An algebra for Kripke polynomial coalgebras

Marcello Bonsangue
LIACS - Leiden University
The Netherlands
marcello@liacs.nl

Jan Rutten
Centrum voor Wiskunde en Informatica and
Vrije Universiteit Amsterdam
The Netherlands
janr@cwi.nl

Alexandra Silva*
Centrum voor Wiskunde en Informatica
The Netherlands
ams@cwi.nl

Abstract

Several dynamical systems, like deterministic automata and labelled transition systems, can be described as coalgebras of so-called Kripke polynomial functors, built up from constants and identities, using product, coproduct and powerset. Locally finite Kripke polynomial coalgebras can be characterized up to bisimulation by a specification language that generalizes Kleene's regular expressions for finite automata. In this paper, we equip this specification language with an axiomatization and prove it sound and complete with respect to bisimulation, using a purely coalgebraic argument. We demonstrate the usefulness of our framework by providing an alternative finite equational system for (non-)deterministic finite automata, labelled transition systems with explicit termination, and automata on guarded strings.

1 Introduction

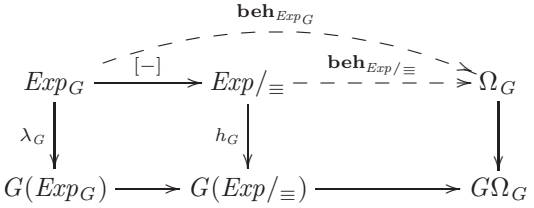
Regular expressions and finite automata can be seen as two different representations of regular languages. The former constitute an algebraic description of languages and have many applications in string processing and specification of systems. The connection between these two formalisms is made explicit in Kleene's theorem, one of the cornerstones of theoretical computer science, that states the correspondence of languages recognized by finite automata and those represented by regular expressions.

It was showed in [5] that Kleene's theorem can be generalized to other types of transition systems. Finite automata are replaced by G-coalgebras (for a polynomial **Set** endofunctor G), regular expressions by a language Exp_G modularly constructed for each G, and language equivalence by the bisimilarity relation $\sim$. Here, a G-coalgebra is a set of states S together with a transition function $g : S \to GS$, where the functor G determines the type of the system. For instance, deterministic automata are coalgebras for the functor $2 \times Id^A$.

The next natural question is whether there exists a finite, sound and complete axiomatization of Exp_G that will allow to manipulate expressions in the same way as Kleene algebra is used for regular expressions. In this paper, we enlarge the class of functors for which Exp_G is defined, by adding the powerset functor. Additionally, we define a finite axiomatization for Exp_G and we prove it sound and complete. For this, we introduce an equivalence relation $\equiv$ on expressions. Thus, altogether, we present a framework that provides an automatic way of deriving specification languages, accompanied by a sound and complete axiomatization, for a wide range of systems, including non-deterministic ones such as labelled transition systems. It is important to note that because the language contains fixed-points, allowing to represent systems with recurrence, providing a sound and complete axiomatization is challenging. We believe that the inherently coalgebraic nature of our approach has been instrumental in achieving

*Partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

this. The beauty of the proof resides in the fact that it can be concisely captured by the following diagram.



We focus on the well-definedness of the constituents and the commutativity of the diagram. First, we explain the elements of the diagram and then address how the soundness and completeness will follow using a coalgebraic reasoning.

The sets Exp_G, $Exp/_\equiv$ and Ω_G denote, respectively, the set of expressions associated with a functor G, the set of expressions modulo $\equiv$ and the final G-coalgebra. The function $[-]$ is the canonical map induced by $\equiv$. The set of expressions Exp_G has a coalgebraic structure [5], given by λ_G. We will show that $Exp_G/_\equiv$ inherits such coalgebraic structure, *i.e.*, that there exists a function h_G such that $[-]$ is an homomorphism of coalgebras. The maps $\mathbf{beh}_{Exp_G}$ and $\mathbf{beh}_{Exp/_\equiv}$ are the *unique* maps into the final coalgebra, which implies

$$\mathbf{beh}_{Exp_G} = \mathbf{beh}_{Exp/_\equiv} \circ [-] \qquad (\star)$$

since $[-]$ is a homomorphism.
Given two expressions ε_1 and ε_2, the equivalence

$$\varepsilon_1 \equiv \varepsilon_2 \iff \varepsilon_1 \sim \varepsilon_2$$

represents soundness ($\Rightarrow$) and completeness ($\Leftarrow$). Using the remarks we made about the above diagram we reason as follows:

$$\begin{array}{lcl} \varepsilon_1 \equiv \varepsilon_2 & \Leftrightarrow & [\varepsilon_1] = [\varepsilon_2] \\ & \overset{\dagger}{\Leftrightarrow} & \mathbf{beh}_{Exp/_\equiv}([\varepsilon_1]) = \mathbf{beh}_{Exp/_\equiv}([\varepsilon_2]) \\ & \overset{(\star)}{\Leftrightarrow} & \mathbf{beh}_{Exp_G}(\varepsilon_1) = \mathbf{beh}_{Exp_G}(\varepsilon_2) \\ & \Leftrightarrow & \varepsilon_1 \sim \varepsilon_2 \end{array}$$

The last step is a consequence of the fact that $\mathbf{beh}_{Exp_G}$ identifies only bisimilar objects. The step marked by † follows trivially in one direction ($\Rightarrow$, corresponding to soundness) and in the other only if $\mathbf{beh}_{Exp/_\equiv}$ is injective. In summary, the two crucial properties of the diagram are: (1) $[-]$ being a homomorphism (which follows from the well-definedness of h_G and implies $(\star)$) and (2) $\mathbf{beh}_{Exp/_\equiv}$ being injective. The proof of soundness only needs (1), while the proof of completeness requires both (1) and (2).

In the rest of the paper, we will fill in the details of this proof. We will first recall the main definitions and results concerning the language of expressions associated to a polynomial functor G and extend them to Kripke polynomial functors (Sections 2 and 3) by adding the powerset functor. We will then present the axiomatization (Section 4) and prove it is sound and complete w.r.t bisimulation (Section 5). We illustrate each step with two running examples, deterministic and non-deterministic automata. We also briefly sketch two other applications of our framework (Section 6): labelled transition systems and automata on guarded strings, which are intimately related with basic process algebra and Kleene algebra with tests. Conclusions and directions for future research are presented in Section 7 and related work is discussed in Section 7.1.

2 Preliminaries

We present the basic definitions for Kripke polynomial functors and coalgebras and introduce the notion of bisimulation.

Let $\mathbf{Set}$ be the category of sets and functions. Sets are denoted by capital letters $X, Y, \ldots$ and functions by lower case $f, g, \ldots$. We write $\{\}$ for the empty set and the collection of all *finite* subsets of a set X is defined as $\mathcal{P}X = \{ Y \subseteq X \mid Y \text{ finite}\}$. The collection of functions from a set X to a set Y is denoted by Y^X. We write $g \circ f$ 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 typically written as $1 = \{*\}$. It can be regarded as the empty product. 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.

Kripke polynomial functors. In our definition of Kripke 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 Kripke 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 KPF of *Kripke polynomial functors* on $\mathbf{Set}$ is inductively defined by putting:

$$G ::= Id \mid B \mid G + G \mid G \times G \mid G^A \mid \mathcal{P}G$$

with B a finite join-semilattice and A a finite set.

Typical examples of Kripke polynomial functors are $D = 2 \times Id^A$, $St = A \times Id$, $N = 2 \times (\mathcal{P}Id)^A$ and $P = (1 + Id)^A$. These functors represent, respectively, the type of *deterministic automata*, *infinite streams*, *non-deterministic automata* and *partial deterministic automata*.

Our definition of Kripke polynomial functors slightly differs from the one of [13, 6] 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. In fact, as we will show, such a language (for this class of functors) is a coalgebra. The intuition behind these extensions becomes clear if one recalls that the set of classical regular expressions carries a join-semilattice structure. If we want to generalize this notion for Kripke polynomial functors then we must guarantee that they have also such structure, namely by imposing it in the constant and $+$. For the $\times$ and $(-)^A$ we do not need to add extra elements because the structure is compositionally inherited.

Next, we give the definition of the ingredient relation, which relates a Kripke polynomial functor G with its *ingredients*, *i.e.* the functors used in its inductive construction. We shall use this relation for typing our expressions.

Let $\lhd \subseteq KPF \times KPF$ be the least reflexive and transitive relation, written infix, on Kripke polynomial functors such that

$$\begin{array}{lll} G_1 \lhd G_1 \times G_2, & G_2 \lhd G_1 \times G_2, & G \lhd \mathcal{P}G, \\ G_1 \lhd G_1 + G_2, & G_2 \lhd G_1 + G_2, & G \lhd G^A. \end{array}$$

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* or dynamics of the G-coalgebra [14]. Classical examples of coalgebras are deterministic automata, infinite streams, non-deterministic automata and partial automata, which are, respectively, coalgebras for the functors D, St, N and P 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 Kripke polynomial functor G there exists a final G-coalgebra (Ω_G, ω_G) [14]. The notion of finality plays a key role in defining the semantics of expressions below.

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, *i.e.* the set 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.

Next we define bisimulation, which plays an important role in the minimization of coalgebras. We will also use bisimulation as a semantic equivalence for our language of expressions.

Let (S, f) and (T, g) be two G-coalgebras. A relation $R \subseteq S \times T$ is called a *bisimulation* [3] if there exists a map $e : R \to G(R)$ such that the projections π_1 and π_2 are coalgebra homomorphisms, *i.e.* the following diagram commutes.

$$\begin{array}{ccccc} S & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & T \\ {\scriptstyle f}\downarrow & & {\scriptstyle \exists e}\downarrow & & \downarrow{\scriptstyle g} \\ FS & \xleftarrow[F\pi_2]{} & FR & \xrightarrow[F\pi_1]{} & FT \end{array}$$

We write $s \sim_G t$ whenever there exists a bisimulation relation containing (s, t) and we call $\sim_G$ the bisimilarity relation. We shall drop the subscript G whenever the functor G is clear from the context. For G-coalgebras (S, f) and (T, g) and $s \in S$, $t \in T$, the maps into the final coalgebra $\mathbf{beh}_S$ and $\mathbf{beh}_T$ have the following important property: $s \sim t \Leftrightarrow \mathbf{beh}_S(s) = \mathbf{beh}_T(t)$. The implication

$\Rightarrow$ holds for all functors G and $\Leftarrow$ holds for some functors including all Kripke polynomial functors.

3 A language of expressions for Kripke polynomial coalgebras

In this section, we recall the main definitions and results concerning the language of expressions associated to a polynomial functor G, introduced in [5], and extend them to Kripke polynomial functors. We also present the generalization of Kleene's theorem, paving the way for the axiomatization in the next section. 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 Kripke polynomial functors.

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:

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

where $a \in A$, $b \in B$ and γ is a *guarded expression* given by:

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

An expression is *closed* if it has no free occurrences of fixpoint variables x. We denote the set of guarded and 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 $l(\varepsilon)$, $r(\varepsilon)$, $l[\varepsilon]$, $r[\varepsilon]$, $a(\varepsilon)$ and $\{\varepsilon\}$ denote the left and right hand-side of products and sums, function application and singleton set, respectively. Here, it is already visible that our approach for the powerset functor differs from classical approaches where $\Box$ and $\Diamond$ are used. This is a choice, justified by the fact that our goal is to have a "process algebra" like language instead of a modal logic one. It also explains why we only consider finite powersets: every finite set can be written as the finite union of its singletons.

Next, we present a typing assignment system that will allow us to associate with each functor G the expressions $\varepsilon \in Exp$ 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 $\varepsilon \in Exp$ using the ingredient relation, for $a \in A$, $b \in B$ and $x \in X$, as follows:

$$\frac{}{\vdash \emptyset : F \lhd G} \qquad \frac{}{\vdash b : B \lhd G}$$

$$\frac{}{\vdash x : G \lhd G} \qquad \frac{\vdash \varepsilon : G \lhd G}{\vdash \mu x.\varepsilon : G \lhd G}$$

$$\frac{\vdash \varepsilon_1 : F \lhd G \quad \vdash \varepsilon_2 : F \lhd G}{\vdash \varepsilon_1 \oplus \varepsilon_2 : F \lhd G} \qquad \frac{\vdash \varepsilon : F \lhd G}{\vdash \{\varepsilon\} : \mathcal{P}F \lhd G}$$

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

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l(\varepsilon) : F_1 \times F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r(\varepsilon) : F_1 \times F_2 \lhd G}$$

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l[\varepsilon] : F_1 + F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r[\varepsilon] : F_1 + F_2 \lhd G}$$

Most of the rules are self-explanatory. The rule involving $Id \lhd G$ reflects the isomorphism between the final coalgebra Ω_G and $G(\Omega_G)$. For further details we refer to [5].

We can now formally define the set of G-expressions: well-typed expressions associated with a Kripke polynomial functor G.

Definition 1 (G-expressions) Let G be a Kripke polynomial functor and F an ingredient of G. We denote by $Exp_{F \lhd G}$ the set:

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

We define the set Exp_G of well-typed G-*expressions* by $Exp_{G \lhd G}$. ♣

To illustrate this definition we instantiate it for the functors $D = 2 \times Id^A$ and $N = 2 \times (\mathcal{P}Id)^A$.

Example 2 (Deterministic expressions) Let A be a finite set of input actions 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 and 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 full composition and star, respectively. These two are semantically equivalent: as we will soon formally state (Theorem 4) the expressions in our syntax are in one-to-one correspondence to deterministic automata, hence equivalent to classical regular expressions.

Example 3 (Non-Deterministic expressions) Let A be a finite set of input actions and let X be a set of fixpoint variables. The set Exp_N of well-typed N-expressions is given by the BNF:

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

where $a \in A$, $x \in X$ and restrictions to ε as before, as can be straightforwardly verified by mutual induction. ♠

The language of expressions induces an algebraic description of systems. In [5], 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. We will reproduce the definition of that function here and add the extra clause relative to the $\mathcal{P}$ functor. The definition makes use of the following two auxiliary constructs.

(i) We define the constant $Empty_{F \lhd G} \in F(Exp_G)$ by induction on the syntactic structure of F:

$$\begin{array}{lcl} Empty_{Id \lhd G} & = & \emptyset \\ Empty_{B \lhd G} & = & \bot_B \\ Empty_{F_1 \times F_2 \lhd G} & = & \langle Empty_{F_1 \lhd G}, Empty_{F_2 \lhd G} \rangle \\ Empty_{F_1 + F_2 \lhd G} & = & \bot \\ Empty_{F^A \lhd G} & = & \lambda a. Empty_{F \lhd G} \\ Empty_{\mathcal{P} F \lhd G} & = & \{\} \end{array}$$

(ii) We define the function

$$Plus_{F \lhd G} : F(Exp_G) \times F(Exp_G) \to F(Exp_G)$$

by induction on the syntactic structure of F:

$$Plus_{Id \lhd G}(\varepsilon_1, \varepsilon_2) = \varepsilon_1 \oplus \varepsilon_2$$
$$Plus_{B \lhd G}(b_1, b_2) = b_1 \vee_B b_2$$
$$Plus_{F_1 \times F_2 \lhd G}(\langle \varepsilon_1, \varepsilon_2 \rangle, \langle \varepsilon_3, \varepsilon_4 \rangle) = \langle Plus_{F_1 \lhd G}(\varepsilon_1, \varepsilon_3), Plus_{F_2 \lhd G}(\varepsilon_2, \varepsilon_4) \rangle$$
$$Plus_{F_1 + F_2 \lhd G}(\kappa_i(\varepsilon_1), \kappa_i(\varepsilon_2)) = \kappa_i(Plus_{F_i \lhd G}(\varepsilon_1, \varepsilon_2)), \quad i \in \{1, 2\}$$
$$Plus_{F_1 + F_2 \lhd G}(\kappa_i(\varepsilon_1), \kappa_j(\varepsilon_2)) = \top \quad \text{for } i, j \in \{1, 2\} \text{ and } i \neq j$$
$$Plus_{F_1 + F_2 \lhd G}(x, \top) = Plus_{F_1 + F_2 \lhd G}(\top, x) = \top$$
$$Plus_{F_1 + F_2 \lhd G}(x, \bot) = Plus_{F_1 + F_2 \lhd G}(\bot, x) = x$$
$$Plus_{F^A \lhd G}(f, g) = \lambda a.\ Plus_{F \lhd G}(f(a), g(a))$$
$$Plus_{\mathcal{P} F \lhd G}(s_1, s_2) = s_1 \cup s_2$$

Now we define $\lambda_{F \lhd G}$. We do this by double induction on the maximum number $N(\varepsilon)$ of nested unguarded occurrences of μ-expressions in ε and on the length of the proofs for typing expressions. We define $N(\varepsilon)$ as follows:

$$N(\varepsilon) = 0 \quad \text{for } \varepsilon \in \{\emptyset, b, a(\varepsilon'), l(\varepsilon'), r(\varepsilon'), l[\varepsilon'], r[\varepsilon'], \{\varepsilon'\}\}$$
$$N(\varepsilon_1 \oplus \varepsilon_2) = max\{N(\varepsilon_1),\ N(\varepsilon_2)\}$$
$$N(\mu x.\varepsilon) = 1 + N(\varepsilon)$$

For every ingredient F of a Kripke polynomial functor G and expression $\varepsilon \in Exp_{F \lhd G}$, the mapping $\lambda_{F \lhd G}(\varepsilon)$ is defined as follows:

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

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

We now present the generalization of Kleene's theorem, also paving the way for the axiomatization in the next section.

Theorem 4 Let G be a Kripke polynomial functor.

1. For every locally finite G-coalgebra (S, g) and for any $s \in S$ there exists an expression $\langle\!\langle\, s \,\rangle\!\rangle \in Exp_G$ such that $\langle\!\langle\, s \,\rangle\!\rangle \sim s$.
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 item 1. implies $\mathbf{beh}(\langle\!\langle\, s \,\rangle\!\rangle) = \mathbf{beh}(s)$. This theorem generalizes Theorems 5 and 6 of [5], where only finite polynomial coalgebras were considered. For reason of space we will omit the proof here. We will show next how to construct $\langle\!\langle\, s \,\rangle\!\rangle$, which we will need in the sequel.

Definition 5 Let G be a Kripke polynomial functor and (S, g) a locally finite G-coalgebra. We construct, for a given state $s \in S$, an expression $\langle\!\langle\, s \,\rangle\!\rangle$, such that $\langle\!\langle\, s \,\rangle\!\rangle \sim s$. If $G = Id$, $\langle\!\langle\, s \,\rangle\!\rangle = \emptyset$. Otherwise we proceed as follows. Let $\langle s \rangle = \{s_1, s_2, \ldots, s_n\}$, where $s_1 = s$. We associate with each state s_i a variable $x_i \in X$ and an equation $\varepsilon_i = \mu x_i.\gamma^G_{g(s_i)}$.

For $F \lhd G$ and $s' \in FS$, the expression $\gamma^F_{s'} \in Exp_{F \lhd G}$ is defined by induction on the structure of F:

$$\gamma^{Id}_s = x_s \qquad \gamma^B_b = b$$
$$\gamma^{F_1 \times F_2}_{\langle s,s' \rangle} = l(\gamma^{F_1}_s) \oplus r(\gamma^{F_2}_{s'})$$
$$\gamma^{F_1+F_2}_{\kappa_1(s)} = l[\gamma^{F_1}_s] \qquad \gamma^{F_1+F_2}_{\kappa_2(s)} = r[\gamma^{F_2}_s]$$
$$\gamma^{F_1+F_2}_{\bot} = \emptyset \qquad \gamma^{F_1+F_2}_{\top} = l[\emptyset] \oplus r[\emptyset]$$
$$\gamma^{F^A}_f = \bigoplus_{a \in A} a(\gamma^F_{f(a)})$$
$$\gamma^{\mathcal{P}F}_S = \begin{cases} \bigoplus_{s \in S} \{\gamma^F_s\} & S \neq \{\} \\ \emptyset & S = \{\} \end{cases}$$

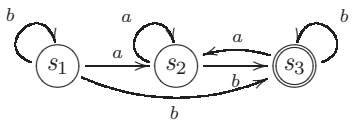
Note that the choice of $l[\emptyset] \oplus r[\emptyset]$ to represent inconsistency is arbitrary but *canonical*, in the sense that any other expression involving sum of $l[\varepsilon_1]$ and $r[\varepsilon_2]$ will be bisimilar, as it will become clear in the axiomatization.

Next we eliminate from our system of equations

$$\varepsilon_1 = \mu x_1.\gamma^G_{g(s_1)}, \ldots, \varepsilon_n = \mu x_n.\gamma^G_{g(s_n)}$$

all free occurrences of $x_1, \ldots, x_n$ by first replacing x_n by ε_n in all equations for $\varepsilon_1, \ldots, \varepsilon_{n-1}$. Next, we replace x_{n-1} by ε_{n-1} in the equations for $\varepsilon_1, \ldots, \varepsilon_{n-2}$. Continuing in this way, we end up with an equation $\varepsilon_1 = \varepsilon$, where ε no longer contains any free variable. We then take $\langle\!\langle\, s \,\rangle\!\rangle = \varepsilon$. ♣

Let us illustrate the construction above. Consider the following non-deterministic automaton over the alphabet $A = \{a, b\}$, whose transition function t is depicted in the following picture (with s_3 as final state):



To compute an expression bisimilar to s_1, we first observe that $\langle s_1 \rangle = \{s_1, s_2, s_3\}$ and we define ε_1 ε_2 and ε_3 by $\varepsilon_i = \mu x_i.\, \gamma^D_{t(s_i)}$, where

$$\gamma^D_{t(s_1)} = l(0) \oplus r(a(\{x_2\}) \oplus b(\{x_1\} \oplus \{x_3\}))$$
$$\gamma^D_{t(s_2)} = l(0) \oplus r(a(\{x_2\}) \oplus b(\{x_3\}))$$
$$\gamma^D_{t(s_3)} = l(1) \oplus r(a(\{x_2\}) \oplus b(\{x_3\}))$$

We start with the system:

$$\varepsilon_1 = \mu x_1.l(0) \oplus r(a(\{x_2\}) \oplus b(\{x_1\} \oplus \{x_3\}))$$
$$\varepsilon_2 = \mu x_2.l(0) \oplus r(a(\{x_2\}) \oplus b(\{x_3\}))$$
$$\varepsilon_3 = \mu x_3.l(1) \oplus r(a(\{x_2\}) \oplus b(\{x_3\}))$$

Replacing x_3 by ε_3 in ε_2 and ε_1 yields

$$\varepsilon_1 = \mu x_1.l(0) \oplus r(a(\{x_2\}) \oplus b(\{x_1\} \oplus \{\varepsilon_3\}))$$
$$\varepsilon_2 = \mu x_2.l(0) \oplus r(a(\{x_2\}) \oplus b(\{\varepsilon_3\}))$$

Note that at this point ε_2 already denotes a closed expression and the expression denoted by ε_1 only contains two free occurrences of the variable x_2. Finally, replacing x_2 by ε_2 in ε_1 results in the following closed expression

$$\varepsilon_1 = \mu x_1.l(0) \oplus r(a(\{\varepsilon_2\}) \oplus b(\{x_1\} \oplus \{\varepsilon'_3\}))$$

where $\varepsilon'_3 = \varepsilon_3[\varepsilon_2/x_2]$.

We should also remark that in the examples we remain faithful to the automatically derived syntax. However, it is obvious that many simplifications can be made in order to obtain a more polished language. For instance, $l(0)$ and $l(1)$ could be abbreviated by 0 and 1 and $r(a(\varepsilon))$ by $a(\varepsilon)$ without any risk of confusion. In Section 6 we will sketch two examples where we apply some simplification to the syntax.

4 Axiomatization

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

1. $(Exp_{F \lhd G}, \oplus, \emptyset)$ is a join-semilattice.

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

2. μ is a least fixed-point.

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

3. The join-semilattice structure propagates

through the expressions.

$$
\begin{array}{ll}
(B-\emptyset) & \emptyset \equiv \bot_B \\
(B-\oplus) & b_1 \oplus b_2 \equiv b_1 \vee_B b_2 \\
(\times-\emptyset-L) & l(\emptyset) \equiv \emptyset \\
(\times-\oplus-L) & l(\varepsilon_1 \oplus \varepsilon_2) \equiv l(\varepsilon_1) \oplus l(\varepsilon_2) \\
(\times-\emptyset-R) & r(\emptyset) \equiv \emptyset \\
(\times-\oplus-R) & r(\varepsilon_1 \oplus \varepsilon_2) \equiv r(\varepsilon_1) \oplus r(\varepsilon_2) \\
(+-\oplus-L) & l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2] \\
(+-\oplus-R) & r[\varepsilon_1 \oplus \varepsilon_2] \equiv r[\varepsilon_1] \oplus r[\varepsilon_2] \\
(+-\oplus-\top) & r[\varepsilon_1] \oplus l[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset] \\
(-^A-\emptyset) & a(\emptyset) \equiv \emptyset \\
(-^A-\oplus) & a(\varepsilon_1 \oplus \varepsilon_2) \equiv a(\varepsilon_1) \oplus a(\varepsilon_2)
\end{array}
$$

4. $\equiv$ is a congruence.

   $(Cong)$ If $\varepsilon_1 \equiv \varepsilon_1'$ and $\varepsilon_2' \equiv \varepsilon_2'$ then:
   $\varepsilon_1 \oplus \varepsilon_2 \equiv \varepsilon_1' \oplus \varepsilon_2'$, $\mu x.\varepsilon_1 \equiv \mu x.\varepsilon_1'$,
   $l(\varepsilon_1) \equiv l(\varepsilon_1')$, $r(\varepsilon_1) \equiv r(\varepsilon_1')$,
   $l[\varepsilon_1] \equiv l[\varepsilon_1']$, $r[\varepsilon_1] \equiv r[\varepsilon_1']$,
   $\{\varepsilon_1\} = \{\varepsilon_1'\}$ and $a(\varepsilon_1) \equiv a(\varepsilon_1')$.

5. α-equivalence

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

We shall write $Exp/_{\equiv}$ for the set of expressions modulo $\equiv$. It is important to remark that in the third group of rules it does not exist any rule applicable to expressions of type $\mathcal{P}F$.

Example 6 Consider the non-deterministic automata over the alphabet $A = \{a\}$:

Applying Definition 5 one can easily compute the expressions correspondent to s_1 and s_1':

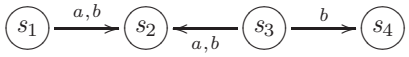
$$\varepsilon_1 = \mu x.l(0) \oplus r(a(\{x\}))$$
$$\varepsilon_1' = \mu x.l(0) \oplus r(a(\{\mu y.l(0) \oplus r(a(\{x\}))\}))$$

We now prove that $\varepsilon_1 \equiv \varepsilon_1'$. We achieve this by proving that both ε_1 and ε_1' are equivalent to $\varepsilon_1' \oplus \varepsilon_1$. For space reasons we will only include the proof $\varepsilon_1' \oplus \varepsilon_1 \equiv \varepsilon_1$. The other equivalence is proved in the exactly same way. In the following calculations let $\varepsilon = \mu x.r(a(\{x\}))$.

$$
\begin{array}{ll}
 & \varepsilon_1' \oplus \varepsilon_1 \equiv \varepsilon_1 \\
\overset{\dagger}{\Leftarrow} & \mu x.r(a(\{r(a(\{x\}))\})) \oplus \varepsilon \equiv \varepsilon \\
\overset{(LFP)}{\Leftarrow} & a(\{a(\{\varepsilon\})\}) \oplus \varepsilon \equiv \varepsilon \\
\overset{(FP)}{\Leftarrow} & a(\{\varepsilon\}) \oplus \varepsilon \equiv \varepsilon \\
\overset{(FP)}{\Leftarrow} & \varepsilon \oplus \varepsilon \equiv \varepsilon
\end{array}
$$

The last step follows trivially because of the $(Idempotency)$ rule and the step marked by † follows by applying $(B-\emptyset)$, $(\times-\emptyset-L)$ and $(Empty)$ to eliminate all the factors $l(0)$ and (FP) to also eliminate μy in ε_1'. Moreover, the $(Cong)$ rule was used in almost every step.

Consider the non-deterministic automata over the alphabet $A = \{a, b\}$:



Using Definition 5 one can easily compute the expressions correspondent to s_2 and s_4 and see that they are equivalent to $\emptyset$. Then, using this, one can compute the following expressions for s_1 and s_3:

$$\varepsilon_1 = \mu x_1.l(0) \oplus r(a(\{\emptyset\}) \oplus b(\{\emptyset\}))$$
$$\varepsilon_3 = \mu x_3.l(0) \oplus r(a(\{\emptyset\} \oplus b(\{\emptyset\} \oplus \{\emptyset\})))$$

Applying the rules $(Idempotency)$ (in ε_3) and (FP) it is straightforward to see that $\varepsilon_1 \equiv \varepsilon_3$. ♠

We can immediately define an F-coalgebra $h_{F \lhd G} : Exp/_{\equiv} \to FExp/_{\equiv}$ by setting $h_{F \lhd G}([\varepsilon]) = (F[-]) \circ \lambda_{F \lhd G}(\varepsilon)$. As before, h_G abbreviates $h_{G \lhd G}$. The following lemma guarantees that h is well-defined.

Lemma 7 Let $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$. Then,

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow h_{F \lhd G}([\varepsilon_1]) = h_{F \lhd G}([\varepsilon_2])$$

We present the proof of this lemma in appendix A. The result is proved by induction on the length of derivations of $\equiv$. For the first group of axioms, it is a direct consequence of the fact that the join-semilattice structure of $(Exp_{F \lhd G}, \oplus, \emptyset)$ is transferred to $(F(Exp_{F \lhd G}), Plus_{F \lhd G}, Empty_F)$ (easy proof by induction on the structure of F). The third and fourth group of axioms, as well as the (FP) rule, follow easily using the definition of $h_{F \lhd G}$ and induction. The most interesting case is in fact the one for the rule (LFP), which requires the following extra lemma (also proved in appendix A) about the interaction of h_G and syntactic substitution.

Lemma 8 Let $\varepsilon_1, \varepsilon_2, \gamma \in Exp_G$. Then,

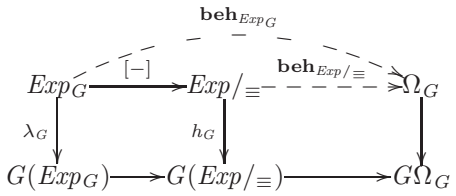
1. $\varepsilon_1 \equiv \varepsilon_2 \Rightarrow h_G([\gamma[\varepsilon_1/x]]) = h_G([\gamma[\varepsilon_2/x]])$
2. $h_G([\gamma[\varepsilon_1/x] \oplus \gamma[\varepsilon_2/x]]) = h_G([\gamma[\varepsilon_1 \oplus \varepsilon_2/x]])$

An important remark is that the coalgebra $h : Exp/_{\equiv} \to GExp/_{\equiv}$ is locally finite. This is in fact a direct consequence of point 2 of the

generalized Kleene's theorem (Theorem 4). In the proof of this theorem (details in [5]) we showed that, given $\varepsilon \in Exp_{F \lhd G}$, the subcoalgebra $\langle \varepsilon \rangle$, resulting from applying $\lambda_{F \lhd G}$ repeatedly and performing normalization at each step, is finite. Normalization was achieved by simply applying the equations $(Idempotency)$, $(Commutativity)$ and $(Associativity)$.

5 Soundness and Completeness

We have now defined all the elements in the diagram presented in the introduction, which concisely captures everything we need to prove soundness and completeness:



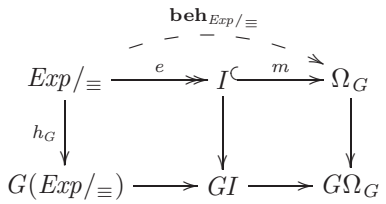
Note that the definition of h_G makes $[-]$ a homomorphism of coalgebras. This is enough to prove soundness (as we already mentioned in the introduction).

Theorem 9 (Soundness) For $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$,

$$\varepsilon_1 \equiv \varepsilon_2 \Rightarrow \varepsilon_1 \sim \varepsilon_2$$

Proof. Let $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$ and suppose that $\varepsilon_1 \equiv \varepsilon_2$. Then, $[\varepsilon_1] = [\varepsilon_2]$ and, thus $\mathbf{beh}_{Exp/_\equiv}([\varepsilon_1]) = \mathbf{beh}_{Exp/_\equiv}([\varepsilon_2])$. The uniqueness of the map $\mathbf{beh}_{Exp_G}$ and the fact that $[-]$ is a coalgebra homomorphism implies that $\mathbf{beh}_{Exp_G} = \mathbf{beh}_{Exp/_\equiv} \circ [-]$ which means that $\mathbf{beh}_{Exp_G}(\varepsilon_1) = \mathbf{beh}_{Exp_G}(\varepsilon_2)$. We also know that in the final coalgebra only the bisimilar elements are identified. Therefore, we must have $\varepsilon_1 \sim \varepsilon_2$. □

Proving completeness, as we argued in the introduction, boils down to proving that $\mathbf{beh}_{Exp_\equiv}$ is injective. We will achieve this as follows. First, we refine the rightmost square in the diagram above by inserting the image $I = \mathbf{beh}_{Exp/_\equiv}(Exp_\equiv)$:



Then, we prove that both coalgebras $Exp/_\equiv \to G(Exp/_\equiv)$ and $I \to GI$ are final in the category of locally finite G-coalgebras (Lemmas 12 and 13, respectively). Since final coalgebras are unique up to isomorphism, it follows that $e : Exp/_\equiv \to I$ is an isomorphism and therefore $\mathbf{beh}_{Exp_\equiv}$ is injective, which will give us completeness.

It is interesting to remark that in the case of the deterministic automata functor $D = 2 \times Id^A$, 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).

The proof of finality of $Exp/_\equiv \to G(Exp/_\equiv)$ and $I \to GI$ will use the following lemma.

Lemma 10 Let $S \to GS$ be a locally finite coalgebra and $\lceil - \rceil : S \to Exp/_\equiv$ the map defined by $\lceil - \rceil = [-] \circ \langle\!\langle - \rangle\!\rangle$, where $\langle\!\langle - \rangle\!\rangle : S \to Exp_G$ was presented in Definition 5. Then:

1. The map $\lceil - \rceil : S \to Exp/_\equiv$ is a coalgebra homomorphism.
2. For any homomorphism $f : S \to T$, $\lceil f(s) \rceil = \lceil s \rceil$.
3. Taking for S the set $Exp/_\equiv$, the map $\lceil - \rceil$ is the identity.

Again we present the proof of this lemma in appendix A. Properties 1. and 2. follow by induction on the structure of G and 3. by double induction on the number of nested occurrences of μ and on the length of proofs for typing expressions. Both the first and the third properties use the following extra lemma (also proved in appendix A).

Lemma 11 Let (S, g) be a locally finite G-coalgebra and $s \in S$. Then,

$$\langle\!\langle\, s \,\rangle\!\rangle \equiv \gamma^G_{g(s)}[x_t \mapsto \langle\!\langle\, t \,\rangle\!\rangle, t \in \langle s \rangle] \qquad (2)$$

Here, we use the notation $\varepsilon[x \mapsto \varepsilon_x, x \in X]$ to denote syntactic substitution of free ocurrences of all $x \in X$ in ε by ε_x.

Although at first the above lemma might seem rather cryptic, it is in fact a generalization of a very useful and intuitive equality in deterministic automata and regular expressions. Given a deterministic automaton $\langle o, \delta \rangle : S \to 2 \times S^A$ and a state $s \in S$, the associated regular expression r_s can be written as

$$r_s = o(s) + \sum_{a \in A} a \cdot r_{\delta(s)(a)}$$

In fact, instantiating (2) for $\langle o, \delta \rangle : S \to 2 \times S^A$, one can easily spot the similarity:

$$\langle\!\langle\, s \,\rangle\!\rangle \equiv l(o(s)) \oplus r(\bigoplus_{a \in A} a(\langle\!\langle\, \delta(s)(a) \,\rangle\!\rangle))$$

We can now prove that the coalgebras $Exp/_{\equiv} \to G(Exp/_{\equiv})$ and $I \to GI$ are both final in the category of locally finite G-coalgebras.

Lemma 12 The coalgebra $I \to GI$ is final in the category $Coalg(G)_{lf}$.

Proof. For any locally finite G-coalgebra (S, f), there exists a homomorphism $e \circ \lceil - \rceil : S \to I$. If there are two homomorphisms $f, g : X \to I$, then by postcomposition with the inclusion $m : I \hookrightarrow \Omega$ we get two homomorphisms into the final G-coalgebra. Thus, f and g must be equal. □

Lemma 13 The coalgebra $Exp/_{\equiv} \to G(Exp/_{\equiv})$ is final in the category $Coalg(G)_{lf}$.

Proof. For any locally finite G-coalgebra (S, f), there exists a homomorphism $\lceil - \rceil : X \to Exp/_{\equiv}$. Suppose we have two homomorphisms $f, g : S \to Exp/_{\equiv}$. Then,

$$f = id \circ f \overset{(10.3)}{=} \lceil - \rceil \circ f \overset{(10.2)}{=} \lceil - \rceil$$
$$\text{and } g = id \circ g \overset{(10.3)}{=} \lceil - \rceil \circ g \overset{(10.2)}{=} \lceil - \rceil$$

The annotations (10.2) and (10.3) mark the steps where we used items 2 and 3 of Lemma 10. □

At this point, we can conclude that the map $\mathbf{beh}_{Exp/_{\equiv}}$ is injective, since it factorizes into an isomorphism followed by a mono. This fact is the last ingredient we need to prove completeness.

Theorem 14 (Completeness) For $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$,

$$\varepsilon_1 \sim \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$$

Proof. Let $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$ 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 $\mathbf{beh}_{Exp_G}(\varepsilon_1) = \mathbf{beh}_{Exp_G}(\varepsilon_2)$ and thus $\mathbf{beh}_{Exp/_{\equiv}}([\varepsilon_1]) = \mathbf{beh}_{Exp/_{\equiv}}([\varepsilon_2])$. Now, because $\mathbf{beh}_{Exp/_{\equiv}}$ is injective we have that $[\varepsilon_1] = [\varepsilon_2]$, which implies that $\varepsilon_1 \equiv \varepsilon_2$. □

6 Two more examples

In this section we apply our framework to two other examples: labelled 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 automatically derived one. The equivalence between both syntaxes is included in Appendix B.

LTS. Labelled transition systems (with explicit termination) are coalgebras for the functor $1 + (\mathcal{P}Id)^A$. Instantiating our framework for this functor produces a language that is equivalent to the closed and guarded expressions generated by the following grammar, where $a \in A$:

$$P ::= \mathbf{0} \mid P + P \mid a.P \mid \delta \mid \surd \mid \mu x.P \mid x$$

together with the equations (omitting the congruence and α-equivalence rules)

$$\begin{array}{ll} P_1 + P_2 \equiv P_2 + P_1 & \\ P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3 & \\ P + P \equiv P & P + \mathbf{0} \equiv P \\ P + \delta \equiv P, \quad \textit{if } P \neq \surd & \\ P[\mu x.P/x] \equiv \mu x.P & \\ P[Q/x] + Q \equiv Q \Rightarrow (\mu x.P) + Q \equiv Q & \end{array}$$

Note that, as expected, there is no law that allows to prove $a.(P + Q) \equiv a.P + a.Q$. Moreover, it is interesting to observe that this axiomatization is very similar to the one presented in [2]. The differences are only in the fact that we consider action prefixing instead of sequential composition.

AGS. It has recently been shown [10] that automata on guarded strings (acceptors of the join irreducible elements of the free Kleene algebra with tests on generators Σ, T) are coalgebras for the functor $B \times Id^{At \times \Sigma}$, where At is the set of atoms, *i.e.* minimal nonzero elements of the free Boolean algebra B generated by T and Σ 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 $b \in B$ and $a \in \Sigma$:

$$P ::= \mathbf{0} \mid \langle b \rangle \mid P + P \mid b \to a.P \mid \mu x.P \mid x$$

accompanied by the equations (omitting the congruence and α-equivalence rules)

$$
\begin{array}{ll}
P_1 + P_2 \equiv P_2 + P_1 & \\
P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3 & \\
P + P \equiv P & P + \mathbf{0} \equiv P \\
\langle b_1 \rangle + \langle b_2 \rangle = \langle b_1 \vee_B b_2 \rangle & \mathbf{0} \equiv \langle \bot_B \rangle \\
(b \to a.\mathbf{0}) = \mathbf{0} & (\bot_B \to a.P) = \mathbf{0} \\
(b \to a.P_2) + (b \to a.P_2) = b \to a.(P_1 + P_2) & \\
(b_1 \to a.P) + (b_2 \to a.P) = (b_1 \vee_B b_2) \to a.P & \\
P[\mu x.P/x] \equiv \mu x.P & \\
P[Q/x] + Q \equiv Q \Rightarrow (\mu x.P) + Q \equiv Q &
\end{array}
$$

For reason of space we will not present a full comparison of this syntax to the one of Kleene algebra with tests [10] (and propositional Hoare triples). However, as a simple example consider a *program* consisting of a single action a. The Hoare partial correctness assertion $\{b\}a\{c\}$ would be written in our syntax as $b \to a.\langle \overline{c} \rangle = \mathbf{0}$.

7 Conclusions and Future Work

In this paper, we present a specification language for Kripke polynomial coalgebras, extending the results presented in [5] for polynomial coalgebras. The main contribution of this paper is a sound and complete axiomatization of such language. This is non-trivial, since the language allows for recursive specifications. The coalgebraic approach was crucial in finding the axiomatization and, more importantly, in providing a rather simple, but instructive proof of soundness and completeness.

We want to investigate automated reasoning about equality of expressions. This can be done either in a purely coalgebraic manner, by implementing Kleene's theorem in, an existing coinductive prover, such as `Circ` [8], or in an algebraic manner, by using the equations as rewriting rules. We would also like to study the precise connection between our language and coalgebraic modal logics [12, 11]. Further, we want to investigate the relation with bialgebras [15, 7].

7.1 Related work

In [9] a sound a complete axiomatization for regular expressions was presented. There, regular expressions form a Kleene algebra. i.e an idempotent semiring. Because we do not have sequential composition we only need *half* of the semiring structure: a join-semilattice. Recently, in [7], Kozen's completeness result was presented in a purely coalgebraic setting. This inspired the structure of our proof of soundness and completeness.

The third group of axioms, relating the join-semilattice structure with the functor specific operators, are similar to the ones coming from domain logic [1] or coalgebraic modal logic [4]. The main novelty of our work compared to [1, 4, 11, 12] is the inclusion of fixed-points.

Acknowledgements. The authors are indebted to F. Bonchi and E. de Vink for useful discussions.

References

[1] S. Abramsky. Domain Theory in Logical Form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991.

[2] L. Aceto and M. Hennessy. Termination, Deadlock, and Divergence. *J. ACM*, 39(1):147–187, 1992.

[3] P. Aczel and N. Mendler. A final coalgebra theorem. In *CTCS*, LNCS 389 , pp. 357–365, 1989.

[4] M. Bonsangue, A. Kurz. Presenting Functors by Operations and Equations. In *FoSSaCS*, LNCS 3921, pp. 172–186, 2006.

[5] M. Bonsangue, J. Rutten, and A. Silva. A Kleene theorem for polynomial coalgebras. In *FoSSaCS*, 2009. To appear.

[6] B. Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *ITA*, 35(1):31–59, 2001.

[7] B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Goguen Festschrift*, LNCS 4060, pp. 375–404, 2006.

[8] D. Lucanu and G. Rosu. CIRC : A Circular Coinductive Prover. In *CALCO*, LNCS 4624, pp. 372–378, 2007.

[9] D. Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inf. Comput.* 110(2):366–390, 1994.

[10] D. Kozen. On the coalgebraic theory of Kleene algebra with tests. Tech. Rep. 10173, CIS, Cornell University, 2008.

[11] C. Kupke and Y. Venema, Coalgebraic Automata Theory: Basic Results. *LMCS*, 4(4), 2008.

[12] D. Pattinson and L. Schröder. Beyond Rank 1: Algebraic Semantics and Finite Models for Coalgebraic Logics. In *FoSSaCS*, LNCS 4962, pp. 66–80, 2008.

[13] M. Rößiger. Coalgebras and modal logic. *ENTCS*, 33, 2000.

[14] J. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249(1):3–80, 2000.

[15] D. Turi and G. Plotkin. Towards a Mathematical Operational Semantics. In *LICS*, pp. 280–291, IEEE, 1997.