A decision procedure for bisimilarity of generalized regular expressions*

Marcello Bonsangue[1], Georgiana Caltais[2,3], Eugen-Ioan Goriac[2,3], Dorel Lucanu[3], Jan Rutten[4,5,6], Alexandra Silva[4]
marcello@liacs.nl, {gcaltais10, egoriac10}@ru.is,
dlucanu@info.uaic.ro, and {janr, ams}@cwi.nl

[1] LIACS - Leiden University, The Netherlands
[2] School of Computer Science - Reykjavik University, Iceland
[3] Faculty of Computer Science - Alexandru Ioan Cuza University, Romania
[4] Centrum voor Wiskunde en Informatica, The Netherlands
[5] Radboud University Nijmegen, The Netherlands
[6] Vrije Universiteit Amsterdam, The Netherlands

Abstract. A notion of generalized regular expressions for a large class of systems modeled as coalgebras, and an analogue of Kleene's theorem and Kleene algebra, were recently proposed by a subset of the authors of this paper. Examples of the systems covered include infinite streams, deterministic automata and Mealy machines. In this paper, we present a tool where the aforementioned expressions can be derived automatically and a novel algorithm to decide whether two expressions are bisimilar or not. The procedure is implemented in the automatic theorem prover CIRC, by reducing coinduction to an entailment relation between an algebraic specification and an appropriate set of equations.

1 Introduction

Regular expressions and deterministic automata (DFA's) constitute two of the most basic structures in computer science. Kleene's theorem [8] gives a fundamental correspondence between these two structures: 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 [14] presented a sound and complete axiomatization (later refined by Kozen in [9, 10]) for proving the equivalence of regular expressions.

Coalgebras arose in the last decade as a suitable mathematical framework to study state-based systems, such as DFA's. For a functor $\mathcal{G}\colon \mathbf{Set} \to \mathbf{Set}$, a

* The work of Georgiana Caltais and Eugen-Ioan Goriac has been partially supported by the PNII grant CNCSIS IDEI 393 and the project 'Meta-theory of Algebraic Process Theories' (nr. 100014021) of the Icelandic Research Fund. The work of Dorel Lucanu has been partially supported by the PNII grant CNCSIS IDEI 393.

$\mathcal{G}$-coalgebra or $\mathcal{G}$-system is a pair (S, g), consisting of a set S of states and a function $g \colon S \to \mathcal{G}(S)$ defining the "transitions" of the states. We call the functor $\mathcal{G}$ the type of the system. For instance, DFA's can be readily modeled as finite coalgebras of the functor $\mathcal{G}(S) = 2 \times S^A$.

For coalgebras of a large class of functors, a language of regular expressions; a corresponding generalization of Kleene's theorem; and a sound and complete axiomatization for the associated notion of behavioral equivalence were introduced in [2, 1]. Both the language of expressions and their axiomatization were derived, in a modular fashion, from the functor defining the type of the system.

Algebra and related tools can be successfully used for reasoning on properties of systems. In this paper, we present a novel method for checking for the bisimilarity of generalized regular expressions using the coinductive theorem prover CIRC [4, 12]. The main novelty of the method lies on the generality of the systems it can handle. CIRC is a metalanguage application implemented in Maude [3], and its target is to prove properties over infinite data structures. It has been successfully used for checking the equivalence of programs, and trace equivalence and strong bisimilarity of processes. The tool may be tested online and downloaded from `http://fsl.cs.uiuc.edu/index.php/Circ`.

The main contributions of this paper can be summarized as follows. We present the algebraic counterpart of the coalgebraic framework of the generalized regular expressions mentioned above. This enables us to automatically derive algebraic specifications that model the language of expressions, and to define an appropriate equational entailment relation for checking for the behavioural equivalence of expressions. Furthermore, the implementation of both the algebraic specification and the entailment relation in CIRC allows for automatic reasoning on the equivalence of expressions.

Organization of the paper Section 2 recalls the basic definitions of the language associated to a polynomial functor. Section 3 formulates the aforementioned language as an algebraic specification, which paves the way to implement in CIRC a procedure to decide equivalence of expressions. The decision procedure and the soundness of its implementation in CIRC are described in Section 4. In Section 4.1 we show, by means of examples, how one can check for bisimilarity, using CIRC. Section 5 contains concluding remarks and pointers for future work.

2 Regular expressions for polynomial coalgebras

In this section, we briefly recall the basic definitions in [2, 15].

Let **Set** denote the category of sets (represented by capital letters $X, Y, \ldots$) and functions (represented by lower case letters $f, g, \ldots$). The notation Y^X represents the family of functions from X to Y. The product of two sets X, Y is written as $X \times Y$ and has the projections functions π_1 and π_2: $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$. We define $X \oplus Y = X \uplus Y \uplus \{\bot, \top\}$ 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 $X \oplus Y$ is different from the classical coproduct of X and Y (which we shall denote by $X + Y$), because of the two extra elements $\bot$ and $\top$. These extra elements will later be used to

represent, respectively, underspecification and inconsistency in the specification of some systems.

For each of the operations defined above on sets, there are analogous ones on functions. Let $f\colon X \to Y$, $f_1\colon X \to Y$ and $f_2\colon Z \to W$. We define the following operations:

$f_1 \times f_2 \colon X \times Z \to Y \times W$	$f_1 \oplus f_2 \colon X \oplus Z \to Y \oplus W$
$(f_1 \times f_2)(\langle x, z\rangle) = \langle f_1(x), f_2(z)\rangle$	$(f_1 \oplus f_2)(c) = c,\ c \in \{\bot, \top\}$
$f^A \colon X^A \to Y^A$	$(f_1 \oplus f_2)(\kappa_i(x)) = \kappa_i(f_i(x)),\ i \in \{1, 2\}$
$f^A(g) = f \circ g$	

Note that here we are using the same symbols that we defined above for the operations on sets. It will always be clear from the context which operation is being used.

In our definition of non-deterministic functors we will use constant sets equipped with an information order. In particular, we will use join-semilattices. A (bounded) join-semilattice is a set B equipped with a binary operation $\vee_{\mathsf{B}}$ and a constant $\bot_{\mathsf{B}} \in \mathsf{B}$, such that $\vee_{\mathsf{B}}$ is commutative, associative and idempotent. The element $\bot_{\mathsf{B}}$ is neutral with respect to $\vee_{\mathsf{B}}$. As usual, $\vee_{\mathsf{B}}$ gives rise to a partial ordering $\leq_{\mathsf{B}}$ on the elements of B: $b_1 \leq_{\mathsf{B}} b_2 \Leftrightarrow b_1 \vee_{\mathsf{B}} b_2 = b_2$. Every set S can be mapped into a join-semilattice by taking B to be the set of all finite subsets of S with union as join.

Coalgebras A coalgebra is a pair $(S, g\colon S \to \mathcal{G}(S))$, where S is a set of states and $\mathcal{G}\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor $\mathcal{G}$, together with the function g, determines the *transition structure* (or dynamics) of the $\mathcal{G}$-coalgebra [13].

Definition 1 (Bisimulation). *Let (S, f) and (T, g) be two $\mathcal{G}$-coalgebras. We call a relation $R \subseteq S \times T$ a* bisimulation *[7] iff*

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

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

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

Polynomial functors They are functors $\mathcal{G}\colon \mathbf{Set} \to \mathbf{Set}$, built inductively from the identity, and constants, using $\times$, $\oplus$ and $(-)^A$:

$$PF \ni \mathcal{G} ::= \mathsf{Id} \mid \mathsf{B} \mid \mathcal{G} \oplus \mathcal{G} \mid \mathcal{G} \times \mathcal{G} \mid \mathcal{G}^A \tag{1}$$

where B is a (non-empty) finite join-semilattice and A is a finite set. Typical examples of polynomial functors include $\mathcal{R} = \mathsf{B} \times \mathsf{Id}$, $\mathcal{M} = (\mathsf{B} \times \mathsf{Id})^A$, $\mathcal{D} = 2 \times \mathsf{Id}^A$ and $\mathcal{Q} = (1 \oplus \mathsf{Id})^A$. These functors represent, respectively, the type of Mealy, deterministic and partial deterministic automata. $\mathcal{R}$-bisimulation is stream equality, whereas $\mathcal{D}$-bisimulation coincides with language equivalence.

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

Definition 2. *Let $\lhd \subseteq PF \times PF$ be the least reflexive and transitive relation on polynomial functors such that*

$$\mathcal{G}_1 \lhd \mathcal{G}_1 \times \mathcal{G}_2, \quad \mathcal{G}_2 \lhd \mathcal{G}_1 \times \mathcal{G}_2, \quad \mathcal{G}_1 \lhd \mathcal{G}_1 \oplus \mathcal{G}_2, \quad \mathcal{G}_2 \lhd \mathcal{G}_1 \oplus \mathcal{G}_2, \quad \mathcal{G} \lhd \mathcal{G}^A$$

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

A language of regular expressions for polynomial coalgebras We now associate a language of expressions $\mathsf{Exp}_{\mathcal{G}}$ with each polynomial functor $\mathcal{G}$.

Definition 3 (Expressions). *Let A be a finite set, B a finite join-semilattice and X a set of fixed-point variables. The set Exp of all* expressions *is given by the following grammar, where $a \in A$, $b \in \mathsf{B}$ and $x \in X$:*

$$\varepsilon ::= \underline{\emptyset} \mid x \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma \mid b \mid l\langle\varepsilon\rangle \mid r\langle\varepsilon\rangle \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \tag{2}$$

where γ is a guarded expression *given by:*

$$\gamma ::= \underline{\emptyset} \mid \gamma \oplus \gamma \mid \mu x.\gamma \mid b \mid l\langle\varepsilon\rangle \mid r\langle\varepsilon\rangle \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon) \tag{3}$$

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

The language of expressions for polynomial coalgebras is a generalization of the classical notion of regular expressions: $\emptyset$, $\varepsilon_1 \oplus \varepsilon_2$ and $\mu x.\gamma$ play similar roles to the regular expressions denoting empty language, the union of languages and the Kleene star. The expressions $l\langle\varepsilon\rangle$, $r\langle\varepsilon\rangle$, $l[\varepsilon]$, $r[\varepsilon]$ and $a(\varepsilon)$ refer to the left and right hand-side of products and coproducts, and function application, respectively. Next, we present a type assignment system for associating expressions to polynomial functors. This will allow us to associate with each functor $\mathcal{G}$ the expressions $\varepsilon \in \mathsf{Exp}^c$ that are valid specifications of $\mathcal{G}$-coalgebras.

Definition 4 (Type system). *We now define a typing relation $\vdash \subseteq \mathsf{Exp} \times PF \times PF$ that will associate an expression ε with two polynomial functors $\mathcal{F}$ and $\mathcal{G}$, which are related by the ingredient relation ($\mathcal{F}$ is an ingredient of $\mathcal{G}$). We shall write $\vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}$ for $\langle \varepsilon, \mathcal{F}, \mathcal{G} \rangle \in \vdash$. The rules that define $\vdash$ are the following:*

$$\frac{}{\vdash \underline{\emptyset} \colon \mathcal{F} \lhd \mathcal{G}} \qquad \frac{}{\vdash b \colon \mathsf{B} \lhd \mathcal{G}} \qquad \frac{}{\vdash x \colon \mathcal{G} \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{G} \lhd \mathcal{G}}{\vdash \mu x.\varepsilon \colon \mathcal{G} \lhd \mathcal{G}}$$

$$\frac{\vdash \varepsilon_1 \colon \mathcal{F} \lhd \mathcal{G} \quad \vdash \varepsilon_2 \colon \mathcal{F} \lhd \mathcal{G}}{\vdash \varepsilon_1 \oplus \varepsilon_2 \colon \mathcal{F} \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{G} \lhd \mathcal{G}}{\vdash \varepsilon \colon \mathsf{Id} \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_2 \lhd \mathcal{G}}{\vdash r[\varepsilon] \colon \mathcal{F}_1 \oplus \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}}{\vdash a(\varepsilon) \colon \mathcal{F}^A \lhd \mathcal{G}}$$

$$\frac{\vdash \varepsilon \colon \mathcal{F}_1 \lhd \mathcal{G}}{\vdash l\langle\varepsilon\rangle \colon \mathcal{F}_1 \times \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_2 \lhd \mathcal{G}}{\vdash r\langle\varepsilon\rangle \colon \mathcal{F}_1 \times \mathcal{F}_2 \lhd \mathcal{G}} \qquad \frac{\vdash \varepsilon \colon \mathcal{F}_1 \lhd \mathcal{G}}{\vdash l[\varepsilon] \colon \mathcal{F}_1 \oplus \mathcal{F}_2 \lhd \mathcal{G}}$$

We can now formally define the set of $\mathcal{G}$-expressions: well-typed expressions associated with a polynomial functor $\mathcal{G}$.

Definition 5 ($\mathcal{G}$-expressions). *Let $\mathcal{G}$ be a polynomial functor and $\mathcal{F}$ an ingredient of $\mathcal{G}$. We define $\mathsf{Exp}_{\mathcal{F}\lhd\mathcal{G}}$ by:*

$$\mathsf{Exp}_{\mathcal{F}\lhd\mathcal{G}} = \{\varepsilon \in \mathsf{Exp}^c \mid \vdash \varepsilon \colon \mathcal{F} \lhd \mathcal{G}\}\,.$$

*We define the set $\mathsf{Exp}_{\mathcal{G}}$ of well-typed $\mathcal{G}$-*expressions *by $\mathsf{Exp}_{\mathcal{G}\lhd\mathcal{G}}$.*

In [2], it was proved that the set of $\mathcal{G}$-expressions for a given polynomial functor $\mathcal{G}$ has a coalgebraic structure:

$$\delta_{\mathcal{G}} \colon \mathsf{Exp}_{\mathcal{G}} \to \mathcal{G}(\mathsf{Exp}_{\mathcal{G}})$$

More precisely, in [2, 15], which we refer to for the complete definition of $\delta_{\mathcal{G}}$, the authors defined a function $\delta_{\mathcal{F}\lhd\mathcal{G}} \colon \mathsf{Exp}_{\mathcal{F}\lhd\mathcal{G}} \to \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$ and then set $\delta_{\mathcal{G}} = \delta_{\mathcal{G}\lhd\mathcal{G}}$.

The coalgebraic structure on the set of expressions enabled the proof of a Kleene like theorem.

Theorem 1 (Kleene's theorem for polynomial coalgebras). *Let $\mathcal{G}$ be a polynomial functor.*

1. *For any $\varepsilon \in \mathsf{Exp}_{\mathcal{G}}$, there exists a finite $\mathcal{G}$-coalgebra (S, g) and $s \in S$ such that $\varepsilon \sim s$.*
2. *For every $\mathcal{G}$-coalgebra (S, g) and $s \in S$ there exists an expression $\varepsilon_s \in \mathsf{Exp}_{\mathcal{G}}$ such that $\varepsilon_s \sim s$.*

In order to provide the reader we intuition over the notions presented above, we illustrate them with an example.

Example 1. Let us instantiate the definition of $\mathcal{G}$-expressions to the functors of streams $\mathcal{R} = \mathsf{B} \times \mathsf{Id}$ (the ingredients of this functor are B, Id and $\mathcal{R}$ itself). Let X be a set of (recursion or) fixed-point variables. The set $\mathsf{Exp}_{\mathcal{R}}$ of *stream expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For $x \in X$:

$$\begin{aligned} \mathsf{Exp}_{\mathcal{R}} \ni \varepsilon &::= \underline{\emptyset} \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid l\langle \varepsilon_1 \rangle \mid r\langle \varepsilon \rangle \\ \varepsilon_1 &::= \underline{\emptyset} \mid b \mid \varepsilon_1 \oplus \varepsilon_1 \end{aligned} \tag{4}$$

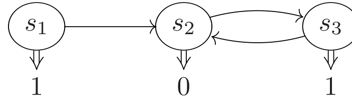
Intuitively, the expression $l\langle b \rangle$ is used to specify that the head of the stream is b, while $r\langle \varepsilon \rangle$ specifies a stream whose tail behaves as specified by ε. For the two element join-semilattice $\mathsf{B} = \{0, 1\}$ (with $\perp_{\mathsf{B}} = 0$), examples of well-typed expressions include $\underline{\emptyset}$, $l\langle 1 \rangle \oplus r\langle l\langle \underline{\emptyset} \rangle \rangle$ and $\mu x.r\langle x \rangle \oplus l\langle 1 \rangle$. The expressions $l[1]$, $l\langle 1 \rangle \oplus 1$ and $\mu x.1$ are examples of non well-typed expressions for $\mathcal{R}$, because the functor $\mathcal{R}$ does not involve $\oplus$, the subexpressions in the sum have different type, and recursion is not at the outermost level (1 has type $\mathsf{B} \lhd \mathcal{R}$), respectively.

By applying the definition in [2], the coalgebra structure on expressions $\delta_{\mathcal{R}}$ would be given by:

$$
\begin{array}{ll}
\delta_{\mathcal{R}} \colon \mathsf{Exp}_{\mathcal{R}} \to \mathsf{B} \times \mathsf{Exp}_{\mathcal{R}} & \\
\delta_{\mathcal{R}}(\underline{\emptyset}) & = \langle 0, \underline{\emptyset} \rangle \\
\delta_{\mathcal{R}}(\varepsilon_1 \oplus \varepsilon_2) & = \langle b_1 \vee b_2, \varepsilon_1' \oplus \varepsilon_2' \rangle \text{ where } \langle b_i, \varepsilon_i \rangle = \delta_{\mathcal{R}}(\varepsilon_i),\ i = 1,2 \\
\delta_{\mathcal{R}}(\mu x.\varepsilon) & = \delta_{\mathcal{R}}(\varepsilon[\mu x.\varepsilon / x]) \\
\delta_{\mathcal{R}}(l\langle \varepsilon_1 \rangle) & = \langle \delta_{\mathsf{B} \triangleleft \mathcal{R}}(\varepsilon_1), \underline{\emptyset} \rangle \\
\delta_{\mathcal{R}}(r\langle \varepsilon \rangle) & = \langle \bot_{\mathsf{B}}, \varepsilon \rangle \\
 & \\
\delta_{\mathsf{B} \triangleleft \mathcal{R}}(\underline{\emptyset}) & = \bot_B \\
\delta_{\mathsf{B} \triangleleft \mathcal{R}}(b) & = b \\
\delta_{\mathsf{B} \triangleleft \mathcal{R}}(\varepsilon_1 \oplus \varepsilon_1') & = \delta_{\mathsf{B} \triangleleft \mathcal{R}}(\varepsilon_1) \vee \delta_{\mathsf{B} \triangleleft \mathcal{R}}(\varepsilon_1')
\end{array}
$$

The proof of Kleene's theorem provides algorithms to go from expressions to streams and vice-versa. We illustrate it by means of examples.

Consider the following stream:



We draw the stream with an automata-like flavor. The transitions indicate the tail of the stream represented by a state and the output value the head. In a more traditional notation, the above automata represents the infinite stream $(1, 0, 1, 0, 1, 0, 1, \ldots)$.

To compute expressions ε_1, ε_2 and ε_3 equivalent to s_1, s_2 and s_3 we associate with each state s_i a variable x_i and we solve the following system of 3 equations in 3 variables:

$$\varepsilon_1 = \mu x_1.l\langle 1 \rangle \oplus r\langle x_2 \rangle \quad \varepsilon_2 = \mu x_2.l\langle 0 \rangle \oplus r\langle x_3 \rangle \quad \varepsilon_3 = \mu x_3.l\langle 1 \rangle \oplus r\langle x_2 \rangle$$

which yields the following closed expressions:

$$\varepsilon_1 = \mu x_1.l\langle 1 \rangle \oplus r\langle \varepsilon_2 \rangle \quad \varepsilon_2 = \mu x_2.l\langle 0 \rangle \oplus r\langle \varepsilon_3 \rangle \quad \varepsilon_3 = \mu x_3.l\langle 1 \rangle \oplus r\langle \mu x_2.l\langle 0 \rangle \oplus r\langle x_3 \rangle \rangle$$

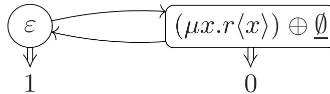
satisfying, by construction, $\varepsilon_1 \sim s_1$, $\varepsilon_2 \sim s_2$ and $\varepsilon_3 \sim s_3$.

For the converse construction, consider the expression $\varepsilon = (\mu x.r\langle x \rangle) \oplus l\langle 1 \rangle$. We construct an automaton by repeatedly applying the coalgebra structure on expressions $\delta_{\mathcal{R}}$, modulo ACI (associativity, commutativity and idempotency of $\oplus$) in order to guarantee finiteness.

Applying the definition of $\delta_{\mathcal{R}}$ above, we have:

$$\delta_{\mathcal{R}}(\varepsilon) = \langle 1, (\mu x.r\langle x \rangle) \oplus \underline{\emptyset} \rangle \text{ and } \delta_{\mathcal{R}}((\mu x.r\langle x \rangle) \oplus \underline{\emptyset}) = \langle 0, (\mu x.r\langle x \rangle) \oplus \underline{\emptyset} \rangle$$

which leads to the following stream (automaton):



Note that, throughout the paper, we will use streams as a basic example to illustrate the definitions. It should be remarked that the framework is general enough to include more complex examples, such as deterministic automata, automata on guarded strings or Mealy machines. The latter will be used as example in Section 4.1.

3 An algebraic approach on the coalgebra of generalized regular expressions

We now have a (theoretical) framework which, given a functor $\mathcal{G}$, allows for the uniform derivation of 1) a language $\mathsf{Exp}_{\mathcal{G}}$ for specifying behaviors of $\mathcal{G}$-systems, and 2) a coalgebraic structure on $\mathsf{Exp}_{\mathcal{G}}$, which provides an operational semantics to the set of expressions. In the rest of the paper, we will extend and adapt the framework of the previous section in order to:

- enable the implementation of a tool which allows for the automatic derivation of 1) and 2) above
- enable automatic reasoning on equivalence of specifications; the reasoning will be performed by the coinductive prover CIRC [12], which is also the core of our target tool.

CIRC is based on algebraic specifications and, therefore, to reach our final goal we need two things:

- *algebraic specifications* that model both the language and the coalgebraic structure of expressions associated to polynomial functors to provide to CIRC
- a decision procedure, implemented in CIRC based on an *equational entailment relation*, in order to check for the bisimilarity of expressions.

We further give the basic notions the reader needs in order to get an easier understanding of the algebraic approach. An *algebraic specification* is a triple $\mathcal{E} = (S, \Sigma, E)$, where S is a set of *sorts*, Σ is a *many-sorted signature* and E is a set of *conditional equations* of the form $(\forall X)\, t = t'$ `if` $(\bigwedge_{i \in I} u_i = v_i)$, where t, t', u_i, and v_i ($i \in I$ – a set of indexes for the conditions) are Σ-terms with variables in X. We say that the *sort of the equation* is s whenever $t, t' \in \mathcal{T}_{\Sigma,s}(X)$. Here, $\mathcal{T}_{\Sigma,s}(X)$ denotes the set of terms of sort s of the Σ-algebra freely generated by X. If $I = \{\}$ then the equation is *unconditional* and may be written as $(\forall X)\, t = t'$.

Let $\vdash$ be the *equational entailment (deduction) relation* defined as in [5]. We write $\mathcal{E} \vdash e$ whenever equation e is deducible from $\mathcal{E}$. We extend $\mathcal{E}$ by adding the freezing operation $\boxed{-}{:}s \to \mathsf{Frozen}$ for each sort $s \in \Sigma$, where Frozen is a fresh sort. By $\boxed{t}$ we represent the *frozen* form of a Σ-term t, and by $\boxed{e}$ a *frozen equation* of the shape $(\forall X)\boxed{t} = \boxed{t'}$ if c. The entailment relation $\vdash$ is defined over frozen equations as in [12]. The need for the frozen operator will become clear in Example 2: without it the congruence rule could be applied freely leading to the derivation of untrue equations.

Fig. 1 briefly illustrates the parallel between the coalgebraic concepts presented in [15, 2] and their algebraic correspondents. In what follows, we will

coalgebraic	algebraic
$\vdash \varepsilon \colon \mathcal{F} \triangleleft \mathcal{G}$	$\mathcal{E}_{\mathcal{G}} \vdash \varepsilon \colon \mathcal{F} \triangleleft \mathcal{G} = true$
$\mathsf{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$	$\{\varepsilon \in \mathcal{T}_{\Sigma,\mathsf{Exp}} \mid \mathcal{E}_{\mathcal{G}} \vdash \varepsilon \colon \mathcal{F} \triangleleft \mathcal{G} = true\}$
$\mathsf{Exp}_{\mathcal{G}}$	$\{\varepsilon \in \mathcal{T}_{\Sigma,\mathsf{Exp}} \mid \mathcal{E}_{\mathcal{G}} \vdash \varepsilon \colon \mathcal{G} \triangleleft \mathcal{G} = true\}$
$\mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$	$\{\sigma \in \mathcal{T}_{\Sigma,\mathsf{ExpStruct}} \mid \mathcal{E}_{\mathcal{G}} \vdash \sigma \colon \mathcal{F}(\mathsf{Exp}\,\mathcal{G}) = true\}$
$\delta_{\mathcal{F} \triangleleft \mathcal{G}} \colon \mathsf{Exp}_{\mathcal{F} \triangleleft \mathcal{G}} \to \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$	$\delta_{_}(_) \colon \mathsf{Ingredient\ Exp} \to \mathsf{ExpStruct}$
$\langle \sigma, \sigma' \rangle \in \overline{\mathcal{F}}(cl(\mathcal{R}_{id}))$	$\mathcal{E}_{\mathcal{G}} \vdash \sigma \colon \mathcal{F}(\mathsf{Exp}\,\mathcal{G}) = true,\ \mathcal{E}_{\mathcal{G}} \vdash \sigma' \colon \mathcal{F}(\mathsf{Exp}\,\mathcal{G}) = true$ $\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\sigma} = \boxed{\sigma'}$ (i)
$cl(\mathcal{R}_{id})$ is a bisimulation	$\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{R})}$ (ii)

Fig. 1. Polynomial functors - coalgebraic vs. algebraic approach

provide some explanations on the algebraic side, in order to model what we presented coalgebraically in the previous section, analyzing the components of Fig. 1.

The algebraic specification of a polynomial functor. For the provided functor $\mathcal{G}$, the specification $\mathcal{E}_{\mathcal{G}} = (S, \Sigma, E)$ is incrementally built according to the items common to all regular expressions, extended with the items specific to $\mathcal{G}$ (*e.g.*, the semilattices, the exponentiation alphabets). As an initial step in the construction of $\mathcal{E}_{\mathcal{G}}$, we use the general rule for translating definitions based on Backus-Naur grammars into algebraic specifications. Each syntactical category and vocabulary is considered as a sort, and each production is considered as a constructor operation or a subsort relation. For instance, according to the grammar of generalized regular expressions in Definition 3, we have: a sort Exp representing expressions ε, $\mathsf{FixpVar}$ the sort for the vocabulary of the fixed-point variables, Alph the sort for the elements of the alphabets, and Slt the sort for the elements of the semilattices. Moreover, we consider constructor operations for all the productions. For example, the production $\varepsilon ::= \varepsilon \oplus \varepsilon$ is represented by an operation $_ \oplus _ \colon \mathsf{Exp}\ \ \mathsf{Exp} \to \mathsf{Exp}$. Using a similar mechanism, we specify:

- structured expressions σ, the counterpart of $\mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$, defined by
$$\sigma ::= \ \varepsilon \mid \langle \sigma, \sigma \rangle \mid k_1(\sigma) \mid k_2(\sigma) \mid \bot \mid \top \mid \lambda x.(a, \mathcal{F} \triangleleft \mathcal{G}, \sigma)$$
we denote the sort of this kind of expressions by $\mathsf{ExpStruct}$ (the construction $\lambda x.(a, \mathcal{F} \triangleleft \mathcal{G}, \sigma)$ has as coalgebraic correspondent a function $f \in \mathcal{F}^A(\mathsf{Exp}_{\mathcal{G}})$)
- polynomial functors defined by grammar (1); the associated sort is $\mathsf{Functor}$
- functor ingredients given in Definition 2; the corresponding sort is $\mathsf{Ingredient}$

The set $\mathsf{Exp}_{\mathcal{F} \triangleleft \mathcal{G}}$ of expressions of type $\mathcal{F} \triangleleft \mathcal{G}$ is algebraically represented by the set of Σ-terms ε of sort Exp, such that $\mathcal{E}_{\mathcal{G}} \vdash \varepsilon \colon \mathcal{F} \triangleleft \mathcal{G} = true$. The type-checking relation in Definition 4 is given by an operation $_ : _ \colon \mathsf{Exp\ Ingredient} \to \mathsf{Bool}$ and an equation for each inference rule defining this relation. For example

$$\frac{\vdash \varepsilon_1 \colon \mathcal{F} \triangleleft \mathcal{G} \quad \vdash \varepsilon_2 \colon \mathcal{F} \triangleleft \mathcal{G}}{\vdash \varepsilon_1 \oplus \varepsilon_2 \colon \mathcal{F} \triangleleft \mathcal{G}}$$

is represented by the equation $\varepsilon_1 \oplus \varepsilon_2 \colon \mathcal{F} \triangleleft \mathcal{G} = \varepsilon_1 \colon \mathcal{F} \triangleleft \mathcal{G} \wedge \varepsilon_2 \colon \mathcal{F} \triangleleft \mathcal{G}$. For the sake of notation, algebraically we write $\varepsilon \colon \mathcal{F} \triangleleft \mathcal{G}$ to represent expressions of type $\mathcal{F} \triangleleft \mathcal{G}$.

The structured expressions $\sigma \in \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$ are given by the set of Σ-terms of sort $\mathsf{ExpStruct}$, such that $\mathcal{E}_{\mathcal{G}} \vdash \sigma \colon \mathcal{F}(\mathsf{Exp}\,\mathcal{G}) = \mathit{true}$ (here : is the extension of the type-checking operator to structured expressions). Algebraically, we write $\sigma \colon \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$ to denote that σ is an element of $\mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$.

The function $\delta_{\mathcal{G}}$, which provides the coalgebraic structure of $\mathcal{G}$-expressions, has the algebraic correspondent $\delta \in \Sigma$, a function parameterized with the functor ingredients.

Recall from Section 2 that a relation $\mathcal{R} \subseteq \mathsf{Exp}_{\mathcal{G} \triangleleft \mathcal{G}} \times \mathsf{Exp}_{\mathcal{G} \triangleleft \mathcal{G}}$ is a bisimulation if and only if $(s,t) \in \mathcal{R} \Rightarrow \langle \delta_{\mathcal{G} \triangleleft \mathcal{G}}(s), \delta_{\mathcal{G} \triangleleft \mathcal{G}}(t) \rangle \in \overline{\mathcal{G}}(R)$. In order to enable the algebraic framework to decide bisimilarity of $\mathcal{G}$-expressions, we define a new entailment relation for polynomial functors $\vdash_{PF}$ (the definitions of $\overline{\mathcal{G}}$ and $\vdash_{PF}$ are closely related).

Definition 6. *The entailment relation $\vdash_{PF}$ is the extension of $\vdash$ with the following inference rules, which allow a restricted contextual reasoning over the frozen equations of structured expressions:*

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{\sigma_1} = \boxed{\sigma_1'} \quad \mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{\sigma_2} = \boxed{\sigma_2'}}{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{\langle \sigma_1, \sigma_2 \rangle} = \boxed{\langle \sigma_1', \sigma_2' \rangle}} \tag{5}$$

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{\sigma} = \boxed{\sigma'}}{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{k_i(\sigma)} = \boxed{k_i(\sigma')}\ (i = \overline{1,2})} \tag{6}$$

$$\frac{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{f(a)} = \boxed{g(a)},\ \text{for all } a \in A}{\mathcal{E}_{\mathcal{G}} \vdash_{PF} \boxed{f} = \boxed{g}} \tag{7}$$

Let $\mathcal{G}$ be a polynomial functor, and $\mathcal{R}$ a binary relation on the set of $\mathcal{G}$-expressions. We will make use of the conventions:

- $\mathcal{R}_{id} = \mathcal{R} \cup \{(\varepsilon, \varepsilon) \mid \mathcal{E}_{\mathcal{G}} \vdash \varepsilon \colon \mathcal{G} \triangleleft \mathcal{G} = \mathit{true}\}$
- $cl(\mathcal{R})$ is the closure of $\mathcal{R}$ under transitivity, symmetry and reflexivity
- $\boxed{\mathcal{R}} = \bigcup_{e \in \mathcal{R}} \{\boxed{e}\}$ (application of the freezing operator to all elements of $\mathcal{R}$)
- $\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}}$ is a shorthand for $(S, \Sigma, E \cup \{\boxed{\varepsilon} = \boxed{\varepsilon'} \mid (\varepsilon, \varepsilon') \in \mathcal{R}\})$
- $\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon = \varepsilon')$ denotes the equation $\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon) = \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon')$
- $\langle \sigma, \sigma' \rangle \in \overline{\mathcal{G}}(\mathcal{R})$ is a shorthand for: (σ, σ') is an element of the set S, where $\mathcal{E}_{\mathcal{G}} \vdash \overline{\mathcal{G}}(\mathcal{R}) = S$ (here, $\overline{\mathcal{G}}(\mathcal{R}) \subseteq \mathcal{T}_{\Sigma,\mathsf{ExpStruct}} \times \mathcal{T}_{\Sigma,\mathsf{ExpStruct}}$)

The following theorem and corollary correspond to the equivalences (i), and respectively (ii), in Fig. 1. Theorem 2 formalizes the connection between the inductive definition of $\overline{\mathcal{G}}$ (on the coalgebraic side) and $\vdash_{PF}$ (on the algebraic side), hence enabling the definition of bisimulations in algebraic terms, in Corollary 1.

Theorem 2. *Consider a polynomial functor $\mathcal{G}$ and $\mathcal{F}$ an ingredient of $\mathcal{G}$. If $\mathcal{R}$ is a binary relation on the set of $\mathcal{G}$-expressions, and $\sigma, \sigma' \colon \mathcal{F}(\mathsf{Exp}_{\mathcal{G}})$ then $\langle \sigma, \sigma' \rangle \in \overline{\mathcal{F}}(cl(\mathcal{R}_{id}))$ iff $\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\sigma} = \boxed{\sigma'}$.*

Proof. The proof is by induction on the structure of $\mathcal{F}$. Take, for example the direct implication " $\Rightarrow$ ". The base case $\mathcal{F} = \mathsf{B}$ holds by the reflexivity of $\vdash_{PF}$. The case $\mathcal{F} = \mathsf{Id}$ follows immediately according to an auxiliary result stating that if $(\varepsilon, \varepsilon') \in cl(\mathcal{R}_{id})$ then $\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\varepsilon} = \boxed{\varepsilon'}$. Inductive steps hold by the rules (5), (6) and (7), defining $\vdash_{PF}$. A similar reasoning is used for proving " $\Leftarrow$ ". □

Corollary 1. *Let $\mathcal{G}$ be a polynomial functor. If $\mathcal{R}$ is a binary relation on the set of $\mathcal{G}$-expressions, then $cl(\mathcal{R}_{id})$ is a bisimulation iff $\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{R})}$.*

Proof. The result follows immediately according to the equivalences:
$cl(\mathcal{R}_{id})$ is a bisimulation $\Leftrightarrow_{\text{(Definition 1)}} (\forall(\varepsilon, \varepsilon') \in cl(\mathcal{R}_{id})).\langle \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon), \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\varepsilon') \rangle \in \overline{\mathcal{G}}(cl(\mathcal{R}_{id})) \Leftrightarrow_{\text{(Theorem 2)}} \mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(cl(\mathcal{R}_{id}))} \Leftrightarrow_{(\text{def.} cl(\mathcal{R}_{id}), \vdash_{PF})} \mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{R}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{R})}$. □

4 A decision procedure for bisimilarity

In this section we describe how the coinductive theorem prover CIRC [11] can be used to implement a decision procedure for the bisimilarity of generalized regular expressions.

CIRC can be seen as an extension of Maude with behavioral features and its implementation is derived from that of Full-Maude. In order to use the prover, one needs to provide a specification (a CIRC theory) and a set of goals. A CIRC theory $\mathcal{B} = (S, (\Sigma, \Delta), (E, \mathcal{I}))$ consists of an algebraic specification (S, Σ, E), a set Δ of *derivatives* (= Σ-contexts), and a set $\mathcal{I}$ of equational interpolants, which are expressions of the form $e \Rightarrow \{e_i \mid i \in I\}$ where e and e_i are equations (for more information on equational interpolants see [6]). A derivative $\delta \in \Delta$ is a Σ-term containing a special variable $*{:}s$, where s is the sort of the variable $*$. If e is an equation $t = t'$ with t and t' of sort s, then $\delta[e]$ is $\delta[t/*{:}s] = \delta[t'/*{:}s]$. Let $\Delta[e]$ denote the set $\{\delta[e] \mid \delta \in \Delta \text{ appropriate for } e\}$.

CIRC implements the coinductive proof system given in [12] using a set of reduction rules of the form $(\mathcal{B}, \mathcal{F}, \mathcal{G}) \Rightarrow (\mathcal{B}, \mathcal{F}', \mathcal{G}')$, where $\mathcal{B}$ represents a specification, $\mathcal{F}$ is the coinductive hypothesis (a set of frozen equations) and $\mathcal{G}$ is the current set of goals. The freezing operator is defined as described in Section 3. Here is a brief description of these rules:

[Done]: $(\mathcal{B}, \mathcal{F}, \{\}) \Rightarrow \cdot$
Whenever the set of goals is empty, the system terminates with success.

[Reduce]: $(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow (\mathcal{B}, \mathcal{F}, \mathcal{G})$ `if` $\mathcal{B} \cup \mathcal{F} \vdash \boxed{e}$
If the current goal is a $\vdash$-consequence of $\mathcal{B} \cup \mathcal{F}$ then $\boxed{e}$ is removed from the set of goals.

[Derive]: $(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow (\mathcal{B}, \mathcal{F} \cup \{\boxed{e}\}, \mathcal{G} \cup \boxed{\Delta[e]})$ `if` $\mathcal{B} \cup \mathcal{F} \not\vdash \boxed{e}$
When the current goal e has the same sort with the special variable $*$, and it is not a $\vdash$-consequence, it is added to the specification and its derivatives to the set of goals. In order to simplify the notation, we write $\delta(e)$ for $\delta(\varepsilon) = \delta(\varepsilon')$, whenever e is of shape $\varepsilon = \varepsilon'$.

[Simplify]: $(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{\theta(e)}\}) \Rightarrow (\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{\theta(e_i)} \mid i \in I\})$
`if` $e \Rightarrow \{e_i \mid i \in I\}$ is a simplification rule from the specification and $\theta \colon X \to \mathcal{T}_{\Sigma}(Y)$ is a substitution.

[Fail]: $(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow$ *failure* `if` $\mathcal{B} \cup \mathcal{F} \not\vdash \boxed{e} \wedge e{:}\mathsf{Bool}$
This rule stops the reduction process with failure whenever the current goal e is of type Bool and the corresponding normal forms are different.

It is worth noting that there is a strong connection between a CIRC proof and the construction of a bisimulation relation. We emphasize this fact and the importance of the freezing operator with a simple example.

Example 2. Consider the case of infinite streams. The set B^{ω} of infinite streams over a set B is the final coalgebra of the functor $\mathcal{R} = \mathsf{B} \times \mathsf{Id}$, with a coalgebra structure given by hd and tl, the functions that return the head and the tail of the stream, respectively. Our purpose is to prove that $0^{\infty} = (00)^{\infty}$. Let z and zz represent the stream on the left hand side and, respectively, on the right hand side. These streams are defined by the equations: $hd(z) = 0, tl(z) = z, hd(zz) = 0, tl(zz) = 0{:}zz$. In Fig. 2 we present the correlation between the CIRC proof and the construction of the bisimulation relation. Note how CIRC collects the elements of the bisimulation as frozen hypothesis.

CIRC proof	Bisimulation construction
`(add goal z = zz .)`	z zz $(zz)'$ 0 0 0
$(\mathcal{B}, \{\}, \{\boxed{z} = \boxed{zz}\})$	$\mathcal{F} = \{\}$; $z \sim zz$?
$\xrightarrow{[Derive]} \left(\mathcal{B}, \{\boxed{z} = \boxed{zz}\}, \left\{\begin{array}{c}\boxed{hd(z)}=\boxed{hd(zz)}\\ \boxed{tl(z)}=\boxed{tl(zz)}\end{array}\right\}\right)$	$\mathcal{F} = \{(z, zz)\}$; $z \xrightarrow{0} z$, $zz \xrightarrow{0} (zz)'$
$\xrightarrow{[Reduce]} (\mathcal{B}, \{\boxed{z} = \boxed{zz}\}, \{\boxed{z} = \boxed{0{:}zz}\})$	$\mathcal{F} = \{(z, zz)\}$; $z \sim (zz)'$?
$\xrightarrow{[Derive]} \left(\mathcal{B}, \left\{\begin{array}{c}\boxed{z}=\boxed{zz}\\ \boxed{z}=\boxed{0{:}zz}\end{array}\right\}, \left\{\begin{array}{c}\boxed{hd(z)}=\boxed{hd(0{:}zz)}\\ \boxed{tl(z)}=\boxed{tl(0{:}zz)}\end{array}\right\}\right)$	$\mathcal{F} = \{(z, zz), (z, (zz)')\}$; $z \xrightarrow{0} z$, $(zz)' \xrightarrow{0} zz$
$\xrightarrow{[Reduce]} \left(\mathcal{B}, \left\{\begin{array}{c}\boxed{z}=\boxed{zz}\\ \boxed{z}=\boxed{0{:}zz}\end{array}\right\}, \{\}\right)$	$\mathcal{F} = \{(z, zz), (z, (zz)')\}$ ✓

Fig. 2. Parallel between a CIRC proof and the bisimulation construction

Let us analyze what happens if the freezing operator $\boxed{-}$ would not be used. Suppose the circular coinduction algorithm would add the equation $z = zz$ in its unfrozen form to the hypothesis. After applying the derivatives we obtain the goals $hd(z) = hd(zz), tl(z) = tl(zz)$. At this point, the prover could use the freshly added equation, and according to the congruence rule, both goals would be proven directly, though we would still be in the process of showing that the

hypothesis holds. By following a similar reasoning, we could then also prove that $0^\infty = 1^\infty$! In order to avoid these situations, the hypotheses are frozen (*i.e.*, their sort is changed from Stream to Frozen) and this stops the application of the congruence rule, forcing the application of the derivatives according to their definition in the specification. Therefore, the use of the freezing operator is vital for the soundness of circular coinduction.

Next, we focus on using CIRC for automatically reasoning on the equivalence of $\mathcal{G}$-expressions. As we will show, the implementation of both the algebraic specifications associated to polynomial functors and the equational entailment relation described in Section 3, is immediate. Given a polynomial functor $\mathcal{G}$, we define a CIRC theory $\mathcal{B}_\mathcal{G} = (S, (\Sigma, \Delta), (E, \mathcal{I}))$ as follows:

- (S, Σ, E) is $\mathcal{E}_\mathcal{G}$
- $\Delta = \{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(*{:}\mathsf{Exp})\}$
- $\mathcal{I}$ consists of the following equational interpolants:

$$\{\langle \sigma_1, \sigma_2 \rangle = \langle \sigma_1', \sigma_2' \rangle\} \Rightarrow \{\sigma_1 = \sigma_1',\ \sigma_2 = \sigma_2'\} \tag{8}$$

$$\{k_i(\sigma) = k_i(\sigma')\} \Rightarrow \{\sigma = \sigma'\} \tag{9}$$

$$\{f = g\} \Rightarrow \{f(a) = g(a) \mid a \in A\} \tag{10}$$

The interpolants (8), (9) and (10) in $\mathcal{I}$ extend the entailment relation $\vdash$ from the system above to $\vdash_{PF}$ (see Definition 6) as follows:

$$\frac{E \vdash e}{E \vdash_{PF} e} \qquad\qquad \frac{E \vdash_{PF} \{e_i \mid i \in I\}}{E \vdash_{PF} e} \text{ if } e \Rightarrow \{e_i \mid i \in I\} \text{ in } \mathcal{I}$$

Theorem 3 (Soundness). *Let $\mathcal{G}$ be a polynomial functor, and $\mathcal{G}$ a binary relation on the set of $\mathcal{G}$-expressions. If $(\mathcal{B}_\mathcal{G}, \mathcal{F}_0 = \{\}, \mathcal{G}_0 = \boxed{\mathcal{G}}) \overset{*}{\Rightarrow} (\mathcal{B}_\mathcal{G}, \mathcal{F}_n, \mathcal{G}_n = \{\})$ using [Reduce], [Derive] and [Simplify], then $\mathcal{G} \subseteq \sim_\mathcal{G}$.*

Proof. The idea of the proof is to identify a bisimulation relation $\widetilde{\mathcal{F}}$ s.t. $\mathcal{G} \subseteq \widetilde{\mathcal{F}}$. On a closer look, based on the reduction rules implemented in CIRC, it is quite easy to see that the initial set of goals $\mathcal{G}$ is a $\vdash_{PF}$-consequence of $\mathcal{B}_\mathcal{G} \cup \boxed{\mathcal{F}}$, where $\mathcal{F}$ is the set of hypothesis (or derived goals) collected during a proof session. In other words, $\mathcal{G} \subseteq cl(\mathcal{F}_{id})$. So, if we anticipate a bit, we should show that $\widetilde{\mathcal{F}} = cl(\mathcal{F}_{id})$ is a bisimulation, *i.e.*, according to Corollary 1, $\mathcal{B}_\mathcal{G} \cup \boxed{\mathcal{F}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{F})}$. This is achieved by proving that $\mathcal{B}_\mathcal{G} \cup \boxed{\mathcal{F}} \vdash_{PF} \mathcal{G}_i (i = \overline{0..n})$ (note that $\boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{F})} \subseteq \bigcup_{i=\overline{0..n}} \mathcal{G}_i$, according to [Derive]). The demonstration is by induction on j, where $n - j$ is the current proof step, and by case analysis on the CIRC reduction rules applied at each step. □

Remark 1. The soundness of the proof system we describe in this paper does not follow directly from Theorem 3 in [12]. This is due to the fact that we do not have an experiment-based definition of bisimilarity. So, even though the mechanism we use for proving $\mathcal{B}_\mathcal{G} \cup \boxed{\mathcal{F}} \vdash_{PF} \boxed{\delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mathcal{F})}$ is similar to the one described in [12], the current soundness proof is conceived in terms of bisimulations (and not experiments).

Remark 2. The entailment relation $\vdash_{PF}$ CIRC uses for checking for the equivalence of generalized regular expressions is an instantiation of the parametric entailment relation $\vdash$ from the proof system in [12]. This approach extends CIRC to automatically reason on a large class of systems that can be modeled as coalgebras of polynomial functors.

As already stated, our final purpose is to use CIRC as a decision procedure for the bisimilarity of generalized regular expressions. That is, whenever provided a set of expressions, the prover stops with an yes/no answer w.r.t. their equivalence. In this context, an important aspect is that the sub-coalgebra generated by an expression $\varepsilon \in \mathsf{Exp}_{\mathcal{G}}$ by repeatedly applying $\delta_{\mathcal{G} \triangleleft \mathcal{G}}$ is, in general, infinite. Take for example the polynomial functor $\mathcal{G} = \mathsf{B} \times \mathsf{Id}$ associated to infinite streams, and consider the property $\mu x.\underline{\emptyset} \oplus r\langle x \rangle = \mu x.r\langle x \rangle$. In order to prove this, CIRC builds an infinite proof sequence by repeatedly applying $\delta_{\mathcal{G} \triangleleft \mathcal{G}}$ as follows:

$$\begin{aligned} \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mu x.\underline{\emptyset} \oplus r\langle x \rangle) &= \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mu x.r\langle x \rangle) \\ &\downarrow \\ \langle 0, \underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x \rangle)\rangle &= \langle 0, \mu x.r\langle x \rangle\rangle \\ \\ \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x \rangle)) &= \delta_{\mathcal{G} \triangleleft \mathcal{G}}(\mu x.r\langle x \rangle) \\ &\downarrow \\ \langle 0, \underline{\emptyset} \oplus \underline{\emptyset} \oplus (\mu x.\underline{\emptyset} \oplus r\langle x \rangle)\rangle &= \langle 0, \mu x.r\langle x \rangle\rangle \ [\ldots] \end{aligned}$$

In this case, the prover would never stop. It is shown in [2, 15] that the axioms for associativity, commutativity and idempotency (ACI) guarantee finiteness of the generated sub-coalgebra (note that these axioms have also been proven sound w.r.t. bisimulation). ACI properties can easily be specified in CIRC as the prover is an extension of Maude, which has a powerful matching modulo ACUI capability. The idempotency is given by the equation $\varepsilon \oplus \varepsilon = \varepsilon$, and the commutativity and associativity are specified as attributes of $\oplus$.

Theorem 4. *Let $\mathcal{G}$ be a set of proof obligations over generalized regular expressions. CIRC can be used as a decision procedure for the equivalences in $\mathcal{G}$, that is, it can assert whenever a goal $(\varepsilon_1, \varepsilon_2) \in \mathcal{G}$ is a true or false equality.*

Proof. The result is a consequence of the fact that by implementing the ACI axioms in CIRC, the set of new goals obtained by repeatedly applying the derivative δ is finite. In these circumstances, whenever CIRC stops according to the reduction rule [Done], the initial proof obligations are bisimilar. On the other hand, whenever it terminates with [Fail], the goals are not bisimilar. □

4.1 A CIRC-based tool

We have implemented a tool that, when provided with a functor $\mathcal{G}$, automatically generates a specification for CIRC which can then be used in order to automatically check whether two $\mathcal{G}$-expressions are bisimilar. The tool is implemented as a metalanguage application in Maude. It can be downloaded from `http://circidei.info.uaic.ro/functorizer/functorizer.maude`.

Let us now show another example: Mealy machines, which are coalgebras for the functor $(\mathsf{B} \times \mathsf{Id})^A$. In what follows we show how CIRC can be used in conjunction with our tool in order to act as a decision procedure when checking for the equivalence of two expressions.

Formally, a Mealy machine is a pair (S, α) consisting of a set S of states and a transition function $\alpha\colon S \to (\mathsf{B} \times S)^A$, which for each state $s \in S$ and input $a \in A$ associates an output value b and a next state s'. Typically, we write $\alpha(s)(a) = \langle b, s' \rangle \Leftrightarrow s \xrightarrow{a|b} s'$. As an example, consider the Mealy machine depicted in Fig. 3, where all the states are bisimilar.

We first show how to check for the equivalence of two expressions characterizing the states s_1 and s_2 from the Mealy machine in Fig. 3. These expressions, which could be computed, using the algorithm in Kleene's theorem, are $\varepsilon_1 = a(r\langle \mu x.a(r\langle x \rangle) \oplus b(\emptyset) \rangle) \oplus b(r\langle \mu y.a(r\langle y \rangle) \oplus b(r\langle y \rangle) \rangle)$ and $\varepsilon_2 = \mu x.a(r\langle x \rangle) \oplus b(r\langle x \rangle)$, respectively.

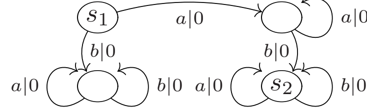


Fig. 3. Mealy machine: $s_1 \sim s_2$

In order to check for the bisimilarity of ε_1 and ε_2 we load the tool and define the semilattice $\mathsf{B} = \{0\}$ and the alphabet $A = \{a, b\}$:

```
(jslt B is 0 bottom 0 . 0 v 0 = 0 . endjslt)
(alph A is a b endalph)
```

We provide the functor $\mathcal{G}$ using the command `(functor (B x Id)^A .)`. The command `(set goal ... .)` specifies the goal we want to prove:

```
(set goal a(r< μ X:FixpVar . a(r< X:FixpVar >) (+) b(∅)>) (+)
          b(r< μ Y:FixpVar . a(r< Y:FixpVar >) (+) b(r< Y:FixpVar >) >) =
          μ X:FixpVar . a(r< X:FixpVar >) (+) b(r< X:FixpVar >) .)
```

In order to generate the CIRC specification we use the command `(generate coalgebra .)`. Next we need to load CIRC along with the resulting specification and start the proving engine using the command `(coinduction .)`.

As already shown, behind the scenes, CIRC builds a bisimulation relation that includes the initial goal. The proof succeeds and the output consists of (a subset of) this bisimulation:

```
Proof succeeded.
 Number of derived goals: 3
 Number of proving steps performed: 82
 [...]
Proved properties:
 [...]
 a(r< μ X . a(r< X >) (+) b(∅) >) (+)
 b(r< μ Y . a(r< Y >) (+) b(r< Y >) >)) =
 μ X . a(r< X >) (+) b(r< X >)
```

As previously mentioned, CIRC is also able to detect when two expressions are not equivalent. Take, for instance, the expressions $\mu x.a(r\langle a(l\langle 1 \rangle) \oplus x \rangle)$ and $a(r\langle a(l\langle 1 \rangle) \rangle) \oplus \mu x.a(r\langle x \rangle)$, characterizing the states s_1 and s_3 from the Mealy

machines in Fig. 4. After following some steps similar to the ones previously enumerated, the proof fails and the output message is `Visible goal [...] failed during coinduction`.

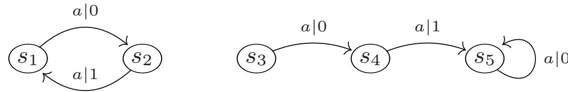


Fig. 4. Mealy machines: $s_1 \not\sim s_3$

5 Conclusions and future work

One of the major contributions of this paper is that we exploited an encoding of coalgebra into algebra, and provided a decision procedure for the bisimilarity of generalized regular expressions. In order to enable the implementation of the decision procedure, we formalized the equivalence between the coalgebraic concepts associated to polynomial coalgebras [2, 1] and their algebraic correspondents. This led to the definition of algebraic specifications ($\mathcal{E}_{\mathcal{G}}$) that model both the language and the coalgebraic structure of expressions. Moreover, we defined an equational deduction relation ($\vdash_{PF}$), used on the algebraic side for reasoning on the bisimilarity of expressions.

The most important result of the parallel between the coalgebraic and algebraic approaches is given in Corollary 1, which formalizes the definition of the bisimulation relations, in algebraic terms. Actually, this result is the key for proving the soundness of the decision procedure implemented in the automated prover CIRC [11]. As a coinductive prover, CIRC builds a relation $\mathcal{F}$ closed under the application of $\delta_{\mathcal{G}}$ with respect to $\vdash_{PF}$ ($\mathcal{E}_{\mathcal{G}} \cup \boxed{\mathcal{F}} \vdash_{PF} \boxed{\delta_{\mathcal{G}}(\mathcal{F})}$), hence automatically computing a bisimulation the initial proof obligations belong to.

The approach we present in this paper enables CIRC to perform a reasoning based on bisimulations (instead of experiments [12]). This way, the prover is extended to checking for the bisimilarity in a large class of systems that can be modeled as $\mathcal{G}$-coalgebras. Note that the constructions above are all automated – the (non-trivial) CIRC algebraic specification describing $\mathcal{E}_{\mathcal{G}}$, together with the interpolants implementing $\vdash_{PF}$ are generated with the Maude tool presented in Section 4.1.

As future work, we intend to extend our proof system to Kripke polynomial coalgebras and to exploit more of the axioms in [1] with the purpose of increasing the prover time performance (our experience so far shows that by adding the axiom for the distribution of the $\underline{\emptyset}$ expression through the constructors, the prover works significantly faster).

Acknowledgments. The authors are grateful for useful comments from Filippo Bonchi and the anonymous reviewers.

References

1. M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. An algebra for Kripke polynomial coalgebras. In *LICS*, pages 49–58. IEEE Computer Society, 2009.

2. M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. A Kleene theorem for polynomial coalgebras. In *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 122–136, 2009.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
4. J. Goguen, K. Lin, and G. Rosu. Circular coinductive rewriting. In *ASE '00: Proceedings of the 15th IEEE international conference on Automated software engineering*, pages 123–132, Washington, DC, USA, 2000. IEEE Computer Society.
5. J. A. Goguen. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.
6. E.-I. Goriac, D. Lucanu, and G. Roşu. Automating Coinduction with Case Analysis. Technical Report TR 10-05, "Al.I.Cuza" University of Iaşi, Faculty of Computer Science, 2010. URL:http://www.infoiasi.ro/ tr/tr.pl.cgi.
7. B. Jacobs. Introduction to coalgebra. towards mathematics of states and observations, 2005.
8. S. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956.
9. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *LICS*, pages 214–225. IEEE Computer Society, 1991.
10. D. Kozen. Myhill-nerode relations on automatic systems and the completeness of Kleene algebra. In A. Ferreira and H. Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2001.
11. D. Lucanu, E.-I. Goriac, G. Caltais, and G. Roşu. CIRC : A behavioral verification tool based on circular coinduction. In *CALCO 2009*, volume 5728 of *LNCS*, pages 433–442. Springer, 2009.
12. G. Roşu and D. Lucanu. Circular Coinduction – A Proof Theoretical Foundation. In *CALCO'09*, LNCS, 2009.
13. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
14. A. Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966.
15. A. Silva, M. M. Bonsangue, and J. J. M. M. Rutten. Non-deterministic Kleene coalgebras. *LMCS*, 2010.