \emptyset \mid \varepsilon'' \oplus \varepsilon'' \mid s \cdot \varepsilon \end{array}$$

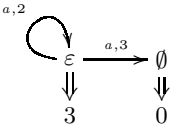
where $s \in \mathbb{S}$, $a \in A$ and all the occurrences of x are guarded. The semantics of these expressions is given by the function $\lambda_{W \lhd W}$ (hereafter denoted by λ_W) which is an instance of the general $\lambda_{F \lhd H}$ defined above. It is given by:

$$\begin{array}{ll} \lambda_W(\emptyset) & = \langle 0, \lambda a.\lambda\varepsilon.0 \rangle \\ \lambda_W(\varepsilon_1 \oplus \varepsilon_2) & = \langle s_1 + s_2, \lambda a.\lambda\varepsilon.(f(a)(\varepsilon) + g(a)(\varepsilon)) \\ \quad \text{where } \langle s_1, f \rangle = \lambda_W(\varepsilon_1) \text{ and } \langle s_2, g \rangle = \lambda_W(\varepsilon_2) & \\ \lambda_W(\mu x.\varepsilon) & = \lambda_W(\varepsilon[\mu x.\varepsilon/x]) \\ \lambda_W(l(s)) & = \langle s, \lambda a.\lambda\varepsilon.0 \rangle \\ \lambda_W(r(\varepsilon')) & = \langle 0, \lambda_{(\mathbb{S}^{Id})^A \lhd W}(\varepsilon') \rangle \end{array}$$

$$\begin{array}{ll} \lambda_{(\mathbb{S}^{Id})^A \lhd W}(\emptyset) & = \lambda a.\lambda\varepsilon.0 \\ \lambda_{(\mathbb{S}^{Id})^A \lhd W}(\varepsilon_1 \oplus \varepsilon_2) & = \lambda a.\lambda\varepsilon.(f_1(a)(\varepsilon) + f_2(s)(\varepsilon)) \\ \quad \text{where } f_i = \lambda_{(\mathbb{S}^{Id})^A \lhd W}(\varepsilon_i), i \in \{1,2\} & \\ \lambda_{(\mathbb{S}^{Id})^A \lhd W}(a(\varepsilon'')) & = \lambda a'. \begin{cases} \lambda_{S^{Id} \lhd W}(\varepsilon'') & a = a' \\ \lambda\varepsilon.0 & oth. \end{cases} \end{array}$$

$$\begin{array}{ll} \lambda_{(\mathbb{S}^{Id}) \lhd W}(\emptyset) & = \lambda\varepsilon.0 \\ \lambda_{(\mathbb{S}^{Id}) \lhd W}(\varepsilon_1 \oplus \varepsilon_2) & = \lambda\varepsilon.(f_1(\varepsilon) + f_2(\varepsilon)) \\ \quad \text{where } f_i = \lambda_{(\mathbb{S}^{Id}) \lhd W}(\varepsilon_i), i \in \{1,2\} & \\ \lambda_{(\mathbb{S}^{Id}) \lhd W}(s \cdot \varepsilon) & = \lambda\varepsilon'. \begin{cases} s & \varepsilon = \varepsilon' \\ 0 & oth. \end{cases} \end{array}$$

The function λ_W assigns to each expression ε a pair $\langle s, t\rangle$, consisting of an output weight $s \in \mathbb{S}$ and a function $t : A \to \mathbb{S}^{Exp_W}$. For a concrete example, let $\mathbb{S} = \mathbb{R}$ and consider $\varepsilon = \mu x.r(a(2 \cdot x \oplus 3 \cdot \emptyset)) \oplus l(1) \oplus l(2)$. The semantics of this expression, obtained by λ_W is described by the weighted automaton below.



In Table 1 a more concise syntax for expressions for weighted automata is presented. To derive that syntax from the one automatically generated, we first write ε' as

$$\varepsilon' ::= \emptyset \mid \varepsilon' \oplus \varepsilon' \mid a(s \cdot \varepsilon)$$

using the axioms $a(\emptyset) \equiv \emptyset$ and $a(\varepsilon_1'' \oplus \varepsilon_2'') \equiv a(\varepsilon_1'') \oplus a(\varepsilon_2'')$. Similarly, using $r(\emptyset) = \emptyset$ and $r(\varepsilon_1' \oplus \varepsilon_2') \equiv r(\varepsilon_1') \oplus r(\varepsilon_2')$, we can write ε as follows.

$$\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid l(s) \mid r(a(s \cdot \varepsilon))$$

In Table 1, we abbreviate $l(s)$ to s and $r(a(s{\cdot}\varepsilon))$ to $a(s{\cdot}\varepsilon)$, without any risk of confusion. Note that the axioms presented in Table 1 also reflect the changes in the syntax of the expressions.

We are now ready to formulate the analogue of Kleene's theorem for quantitative systems.

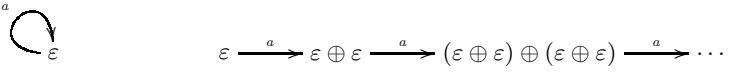
Theorem 2 (Kleene's theorem for quantitative functors). *Let H be a quantitative functor.*

1. *For every locally finite H-coalgebra (S, h) and for every $s \in S$ there exists an expression $\varepsilon_s \in Exp_H$ such that $s \sim \varepsilon_s$.*
2. *For every $\varepsilon \in Exp_H$, there exists a finite H-coalgebra (S, h) with $s \in S$ such that $s \sim \varepsilon$.*

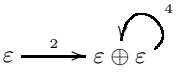
The proof of the theorem can be found in [6], but let us explain what are the technical difficulties that arise when compared with Theorem 1, where only polynomial functors are considered.

In the proof of item 2. in Theorem 1, we start by constructing the subcoalgebra generated by ε, using the fact that the set Exp_G has a coalgebra structure given by λ_G. Then, we observe that such subcoalgebra might not be finite and, following a similar result for classical regular expressions, we show that finiteness can be obtained by taking the subcoalgebra generated modulo $(Associativity)$, $(Commutativity)$ and $(Idempotency)$ (ACI).

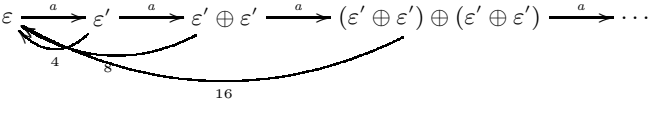
Consider for instance the expression $\varepsilon = \mu x.r(a(x \oplus x))$ of type $D = 2 \times Id^A$. The subcoalgebras generated with and without applying ACI are the following:



We cannot apply ACI in the quantitative setting, since the idempotency axiom does not hold anymore. However, surprisingly enough, in the case of the functor $\mathbb{M}^H$, we are able to prove finiteness of the subcoalgebra $\langle\varepsilon\rangle$ just by using $(Commutativity)$ and $(Associativity)$. The key observation is that the monoid structure will be able to avoid the infinite scenario described above. In fact, for the functor M^H one can prove that, if $\varepsilon \oplus \varepsilon$ is one of the successors of ε then the successors of $\varepsilon \oplus \varepsilon$ will all be contained in the set of direct successors of ε, which we know is finite . What happens is concisely captured by the following example. Take the expression $\varepsilon = \mu x.2\cdot(x \oplus x)$ for the functor $\mathbb{R}^{Id}$. Then, the subcoalgebra generated by ε is depicted in the following picture:



In this manner, we are able to deal with the base cases G (polynomial functor) and $\mathbb{M}^H$ of the inductive definition of the set of quantitative functors. Moreover, the functors $\mathbb{M}^H \times \mathbb{M}^H$ and $\mathbb{M}^H + \mathbb{M}^H$ inherit the above property from $\mathbb{M}^H$ and do not pose any problem in the proof of Kleene's theorem. The syntactic restriction that excludes mixed functors is needed because of the following problem. Take as an example the functor $\mathbb{M}^{Id} \times Id^A$. 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 ε 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 $H \in QF$, the derivation of a language Exp_H. Moreover, Theorem 2 guarantees that for each expression $\varepsilon \in Exp_H$, there exists a finite H-coalgebra (S, h) that contains a state $s \in S$ bisimilar to ε and, conversely, for each locally finite H-coalgebra (S, h) and for every state in s there is an expression $\varepsilon_s \in Exp_H$ bisimilar to s. The proof of Theorem 2, which can be found in [6], shows how to compute the H-coalgebra (S, h) corresponding to an expression ε and vice-versa.

The axiomatization presented above is sound and complete:

Theorem 3 (Soundness and Completeness). *Let H be a quantitative functor and let $\varepsilon_1, \varepsilon_2 \in Exp_H$. Then, $\varepsilon_1 \sim \varepsilon_2 \iff \varepsilon_1 \equiv \varepsilon_2$.*

The proof of this theorem follows a similar strategy as in [7, 20] and can be found in [6].

5 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 4.

Given F and G two endofunctors on $\mathbf{Set}$, a *natural transformation* $\alpha{:}F \Rightarrow G$ is a family of functions $\alpha_S{:}F(S) \to G(S)$ (for all sets S), such that for all functions $h{:}T \to U$, $\alpha_U \circ F(h) = G(h) \circ \alpha_T$. If all the α_S are injective, then we say that α is injective.

Proposition 3. *An injective natural transformation $\alpha{:}F \Rightarrow G$ induces a functor $\alpha \circ (-) : Coalg_{lf}(F) \to Coalg_{lf}(G)$ that preserves and reflects bisimilarity.*

This result (proof can be found in [6]) allows us to extend both regular expressions and axiomatization to many functors. Indeed, consider a functor F that is not quantitative, but that has an injective natural transformation α into some quantitative functor H. A (locally finite) F-coalgebra can be translated into a (locally finite) H-coalgebra via the functor $\alpha \circ (-)$ and then it can be characterized by using expressions in Exp_H (as we will show soon, for the converse some care is needed). The axiomatization for Exp_H is still sound and complete for F-coalgebras, since the functor $\alpha \circ (-)$ preserves and reflects bisimilarity.

However, notice that Kleene's theorem does not hold anymore, because not all the expressions in Exp_H denote F-regular behaviours or, more precisely, not all expressions of Exp_H are equivalent to H-coalgebras that are in the image of $\alpha \circ (-)$. Thus, in order to retrieve Kleene's theorem, one has just to exclude such expressions. In many situations, this is feasible by simply imposing some syntactic constraints on Exp_H.

As an example, we recall the definition of the probability functor that, in the next section, will allow us to derive regular expressions for probabilistic systems.

Definition 2 (Probability functor). *A probability distribution over a set S is a function $d : S \to [0,1]$ such that $\sum_{s \in S} d(s) = 1$. The probability functor $\mathcal{D}_\omega{:}\mathbf{Set} \to \mathbf{Set}$ 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 $h : S \to T$, $\mathcal{D}_\omega(h)$ maps each $d \in \mathcal{D}_\omega(S)$ into d^h as defined in Definition 1.*

Now recall the functor $\mathbb{R}^{Id}$ from Section 3. Note that for any set S, $D_\omega(S) \subseteq \mathbb{R}^S$ since probability distributions are also functions from S to $\mathbb{R}$. Let ι be the family of inclusions $\iota_S{:}D_\omega(S) \to \mathbb{R}^S$. It is easy to see that ι is a natural transformation between D_ω and $\mathbb{R}^{Id}$ (the two functors are defined in the same way on arrows). Thus, in order to specify D_ω-coalgebras, we can use $\varepsilon \in Exp_{\mathbb{R}^{Id}}$ which are the closed and guarded expressions given by $\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid r \cdot \varepsilon$, for $r \in \mathbb{R}$. However, this language allows us to specify $\mathbb{R}^{Id}$-behaviours that are not D_ω-behaviours, such as for example, $\mu x.2 \cdot x$ and $\mu x.0 \cdot x$. In order to obtain a language that specifies all and only the regular D_ω-behaviours, it is enough to restrict the syntax of $Exp_{\mathbb{R}^{Id}}$, as follows:

$$\varepsilon ::= x \mid \mu x.\varepsilon \mid \bigoplus_{i \in 1 \ldots n} p_i \cdot \varepsilon_i \quad \text{for } p_i \in (0,1] \text{ such that } \sum_{i \in 1 \ldots n} p_i = 1$$

where, with a slight abuse of notation, $\bigoplus_{i \in 1 \ldots n} p_i \cdot \varepsilon_i$ denotes $p_1 \cdot \varepsilon_1 \oplus \cdots \oplus p_n \cdot \varepsilon_n$.

In the next section, we will use this kind of syntactic restrictions for defining regular expressions of probabilistic systems.

For another example, consider the functors Id and $\mathcal{P}_\omega(Id)$. Let τ be the family of functions $\tau_S : S \to \mathcal{P}_\omega(S)$ mapping each $s \in S$ in the singleton set $\{s\}$. It is easy to see

that τ is an injective natural transformation. With the above observation, we can also get regular expressions for the functor $\mathbb{M}^{Id} \times Id^A$ that, as discussed in Section 4, does not belong to our class of quantitative functors. Indeed, by extending τ, we can construct an injective natural transformation $\mathbb{M}^{Id} \times Id^A \Rightarrow \mathbb{M}^{Id} \times \mathcal{P}_\omega(Id)^A$.

In the same way, we can construct an injective natural transformation from the functor $D_\omega(Id)+(A\times Id)+1$ (that is the type of stratified systems) into $\mathbb{R}^{Id}+(A\times\mathcal{P}_\omega(Id))+1$. 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 behaviours, we again have to restrict the syntax.

The procedure of appropriately restricting the syntax usually requires some ingenuity. We shall see that in many concrete cases, as for instance D_ω above, it is fairly intuitive which restriction to choose.

6 Probabilistic Systems

Many different types of probabilistic systems have been defined in literature: 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 for the corresponding functor. A systematic study of all these systems as coalgebras was made in [5]. In particular, Fig.1 of [5] provides a full correspondence between types of systems and functors. By employing this correspondence, 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. Table 1 shows the expressions and the axiomatizations that we have obtained, after some simplification of the canonically derived syntax (which is often verbose and redundant).

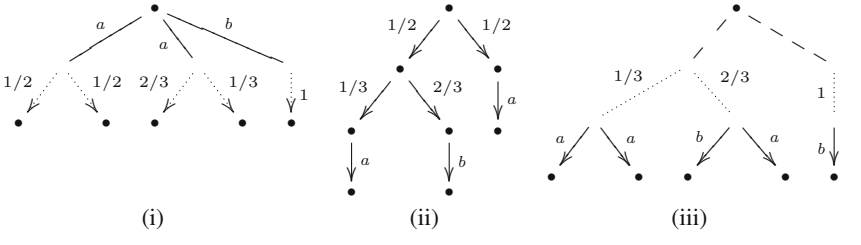


Fig. 1. (i) A simple Segala system, (ii) a stratified system and (iii) a Pnueli-Zuck system

Simple Segala systems. Simple Segala systems are coalgebras of type $\mathcal{P}_\omega(D_\omega(Id))^A$ (recall that $\mathcal{P}_\omega$ is the functor $\mathbf{2}^-$). These are like labelled transition systems, but each labelled transition leads to a probability distribution of states instead of a single state. An example is shown in Fig.1(i).

Table 1 shows expressions and axioms for simple Segala systems. In the following we show how to derive these. As described in Section 5, we can derive the expressions for $\mathcal{P}_\omega(\mathbb{R}^{Id})^A$ instead of $\mathcal{P}_\omega(D_\omega(Id))^A$, and then impose some syntactic constraints on $Exp_{\mathcal{P}_\omega(\mathbb{R}^{Id})^A}$ in order to characterize all and only the $\mathcal{P}_\omega(D_\omega(Id))^A$ behaviours. By simply applying our typing systems to $\mathcal{P}_\omega(\mathbb{R}^{Id})^A$, we derive the expressions:

$$\begin{array}{l}\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid a(\varepsilon') \\ \varepsilon' ::= \emptyset \mid \varepsilon' \oplus \varepsilon' \mid 1 \cdot \varepsilon'' \mid 0 \cdot \varepsilon'' \\ \varepsilon'' ::= \emptyset \mid \varepsilon'' \oplus \varepsilon'' \mid p \cdot \varepsilon\end{array}$$

where $a \in A$, $p \in \mathbb{R}$ and 0 and 1 are the elements of the boolean monoid $\mathbf{2}$.

Now, observe that the syntax for ε', due to the axiom $0 \cdot \varepsilon'' \equiv \emptyset$ can be reduced to

$$\varepsilon' ::= \emptyset \mid \varepsilon' \oplus \varepsilon' \mid 1 \cdot \varepsilon''$$

which, because of $a(\varepsilon'_1) \oplus a(\varepsilon'_2) \equiv a(\varepsilon'_1 \oplus \varepsilon'_2)$ and $a(\emptyset) \equiv \emptyset$ is equivalent to the simplified syntax:

$$\varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid a(\{\varepsilon''\})$$

Here, and in what follows, $\{\varepsilon''\}$ abbreviates $1 \cdot \varepsilon''$. Note that the axiomatization would have to include the axiom $a(\{\varepsilon''\}) \oplus a(\{\varepsilon''\}) \equiv a(\{\varepsilon''\})$, as a consequence of $(\mathbb{M}^- - \oplus)$ and $(-^A - \oplus)$. However, this axiom is subsumed by the $(Idempotency)$ axiom, which we add to the axiomatization, since it holds for expressions $\varepsilon \in Exp_{\mathcal{P}_\omega(\mathbb{R}^{Id})^A}$. This, combined with the restrictions to obtain $\mathbb{D}_\omega$ out of $\mathbb{R}^{Id}$, leads to the expressions and the axiomatization in Table 1 where, in order to avoid confusion, we use $\boxplus$ instead of $\oplus$, making a clear distinction between the idempotent and non-idempotent sums.

As an example, the expression $a(\{1/2 \cdot \emptyset \oplus 1/2 \cdot \emptyset\}) \boxplus a(\{1/3 \cdot \emptyset \oplus 2/3 \cdot \emptyset\}) \boxplus b(\{1 \cdot \emptyset\})$ describes the simple Segala system in Fig.1(i).

Stratified systems. Stratified systems are coalgebras of the functor $D_\omega(Id) + (B \times Id) + 1$. Each state of these systems either performs unlabelled probabilistic transitions or one B-labelled transition or it terminates. To get the intuition for the syntax presented in Table 1, note that the stratified system in Fig.1.(ii) would be specified by the expression $1/2 \cdot (1/3 \cdot \langle a, \downarrow\rangle \oplus 2/3 \cdot \langle b, \downarrow\rangle) \oplus 1/2 \cdot \langle a, \downarrow\rangle$. Again, we added some syntactic sugar to our original regular expressions: $\downarrow$, denoting termination, corresponds to our expression $r[r[1]]$, while $\langle b, \varepsilon\rangle$ corresponds to $r[l[l(b) \oplus r(\{\varepsilon\})]]$. The derivation of the simplified syntax and axioms follows a similar strategy as in the previous example and thus is omitted here. As described in Section 5, we first derive expressions and axioms for $\mathbb{R}^{Id} + (B \times \mathcal{P}_\omega(Id)) + 1$ and then we restrict the syntax to characterize only $D_\omega(Id) + (B \times Id) + 1$-behaviours.

Pnueli-Zuck systems. These systems are coalgebras of the functor $\mathcal{P}_\omega D_\omega(\mathcal{P}_\omega(Id))^A$. Intuitively, the ingredient $\mathcal{P}_\omega(Id)^A$ denotes A-labelled transitions to other states. Then, $D_\omega(\mathcal{P}_\omega(Id))^A$ corresponds to a probability distribution of labelled transitions and then, each state of a $\mathcal{P}_\omega D_\omega(\mathcal{P}_\omega(Id))^A$-coalgebra performs a non deterministic choice amongst probability distributions of labelled transitions. 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 Fig.1 (iii). Notice that we use the same symbol $\boxplus$ for denoting two different kinds of non-deterministic choice. This is safe, since they satisfy the same axioms. Again, the derivation of the simplified syntax and axioms is omitted here.

7 Conclusions

We presented a general framework to canonically derive expressions and axioms for quantitative regular behaviours. To illustrate the effectiveness and generality of our approach we derived expressions and equations for weighted automata, simple Segala, stratified and Pnueli-Zuck systems.

We recovered the syntaxes proposed in [10, 14, 38] for the first three models and the axiomatization of [14]. For weighted automata and stratified systems we derived new axiomatizations and for Pnueli-Zuck systems both a novel language of expressions and axioms. It should be remarked that [10, 14, 38] considered process calculi that are also equipped with the parallel composition operator and thus they slightly differ from our languages, which are more in the spirit of Kleene and Milner's expressions. Also [4, 13, 37] study expressions without parallel composition 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 [13], while the expressions in [37] differ from the ones resulting from our approach, since they use a probabilistic choice operator $+_p$. For alternating systems, our approach would bring some new insights, since [4] considers only expressions without recursion.

Acknowledgments. The authors are grateful to Catuscia Palamidessi for interesting discussions and pointers to the literature.

References

1. Aceto, L., Ésik, Z., Ingólfsdóttir, A.: Equational axioms for probabilistic bisimilarity. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 239–253. Springer, Heidelberg (2002)
2. Baeten, J., Bergstra, J., Smolka, S.: Axiomization probabilistic processes: Acp with generative probabililties (extended abstract). In: Cleaveland [11], pp. 472–485
3. Baeten, J., Klop, J. (eds.): CONCUR 1990. LNCS, vol. 458. Springer, Heidelberg (1990)
4. Bandini, E., Segala, R.: Axiomatizations for probabilistic bisimulation. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 370–381. Springer, Heidelberg (2001)
5. Bartels, F., Sokolova, A., de Vink, E.: A hierarchy of probabilistic system types. TCS 327(1-2), 3–22 (2004)
6. Bonchi, F., Bonsangue, M., Rutten, J., Silva, A.: Deriving syntax and axioms for quantitative regular behaviours. CWI technical report (2009)
7. Bonsangue, M., Rutten, J., Silva, A.: An algebra for Kripke polynomial coalgebras. In: LICS (to appear, 2009)
8. Brzozowski, J.: Derivatives of regular expressions. Journal of the ACM 11(4), 481–494 (1964)
9. Buchholz, P.: Bisimulation relations for weighted automata. TCS 393(1-3), 109–123 (2008)
10. Buchholz, P., Kemper, P.: Quantifying the dynamic behavior of process algebras. In: de Luca, L., Gilmore, S. (eds.) PROBMIV 2001, PAPM-PROBMIV 2001, and PAPM 2001. LNCS, vol. 2165, pp. 184–199. Springer, Heidelberg (2001)
11. Cleaveland, R. (ed.): CONCUR 1992. LNCS, vol. 630. Springer, Heidelberg (1992)
12. D'Argenio, P., Hermanns, H., Katoen, J.-P.: On generative parallel composition. ENTCS 22 (1999)
13. Deng, Y., Palamidessi, C.: Axiomatizations for Probabilistic Finite-State Behaviors. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 110–124. Springer, Heidelberg (2005)

14. Deng, Y., Palamidessi, C., Pang, J.: Compositional reasoning for probabilistic finite-state behaviors. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 309–337. Springer, Heidelberg (2005)
15. Droste, M., Gastin, P.: Weighted Automata and Weighted Logics. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 513–525. Springer, Heidelberg (2005)
16. Giacalone, A., Jou, C., Smolka, S.: Algebraic reasoning for probabilistic concurrent systems. In: Broy, Jones (eds.) Proc. of IFIP TC 2 (1990)
17. Gumm, H., Schröder, T.: Monoid-labeled transition systems. ENTCS 44(1) (2001)
18. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comp. 6(5), 512–535 (1994)
19. Jacobs, B.: Many-sorted coalgebraic modal logic: a model-theoretic study. ITA 35(1), 31–59 (2001)
20. Jacobs, B.: A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 375–404. Springer, Heidelberg (2006)
21. Jou, C., Smolka, S.: Equivalences, congruences, and complete axiomatizations for probabilistic processes. In: Baeten, Klop [3], pp. 367–383
22. Kleene, S.: Representation of events in nerve nets and finite automata. Automata Studies, 3–42 (1956)
23. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. In: Logic in Computer Science, pp. 214–225 (1991)
24. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inf. Comp. 94(1), 1–28 (1991)
25. Larsen, K., Skou, A.: Compositional verification of probabilistic processes. In: Cleaveland [11], pp. 456–471
26. Milner, R.: A complete inference system for a class of regular behaviours. J. Comp. Syst. Sci. 28(3), 439–466 (1984)
27. Mislove, M., Ouaknine, J., Worrell, J.: Axioms for probability and nondeterminism. ENTCS 96, 7–28 (2004)
28. Pnueli, A., Zuck, L.: Probabilistic verification by tableaux. In: LICS, pp. 322–331. IEEE, Los Alamitos (1986)
29. Rabin, M.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
30. Rößiger, M.: Coalgebras and modal logic. ENTCS 33 (2000)
31. Rutten, J.: Universal coalgebra: a theory of systems. TCS 249(1), 3–80 (2000)
32. Salomaa, A.: Two complete axiom systems for the algebra of regular events. J. ACM 13(1), 158–169 (1966)
33. Schützenberger, M.: On the definition of a family of automata. Information and Control 4(2-3), 245–270 (1961)
34. Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis, MIT (1995)
35. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 481–496. Springer, Heidelberg (1994)
36. Smolka, S., Steffen, B.: Priority as extremal probability. In: Baeten, Klop [3], pp. 456–466
37. Stark, E., Smolka, S.: A complete axiom system for finite-state probabilistic processes. In: Plotkin, et al. (eds.) Proof, Language, and Interaction, pp. 571–596. MIT Press, Cambridge (2000)
38. van Glabbeek, R., Smolka, S., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Inf. Comput. 121(1), 59–80 (1995)
39. Vardi, M.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338. IEEE, Los Alamitos (1985)