CWI

Kleene Coalgebras

A.M. Silva, M.M. Bonsangue, J.J.M.M. Rutten

SEN-1001

Centrum Wiskunde & Informatica

Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Science Park 123, 1098 XG Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

KLEENE COALGEBRAS

ALEXANDRA SILVA, MARCELLO BONSANGUE, AND JAN RUTTEN

CWI, Amsterdam, The Netherlands
e-mail address: ams@cwi.nl

LIACS, University of Leiden,The Netherlands
e-mail address: marcello@liacs.nl

CWI (Amsterdam), VUA (Amsterdam) and RUN (Nijmegen) , The Netherlands
e-mail address: janr@cwi.nl

ABSTRACT. In this paper, we present a systematic way of deriving (1) languages of (generalised) regular expressions, and (2) sound and complete axiomatizations thereof, for a wide variety of systems. This generalizes both the results of Kleene (on regular languages and deterministic finite automata) and Milner (on regular behaviours and finite labelled transition systems), and includes many other systems such as Mealy and Moore machines.

1. INTRODUCTION

In a previous paper [7], we presented a language to describe the behaviour of Mealy machines and a sound and complete axiomatization thereof. These can be seen as the analogue of classical regular expressions [18] and Kleene algebra [19], for deterministic finite automata (DFA), or the process algebra and axiomatization for labelled transition systems (LTS) [25].

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

Under mild conditions, functors G have a *final coalgebra* (unique up to isomorphism) into which every G-coalgebra can be mapped via a unique so-called *G-homomorphism*. The

1998 ACM Subject Classification: MANDATORY list of acm classifications.
Key words and phrases: Coalgebra, Kleene's theorem, axiomatization.
The first author was partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

LOGICAL METHODS
IN COMPUTER SCIENCE
DOI:10.2168/LMCS-???

final coalgebra can be viewed as the universe of all possible *G-behaviours*: the unique homomorphism into the final coalgebra maps every state of a coalgebra to a canonical representative of its behaviour. This provides a general notion of behavioural equivalence: two states are equivalent iff they are mapped to the same element of the final coalgebra. Instantiating the notion of final coalgebra for the aforementioned examples, the result is as expected: for DFA's the final coalgebra is the set 2^{A^*} of all languages over A; for Mealy machines it is the set of causal functions $f : A^\omega \to B^\omega$; and for LTS's it is the set of finitely branching trees with arcs labelled by $a \in A$. The notion of equivalence also specializes to the familiar notions: for DFA's, two states are equivalent when they accept the same language; for Mealy machines, if they accept the same causal function; and for LTS's if they are bisimilar.

It is the main aim of this paper to show how the type of a system, given by the functor G, is not only enough to determine a notion of behaviour and behavioural equivalence, but also allows for a uniform derivation of both a set of expressions describing behaviour and a corresponding axiomatization. The theory of universal coalgebra [28] provides a standard equivalence and a universal domain of behaviours, uniquely based on the functor G. The main contributions of this paper are (1) the definition of a set of expressions Exp_G describing G-behaviours, (2) the proof of the coincidence between behaviours described by $\varepsilon \in Exp_G$ and locally finite G-coalgebras (this is the analogue of Kleene's theorem), and (3) a corresponding sound and complete axiomatization, wrt bisimulation, of Exp_G (this is the analogue of Kleene algebra). All of these results in this paper are solely based on the type of the system, given by the functor on G.

Organization of the paper. In Section 2 we introduce the class of non-deterministic functors and coalgebras. In Section 3 we associate with each non-deterministic functor G a generalized language Exp_G of regular expressions and we present an analogue of Kleene's theorem, which makes precise the connection between Exp_G and G-coalgebras. A sound and complete axiomatization of Exp_G is presented in Section 4. Section 6 presents concluding remarks, directions for future work and discusses related work.

2. PRELIMINARIES

We give the basic definitions on non-deterministic functors and coalgebras and introduce the notion of bisimulation.

First we fix notation on sets and operations on them. Let **Set** be the category of sets and functions. Sets are denoted by capital letters $X, Y, \ldots$ 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 $\mathscr{P}_\omega(X) = \{Y \subseteq X \mid Y \text{ finite}\}$. The collection of functions from a set X to a set Y is denoted by Y^X. We write id_X for the identity function on set X. Given functions $f : X \to Y$ and $g : Y \to Z$ we write their composition as $g \circ f$. The product of two sets X, Y is written as $X \times Y$, with projection functions $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$ .The set 1 is a singleton set typically written as $1 = \{*\}$ and it can be regarded as the empty product. We define $X \oplus Y$ 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 $X \oplus Y$ 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. The intuition behind the need of these extra elements will become clear

when we present our language of expressions and concrete examples, in Section 3.3.1, of systems whose type involves $\oplus$. It is interesting to remark that $X \oplus X \ncong 2 \times X \cong X + X$.

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

$$\begin{array}{ll} f_1 \times f_2 : X \times Z \to Y \times W & f_1 \oplus f_2 : X \oplus Z \to Y \oplus W \\ (f_1 \times f_2)(\langle x, z\rangle) = \langle f_1(x), f_2(z)\rangle & (f \oplus f_2)(c) = c,\ c \in \{\bot, \top\} \\ & (f_1 \oplus f_2)(\kappa_i(x)) = \kappa_i(f_i(x)),\ i \in \{1,2\} \\ f_1^A : X^A \to Y^A & \mathscr{P}_\omega(f_1) : \mathscr{P}_\omega(X) \to \mathscr{P}_\omega(Y) \\ f_1^A(g) = \lambda a.f_1(g(a)) & \mathscr{P}_\omega(f_1)(S) = \{f_1(x) \mid x \in S\} \end{array}$$

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

In our definition of non-deterministic functors we will use constant sets equipped with an information order. In particular, we will use join-semilattices. A (bounded) join-semilattice is a set B equipped with a binary operation $\vee_B$ and a constant $\bot_B \in B$, such that $\vee_B$ is commutative, associative and idempotent. The element $\bot_B$ is neutral w.r.t. $\vee_B$. As usual, $\vee_B$ gives rise to a partial ordering $\leq_B$ on the elements of B:

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

Every set S can be transformed into a join-semilattice by taking B to be the set of all finite subsets of S with union as join.

Non-deterministic functors. Non-deterministic functors are functors G: **Set** $\to$ **Set**, built inductively from the identity and constants, using $\times$, $\oplus$, $(-)^A$ and $\mathscr{P}_\omega$.

2.1. **Definition.** The class *NDF* of *non-deterministic functors* on **Set** is inductively defined by putting:

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

where B is a finite join-semilattice and A is a finite set. ♣

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

We now show the explicit definition of the functors above on a set X and on a morphism $f : X \to Y$ (note that $G(f) : G(X) \to G(Y)$).

$$\begin{array}{lll} Id(X) = X & B(X) = B & (G_1 \oplus G_2)(X) = G_1(X) \oplus G_2(X) \\ Id(f) = f & B(f) = id_B & (G_1 \oplus G_2)(f) = G_1(f) \oplus G_2(f) \\ (G^A)(X) = G(X)^A & (\mathscr{P}_\omega G)(X) = \mathscr{P}_\omega(G(X)) & (G_1 \times G_2)(X) = G_1(X) \times G_2(X) \\ (G^A)(f) = G(f)^A & (\mathscr{P}_\omega G)(f) = \mathscr{P}_\omega(G(f)) & (G_1 \times G_2)(f) = G_1(f) \times G_2(f) \end{array}$$

Typical examples of non-deterministic functors include $M = (B \times Id)^A$, $D = 2 \times Id^A$, $P = (1 + Id)^A$ and $N = 2 \times (\mathscr{P}_\omega Id)^A$. These functors represent, respectively, the type of Mealy, deterministic, partial deterministic and non-deterministic automata. In this paper, we will use the last three as running examples. In [7], we have studied in detail regular expressions for Mealy automata. Similarly to what happened there, we impose a join-semilattice structure on the constant functor. The product, exponentiation and powerset functors preserve the

join-semilattice structure and thus need not to be changed. This is not the case for the classical coproduct and thus we use $\oplus$ instead, which also guarantees that the join semilattice structure is preserved.

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

2.2. **Definition.** Let $\lhd \subseteq NDF \times NDF$ be the least reflexive and transitive relation on non-deterministic functors such that

$$G_1 \lhd G_1 \times G_2, \quad G_2 \lhd G_1 \times G_2, \quad G_1 \lhd G_1 \oplus G_2, \quad G_2 \lhd G_1 \oplus G_2, \quad G \lhd G^A, \quad G \lhd \mathscr{P}_\omega G$$

♣

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

Non-deterministic coalgebras. A non-deterministic coalgebra is a pair $(S, f : S \to G(S))$, where S is a set of states and G is a non-deterministic functor. The functor G, together with the function f, determines the *transition structure* (or dynamics) of the G-coalgebra [28]. Mealy, deterministic, partial deterministic and non-deterministic automata are, respectively, coalgebras for the functors $M = (B \times Id)^A$, $D = 2 \times Id^A$, $P = (1 + Id)^A$ and $N = 2 \times (\mathscr{P}_\omega Id)^A$.

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 $g \circ h = G(h) \circ f$.

2.3. **Definition.** 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 non-deterministic functor G there exists a final G-coalgebra (Ω_G, ω_G) [28]. For instance, as we already mentioned in the introduction, the final coalgebra for the functor D is the set of languages 2^{A^*} over A, together with a transition function $d : 2^{A^*} \to 2 \times (2^{A^*})^A$ defined as $d(\phi) = \langle \phi(\epsilon), \lambda a \lambda w.\phi(aw)\rangle$. Here ϵ denotes the empty sequence and aw denotes the word resulting from prefixing w with the letter a. The notion of finality will play a key role later in providing a semantics to expressions.

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 $g : V \to G(V)$ such that i is a homomorphism. Given $s \in S$, $\langle s\rangle \subseteq S$ denotes the smallest subcoalgebra generated by s, *i.e.* the set of states that are reachable from s.

We will write $Coalg(G)$ for the category of G-coalgebras together with coalgebra homomorphisms. We also write $Coalg_{\mathbf{LF}}(G)$ 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.

Let (S, f) and (T, g) be two G-coalgebras. We call a relation $R \subseteq S \times T$ a *bisimulation* [16] iff

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

where $\overline{G}(R)$ is defined as

$$\overline{G}(R) = \{\langle G(\pi_1)(x), G(\pi_2)(x)\rangle \mid x \in G(R)\}$$

For an inductive definition of $\overline{G}$ we refer to [16].

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 all G-coalgebras (S,f) and (T,g) and $s \in S, t \in T$, it holds that $s \sim t \iff \mathbf{beh}(s) = \mathbf{beh}(t)$ (the left to right implication always holds, whereas the right to left implication only holds for certain classes of functors, which include the ones we consider in this paper [28, 31]).

3. A LANGUAGE OF EXPRESSIONS FOR NON-DETERMINISTIC COALGEBRAS

In this section, we generalize the classical notion of regular expressions to non-deterministic coalgebras. We start by introducing an untyped language of expressions and then we single out the well-typed ones via an appropriate typing system, thereby associating expressions to non-deterministic functors.

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

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

where γ is a *guarded expression* given by:

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

♣

In the expression $\mu x.\gamma$, μ is a binder for all the 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 guarded and closed expressions by *Exp*.

Intuitively, expressions denote elements of the final coalgebra. The expressions $\emptyset$, $\varepsilon_1 \oplus \varepsilon_2$ and $\mu x.\varepsilon$ will play a similar role to, respectively, the empty language, the union of languages and the Kleene star in classical regular expressions for deterministic automata. The expressions $l\langle\varepsilon\rangle$ and $r\langle\varepsilon\rangle$ refer to the left and right hand-side of products. Similarly, $l[\varepsilon]$ and $r[\varepsilon]$ refer to the left and right hand-side of sums. The expressions $a(\varepsilon)$ and $\{\varepsilon\}$ denote function application and a singleton set, respectively. We shall soon illustrate, by means of examples, the role of these expressions. Here, it is already visible that our approach (to define a language) for the powerset functor differs from classical modal logic 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 powerset: every finite set can be written as the finite union of its singletons.

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

Next, we present a typing assignment system for associating expressions to non-deterministic functors. This will 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.

3.2. **DEFINITION** (Type system)**.** We now define a typing relation $\vdash \subseteq Exp \times NDF \times NDF$ that will associate an expression ε with two non-deterministic functors F and G, which are

related by the ingredient relation (F is an ingredient of G). We shall write $\varepsilon : F \lhd G$ for $\langle \varepsilon, F, G\rangle \in \vdash$. The rules that define $\vdash$ are the following:

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

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

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l\langle\varepsilon\rangle : F_1 \times F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r\langle\varepsilon\rangle : F_1 \times F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l[\varepsilon] : F_1 \Plus F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r[\varepsilon] : F_1 \Plus F_2 \lhd G}$$

♣

Intuitively, $\varepsilon : F \lhd G$ means that ε is an element (up to bisimulation) of $F(\Omega_G)$, where Ω_G is the final coalgebra of G. As expected, there is a rule for each expression construct. The extra rule involving $Id \lhd G$ reflects the isomorphism between the final coalgebra Ω_G and $G(\Omega_G)$. Only fixed-points at the outermost level of the functor are allowed. This does not mean however that we disallow nested fixed-points. For instance, $\mu x.a(x \oplus \mu y.a(y))$ would be a well-typed expression for the functor D of deterministic automata, as it will become clear below, when we will present more examples of well-typed and non-well-typed expressions. The presented type system is decidable (expressions are of finite length and the system is inductive on the structure of $\varepsilon \in Exp$).

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

3.3. **Definition** (G-expressions)**.** Let G be a non-deterministic functor and F an ingredient of G. We denote by $Exp_{F \lhd G}$ the following 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 $Exp_{G \lhd G}$. ♣

Examples of well-typed expressions for the functor $D = (2 \times Id)$ (with $2 = \{0, 1\}$; recall that the ingredients of D are 2, $2 \times Id$ and D itself) include $r\langle a(\emptyset)\rangle$, $l\langle 1\rangle \oplus r\langle a(l\langle 0\rangle)\rangle$ and $\mu x.r\langle a(x)\rangle \oplus l\langle 1\rangle$. The expressions $l[1]$, $l\langle 1\rangle \oplus 1$ and $\mu x.1$ are examples of non well-typed expressions, because the functor D does not involve $\Plus$, the subexpressions in the sum have different type, and recursion is not at the outermost level (1 has type $2 \lhd D$), respectively.

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

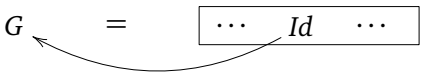
3.4. **Example** (Deterministic expressions)**.** Let A be a finite set of input actions and let X be a set of (recursion or) fixed-point variables. The set Exp_D of *deterministic expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For $a \in A$ and $x \in X$:

$$\begin{aligned}\varepsilon &::= \emptyset \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid l\langle\varepsilon_1\rangle \mid r\langle\varepsilon_2\rangle\\ \varepsilon_1 &::= \emptyset \mid 0 \mid 1 \mid \varepsilon_1 \oplus \varepsilon_1\\ \varepsilon_2 &::= \emptyset \mid a(\varepsilon) \mid \varepsilon_2 \oplus \varepsilon_2\end{aligned}$$

♠

It is easy to see that the closed (and guarded) expressions generated by the grammar presented above are exactly the elements of Exp_D. Te most interesting case to check is the

expression $r\langle a(\varepsilon)\rangle$. Note that $a(\varepsilon)$ has type $Id^A \lhd D$ as long as ε has type $Id \lhd D$. And the crucial remark here is that, by definition of $\vdash$, $Exp_{Id \lhd G} = Exp_G$. Intuitively, this can be explained by the fact that for a non-deterministic functor G, if Id is one of the ingredients of G, then it is functioning as a pointer to the functor being defined:



Therefore, ε has type $Id \lhd D$ if it is of type $D \lhd D$, or more precisely, if $\varepsilon \in Exp_D$, which explains why the grammar above is correct.

At this point, we should remark that the syntax of our expressions differs from the classical regular expressions in the use of μ and action prefixing $a(\varepsilon)$ instead of star and full concatenation. We shall prove later that these two syntactically different formalisms are equally expressive (Theorems 3.12 and 3.14), but, to increase the intuition behind our expressions, let us now present the syntactic translation from classical regular expressions to Exp_D (this translation is inspired by [25]) and back.

3.5. **Definition.** The set of regular expressions is given by the following syntax

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

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

$(-)^\dagger : RE \to Exp_D$		$(-)^\ddagger : Exp_D \to RE$	
$(\underline{\emptyset})^\dagger$	$= \emptyset$	$(\emptyset)^\ddagger$	$= \underline{\emptyset}$
$(\epsilon)^\dagger$	$= l\langle 1\rangle$	$(l\langle 0\rangle)^\ddagger$	$= \underline{\emptyset}$
$(a)^\dagger$	$= r\langle a(l\langle 1\rangle)\rangle$	$(l\langle 1\rangle)^\ddagger$	$= \epsilon$
$(r_1 + r_2)^\dagger$	$= (r_1)^\dagger \oplus (r_2)^\dagger$	$(l\langle b_1 \oplus b_2\rangle)^\ddagger$	$= (l\langle b_1\rangle)^\ddagger + (l\langle b_2\rangle)^\ddagger$
$(r_1 \cdot r_2)^\dagger$	$= (r_1)^\dagger[(r_2)^\dagger/1]$	$(r\langle a(\varepsilon)\rangle)^\ddagger$	$= a \cdot (\varepsilon)^\ddagger$
$(r^*)^\dagger$	$= \mu x.(r)^\dagger[x/1] \oplus l\langle 1\rangle$	$(r\langle a(\varepsilon_1) \oplus a(\varepsilon_2)\rangle)^\ddagger$	$= (r\langle a(\varepsilon_1)\rangle)^\ddagger + (r\langle a(\varepsilon_2)\rangle)^\ddagger$
		$(\varepsilon_1 \oplus \varepsilon_2)^\ddagger$	$= (\varepsilon_1)^\ddagger + (\varepsilon_2)^\ddagger$
		$(\mu x.\varepsilon)^\ddagger$	$= \mathbf{sol}(\mathbf{eqs}(\mu x.\varepsilon))$

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

The translations above have the expected properties: $r \sim r^\dagger$ and $\varepsilon \sim \varepsilon^\ddagger$. ♣

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

We present next the syntax for the expressions in Exp_P and in Exp_N.

3.6. **Example** (Partial expressions)**.** Let A be a finite set of input actions and X be a set of (recursion or) fixed-point variables. The set Exp_P of *partial expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For $a \in A$ and

$x \in X$:

$$\begin{aligned}\varepsilon &::= \emptyset \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid a(\varepsilon_1)\\ \varepsilon_1 &::= \emptyset \mid \varepsilon_1 \oplus \varepsilon_1 \mid l[\varepsilon_2] \mid r[\varepsilon]\\ \varepsilon_2 &::= \emptyset \mid \varepsilon_2 \oplus \varepsilon_2 \mid *\end{aligned}$$

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

3.7. **Example** (Non-deterministic expressions)**.** Let A be a finite set of input actions and X be a set of (recursion or) fixed-point variables. The set Exp_N of *non-deterministic expressions* is given by the set of closed and guarded expressions generated by the following BNF grammar. For $a \in A$ and $x \in X$:

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

Intuitively, the expression $r\langle a(\{\varepsilon_1\} \oplus \{\varepsilon_2\})\rangle$ specifies a state which has two outgoing transitions labelled with the input letter a, one to a state specified by ε_1 and another to a state specified by ε_2. ♠

We have now defined a language of expressions which gives us an algebraic description of systems. We should also remark at this point that in the examples we strictly follow the type system to derive the syntax of the expressions. However, it is obvious that many simplifications can be made in order to obtain a more polished language. In particular, after the axiomatization we will be able to decrease the number of levels in the above grammars, since will we have axioms of the shape $a(\varepsilon) \oplus a(\varepsilon') \equiv a(\varepsilon \oplus \varepsilon')$. In Section 4.2, we will sketch two examples where we apply some simplification to the syntax.

The goal is now to present a generalization of Kleene's theorem for non-deterministic coalgebras (Theorems 3.12 and 3.14). Recall that, for regular languages, the theorem states that a language is regular if and only if it is recognized by a finite automaton. In order to achieve our goal we will first show that the set Exp_G of G-expressions carries a G-coalgebra structure.

3.1. **Expressions are coalgebras.** In this section, we show that the set of G-expressions for a given non-deterministic functor G has a coalgebraic structure $\delta_G : Exp_G \to G(Exp_G)$. More precisely, we are going to define a function

$$\delta_{F \lhd G} : Exp_{F \lhd G} \to F(Exp_G)$$

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

3.8. **Definition.** For every $G \in NDF$ and for every F with $F \lhd G$:

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

$$\begin{array}{lcllcl} Empty_{Id \lhd G} & = & \emptyset & Empty_{F_1 \oplus F_2 \lhd G} & = & \bot \\ Empty_{B \lhd G} & = & \bot_B & Empty_{F^A \lhd G} & = & \lambda a.Empty_{F \lhd G} \\ Empty_{F_1 \times F_2 \lhd G} & = & \langle Empty_{F_1 \lhd G}, Empty_{F_2 \lhd G}\rangle & Empty_{\mathscr{P}_\omega F \lhd G} & = & \{\} \end{array}$$

(ii) we define a function $Plus_{F\lhd G}\colon F(Exp_G)\times F(Exp_G)\to F(Exp_G)$ by induction on the syntactic structure of F:

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

Intuitively, one can think of the constant $Empty_{F\lhd G}$ and the function $Plus_{F\lhd G}$ as liftings of $\emptyset$ and $\oplus$ to the level of $F(Exp_G)$. ♣

The last ingredient we need to define $\delta_{F\lhd G}$ is a measure $N(\varepsilon)$ that counts the maximum number of nested unguarded ocurrences of μ-expressions in ε. We say that a subexpression $\mu x.\varepsilon_1$ of ε occurs unguarded if it is not in the scope of one of the operators $l\langle-\rangle$, $r\langle-\rangle$, $l[-]$, $r[-]$, $a(-)$ or $\{-\}$.

3.9. **Definition.** For every $\varepsilon\in Exp_{F\lhd G}$, we define $N(\varepsilon)$ as follows:

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

♣

Now we have all we need to define $\delta_{F\lhd G}\colon Exp_{F\lhd G}\to F(Exp_G)$. This function will be defined by double induction on the maximum number $N(\varepsilon)$ of nested unguarded occurrences of μ-expressions in ε and on the height of the proofs for typing expressions.

3.10. **Definition.** For every ingredient F of a non-deterministic functor G and an expression $\varepsilon\in Exp_{F\lhd G}$, we define $\delta_{F\lhd G}(\varepsilon)$ as follows:

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

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

In order to see that the definition of $\delta_{F \lhd G}$ is well-formed, note the interplay between the two inductions: the height of the typing proof of the arguments in the recursive calls is strictly decreasing, except in the case of $\mu x.\varepsilon$; but, in this case we have that $N(\varepsilon) = N(\varepsilon[\mu x.\varepsilon/x])$, which can easily be proved by (standard) induction on the syntactic structure of ε, since ε is guarded (in x), and it guarantees that $N(\varepsilon[\mu x.\varepsilon/x]) < N(\mu x.\varepsilon)$. Also note that clause 4 of the above definition overlaps with clauses 1 and 2 (by taking $F = Id$). However, they give the same result and thus the function $\delta_{F \lhd G}$ is well-defined.

3.11. **Definition.** We can now define, for each non-deterministic functor G, a G-coalgebra

$$\delta_G : Exp_G \to G(Exp_G)$$

by putting $\delta_G = \delta_{G \lhd G}$. ♣

It is interesting to remark that δ_G is the generalization of the well-known notion of Brzozowski derivative [11] for regular expressions and, moreover, it provides an operational semantics for expressions, as we shall see in Section 3.2.

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

3.2. **Expressions are expressive.** Having a G-coalgebra structure on Exp_G has two advantages. First, it provides us, by finality, directly with a natural semantics because of the existence of a (unique) homomorphism **beh**: $Exp_G \to \Omega_G$, that assigns to every expression ε an element **beh**(ε) of the final coalgebra Ω_G.

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

The following theorem states that every state in a finite G-coalgebra can be represented by an expression in our language. This generalizes *half* of Kleene's theorem for deterministic automata: if a language is accepted by a finite automaton then it is regular (*i.e.* it can be denoted by a regular expression). The generalization of the other *half* of the theorem (if a language is regular then it is accepted by a finite automaton) will be presented in Section 3.3.

3.12. **Theorem.** *Let G be a non-deterministic functor and let (S, g) be a locally-finite G-coalgebra. Then, for any $s \in S$, there exists an expression $\langle\!\langle\, s \,\rangle\!\rangle \in Exp_G$ such that $s \sim \langle\!\langle\, s \,\rangle\!\rangle$.*

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

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

For $G \neq Id$, we proceed in the following way. Let, for every i, $A_i = \mu x_i.\gamma^G_{g(s_i)}$ where, for $F \lhd G$ and $s' \in F\langle s \rangle$, the expression $\gamma^F_{s'} \in Exp_{F \lhd G}$ is defined by induction on the structure of

F:

$$\gamma^{Id}_{s_i} = x_i \qquad \gamma^{B}_{b} = b \qquad \gamma^{F_1\times F_2}_{\langle s,s'\rangle} = l\langle \gamma^{F_1}_{s}\rangle \oplus r\langle \varepsilon^{F_2}_{s'}\rangle \qquad \gamma^{F^A}_{f} = \bigoplus_{a\in A} a(\gamma^{F}_{f(a)})$$

$$\gamma^{F_1 \oplus F_2}_{\kappa_1(s)} = l[\gamma^{F_1}_{s}] \qquad \gamma^{F_1 \oplus F_2}_{\kappa_2(s)} = r[\gamma^{F_2}_{s}] \qquad \gamma^{F_1 \oplus F_2}_{\bot} = \emptyset \qquad \gamma^{F_1 \oplus F_2}_{\top} = l[\emptyset] \oplus r[\emptyset]$$

$$\gamma^{\mathscr{P}_\omega F}_{T} = \begin{cases} \bigoplus_{t\in T}\{\gamma^{F}_{t}\} & T \neq \{\} \\ \emptyset & \text{otherwise} \end{cases}$$

Note that here 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.

Now, let $A^0_i = A_i$, define $A^{k+1}_i = A^k_i\{A^k_{k+1}/x_{k+1}\}$ and then set $\langle\!\langle s_i \rangle\!\rangle = A^n_i$. Here, $A\{A'/x\}$ denotes syntactic replacement (that is, substitution without renaming of free variables in A' that become bound in A).

Observe that the term

$$A^n_i = (\mu x_i.\gamma^{G}_{g(s_i)})\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\}$$

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

It remains to prove that $s_i \sim \langle\!\langle s_i \rangle\!\rangle$. We show that $R = \{\langle s_i, \langle\!\langle s_i \rangle\!\rangle\rangle \mid s_i \in \langle s\rangle\}$ is a bisimulation. For that, we first define, for $F \lhd G$ and $s' \in F\langle s\rangle$, $\xi^F_{s'} = \gamma^F_{s'}\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\}$ and the relation

$$R_{F\lhd G} = \{\langle s', \delta_{F\lhd G}(\xi^F_{s'})\rangle \mid s' \in F\langle s\rangle\}.$$

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

① By induction on the structure of F.

$\boxed{F = Id}$ Note that $R_{Id\lhd G} = \{\langle s_i, \xi^{Id}_{s_i}\rangle \mid s_i \in \langle s\rangle\}$ which is equal to $Id(R) = R$ provided that $\xi^{Id}_{s_i} = \langle\!\langle s_i \rangle\!\rangle$. The latter is indeed the case:

$$\begin{array}{rcll} \xi^{Id}_{s_i} & = & \gamma^{Id}_{s_i}\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\} & (\text{def. } \xi^{Id}_{s_i}) \\ & = & x_i\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\} & (\text{def. } \gamma^{Id}_{s_i}) \\ & = & A^{i-1}_i\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\} & (\{A^{i-1}_i/x_i\}) \\ & = & A^0_i\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\} & (\text{def. } A^{i-1}_i) \\ & = & \langle\!\langle s_i \rangle\!\rangle & (\text{def. } \langle\!\langle s_i \rangle\!\rangle) \end{array}$$

$\boxed{F = B}$ Note that, for $b \in B$, $\xi^B_b = \gamma^B_b\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\} = b$. Thus, we have that $R_{B\lhd G} = \{\langle s_i, B_{s_i}\rangle \mid s_i \in B\langle s\rangle\} = \{\langle b, b\rangle \mid b \in B\} = \overline{B}(R)$.

$\boxed{F = F_1 \times F_2}$

$$
\begin{array}{lll}
 & \langle\langle u_1, v_1\rangle, \langle u_2, v_2\rangle\rangle \in \overline{F_1 \times F_2}(R) & \\
\iff & \langle u_1, u_2\rangle \in \overline{F_1}(R) \text{ and } \langle v_1, v_2\rangle \in \overline{F_2}(R) & (\text{def. } \overline{F_1 \times F_2}) \\
\iff & \langle u_1, u_2\rangle \in R_{F_1 \lhd G} \text{ and } \langle v_1, v_2\rangle \in R_{F_2 \lhd G} & (\text{ind. hyp.}) \\
\iff & \langle u_1, u_2\rangle = \langle s_1', \delta_{F_1 \lhd G}(\xi^{F_1}_{s_1'})\rangle \text{ and } \langle v_1, v_2\rangle = \langle s_2', \delta_{F_2 \lhd G}(\xi^{F_2}_{s_2'})\rangle & (\text{def. } R_{F_i \lhd G}) \\
\iff & \langle u_1, v_1\rangle = \langle s_1', s_2'\rangle \text{ and } \langle u_2, v_2\rangle = \delta_{F_1 \times F_2 \lhd G}(l(\xi^{F_1}_{s_1'}) \oplus r(\xi^{F_2}_{s_2'})) & (\text{def. } \delta_{F \lhd G}) \\
\iff & \langle u_1, v_1\rangle = \langle s_1', s_2'\rangle \text{ and } \langle u_2, v_2\rangle = \delta_{F_1 \times F_2 \lhd G}(\xi^{F_1 \times F_2}_{\langle s_1', s_2'\rangle}) & (\text{def. } \xi^F) \\
\iff & \langle\langle u_1, v_1\rangle, \langle u_2, v_2\rangle\rangle \in R_{F_1 \times F_2 \lhd G} &
\end{array}
$$

$\boxed{F = F_1 + F_2}$, $\boxed{F = F_1^A}$ and $\boxed{F = \mathscr{P}_\omega F_1}$: similar to $F_1 \times F_2$.

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

$$
\begin{array}{ll}
 & \delta_G(\langle\!\langle s_i \rangle\!\rangle) \\
= & \delta_G((\mu x_i.\gamma^G_{g(s_i)})\{A^0_1/x_1\}\ldots\{A^{n-1}_n/x_n\}) \qquad (\text{def. of } \langle\!\langle s_i \rangle\!\rangle) \\
= & \delta_G(\mu x_i.\gamma^G_{g(s_i)}\{A^0_1/x_1\}\ldots\{A^{i-2}_{i-1}/x_{i-1}\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\}) \\
= & \delta_G(\gamma^G_{g(s_i)}\{A^0_1/x_1\}\ldots\{A^{i-2}_{i-1}/x_{i-1}\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\}[A^n_i/x_i]) \qquad (\text{def. of } \delta_G) \\
= & \delta_G(\gamma^G_{g(s_i)}\{A^0_1/x_1\}\ldots\{A^{i-2}_{i-1}/x_{i-1}\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\}\{A^n_i/x_i\}) \qquad ([A^n_i/x_i] = \{A^n_i/x_i\}) \\
= & \delta_G(\gamma^G_{g(s_i)}\{A^0_1/x_1\}\ldots\{A^{i-2}_{i-1}/x_{i-1}\}\{A^n_i/x_i\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\}) \\
= & \delta_G(\xi^G_{g(s_i)})
\end{array}
$$

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

$$
\begin{array}{ll}
 & \{A^n_i/x_i\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\} \\
= & \{A^{i-1}_i\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\}/x_i\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\} \\
= & \{A^{i-1}_i/x_i\}\{A^i_{i+1}/x_{i+1}\}\ldots\{A^{n-1}_n/x_n\} \qquad (3.1)
\end{array}
$$

Equation (3.1) uses the syntactic identity

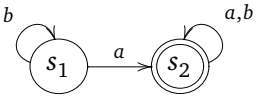
$$A\{B\{C/y\}/x\} = A\{B/x\}\{C/y\} \qquad (3.2)$$

and the fact that, for all i, one has $A\{A^{i-1}_i/x_i\}\{A^{i-1}_i/x_i\} = A\{A^{i-1}_i/x_i\}$, since the first replacement bounds variable x_i in A and, thus, the second replacement is innocuous.

□

Let us illustrate the construction appearing in the proof of Theorem 3.12 by some examples. These examples will illustrate the similarity with the proof Kleene's Theorem presented in most textbooks, where a regular expression denoting the language recognized by a state of a deterministic automaton is built using a system of equations.

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



Now define $A_1 = \mu x_1.\gamma^D_{g(s_1)}$ and $A_2 = \mu x_2.\gamma^D_{g(s_2)}$ where

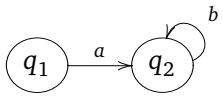
$$\gamma^D_{g(s_1)} = l\langle 0\rangle \oplus r\langle b(x_1) \oplus a(x_2)\rangle \quad \gamma^D_{g(s_2)} = l\langle 1\rangle \oplus r\langle a(x_2) \oplus b(x_2)\rangle$$

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

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

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

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



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

We will have the equations

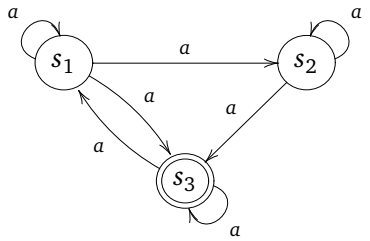
$$A_1 = A_1^0 = A_1^1 = \mu x_1.b(l[*]) \oplus a(x_2)\{A_2^1/x_2\}$$
$$A_2 = A_2^0 = A_2^1 = \mu x_2.a(l[*]) \oplus b(x_2)$$

Thus:

$$\langle\!\langle s_1 \rangle\!\rangle = A_1^2 = \mu x_1.\, b(l[*]) \oplus a(\mu x_2.\, a(l[*]) \oplus b(x_2))$$
$$\langle\!\langle s_2 \rangle\!\rangle = \mu x_2.a(l[*]) \oplus b(x_2)$$

Again we have $s_1 \sim \langle\!\langle s_1 \rangle\!\rangle$ and $s_2 \sim \langle\!\langle s_2 \rangle\!\rangle$.

As a last example, let us consider the following non-deterministic automaton, over a one letter alphabet $A = \{a\}$:



We start with the equations:

$$\begin{aligned} A_1 &= \mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\} \oplus \{x_2\} \oplus \{x_3\})\rangle \\ A_2 &= \mu x_2.l\langle 0\rangle \oplus r\langle a(\{x_2\} \oplus \{x_3\})\rangle \\ A_3 &= \mu x_3.l\langle 1\rangle \oplus r\langle a(\{x_1\} \oplus \{x_3\})\rangle \end{aligned}$$

Then we have the following iterations:

$$
\begin{aligned}
A_1^1 &= A_1\\
A_1^2 &= A_1\{A_2^1/x_2\} = \mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\}\oplus\{A_2\}\oplus\{x_3\})\rangle\\
A_1^3 &= A_1\{A_2^1/x_2\}\{A_3^2/x_3\} = \mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\}\oplus\{(A_2\{A_3^2/x_3\})\}\oplus\{A_3^2\})\rangle
\end{aligned}
$$

$$
\begin{aligned}
A_2^1 &= A_2\{A_1/x_1\} = A_2\\
A_2^2 &= A_2\{A_1/x_1\} = A_2\\
A_2^3 &= A_2\{A_1/x_1\}\{A_3^2/x_3\} = \mu x_2.l\langle 0\rangle \oplus r\langle a(\{x_2\}\oplus\{A_3^2\})\rangle
\end{aligned}
$$

$$
\begin{aligned}
A_3^1 &= A_3\{A_1/x_1\} = \mu x_3.l\langle 1\rangle \oplus r\langle a(\{A_1\}\oplus\{x_3\})\rangle\\
A_3^2 &= A_3\{A_1/x_1\}\{A_2^1/x_2\} = \mu x_3.l\langle 1\rangle \oplus r\langle a(\{(A_1\{A_2^1/x_2\})\}\oplus\{x_3\})\rangle\\
A_3^3 &= A_3^2
\end{aligned}
$$

This yields the following expressions:

$$
\begin{aligned}
&\langle\!\langle s_1\rangle\!\rangle = \mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\}\oplus\{\langle\!\langle s_2\rangle\!\rangle\}\oplus\{\langle\!\langle s_3\rangle\!\rangle\})\rangle\\
&\langle\!\langle s_2\rangle\!\rangle = \mu x_2.l\langle 0\rangle \oplus r\langle a(\{x_2\}\oplus\{\langle\!\langle s_3\rangle\!\rangle\})\rangle\\
&\langle\!\langle s_3\rangle\!\rangle = \mu x_3.l\langle 1\rangle \oplus r\langle a(\{\mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\}\oplus\{\mu x_2.l\langle 0\rangle \oplus r\langle a(\{x_2\}\oplus\{x_3\})\rangle\}\oplus\{x_3\})\rangle\}\oplus\{x_3\})\rangle
\end{aligned}
$$

3.3. **Finite systems for expressions.** We now prove the converse of Theorem 3.12, that is, we show how to construct a *finite* G-coalgebra (S,g) from an arbitrary expression $\varepsilon \in Exp_G$, such that there exists a state $s \in S$ with $\varepsilon \sim_G s$.

The immediate way of obtaining a coalgebra from an expression $\varepsilon \in Exp_G$ is to compute the subcoalgebra $\langle\varepsilon\rangle$, since we have provided the set Exp_G with a coalgebra structure $\delta_G : Exp_G \to G(Exp_G)$. However, since the subcoalgebra generated by an expression $\varepsilon \in Exp_G$ by repeatedly applying δ_G is, in general, infinite. Take for instance the deterministic expression $\varepsilon_1 = \mu x.\, r\langle a(x \oplus \mu y.\, r\langle a(y)\rangle)\rangle$ and observe that:

$$
\begin{aligned}
\delta_D(\varepsilon_1) &= \langle 0, \varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle\rangle\\
\delta_D(\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle) &= \langle 0, \varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle\rangle\\
\delta_D(\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle) &= \langle 0, \varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle\rangle\\
&\vdots
\end{aligned}
$$

As one would expect, all the new states are equivalent and will be identified by **beh** (the morphism into the final coalgebra). However, the function δ_D does not make any state identification and thus yields an infinite coalgebra.

This phenomena occurs also in classical regular expressions. It was shown in [11] that the normalizing the expressions using the axioms for associativity, commutativity and idempotency was enough to guarantee finiteness. We will show in this section that this also holds in our setting.

Consider the following axioms (only the first three are essential, but we include the fourth to obtain smaller automata):

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

We define the relation $\equiv_{ACIE} \subseteq Exp_{F \lhd G} \times Exp_{F \lhd G}$, written infix, as the least reflexive and transitive relation containing the four identities above. The relation $\equiv_{ACIE}$ gives rise to the equivalence map $[\varepsilon]_{ACIE} = \{\varepsilon' \mid \varepsilon \equiv_{ACIE} \varepsilon'\}$. The following diagram shows the maps defined so far:

$$\begin{array}{ccc} Exp_{F \lhd G} & \xrightarrow{[-]_{ACIE}} & Exp_{F \lhd G}/_{\equiv_{ACIE}} \\ {\scriptstyle \delta_{F \lhd G}}\downarrow & & \\ F(Exp_G) & \xrightarrow[F([-]_{ACIE})]{} & F(Exp_G/_{\equiv_{ACIE}}) \end{array}$$

In order to complete the diagram, we next prove that $\equiv_{ACIE}$ is contained in the kernel of $F([-]_{ACIE}) \circ \delta_{F \lhd G}$[1]. This will guarantee the existence of a function

$$\overline{\delta}_{F \lhd G} : Exp_{F \lhd G}/_{\equiv_{ACIE}} \to F(Exp_G/_{\equiv_{ACIE}})$$

which, when $F = G$, provides $Exp_G/_{\equiv}$ with a coalgebraic structure

$$\overline{\delta}_G : Exp_G/_{\equiv_{ACIE}} \to G(Exp_G/_{\equiv_{ACIE}})$$

(as before we write $\overline{\delta}_G$ for $\overline{\delta}_{G \lhd G}$) and which makes $[-]_{ACIE}$ a homomorphism of coalgebras.

3.13. **Lemma.** *Let G and F be non-deterministic functors, with $F \lhd G$. For all $\varepsilon_1, \varepsilon_2 \in Exp_{F \lhd G}$,*

$$\varepsilon_1 \equiv_{ACIE} \varepsilon_2 \Rightarrow (F([-]_{ACIE}))(\delta_{F \lhd G}(\varepsilon_1)) = (F([-]_{ACIE}))(\delta_{F \lhd G}(\varepsilon_2))$$

Proof. In order to improve readability, in this proof we will use $[-]$ to denote $[-]_{ACIE}$.

It is enough to prove that for all $x_1, x_2, x_3 \in F(Exp_{F \lhd G})$:

① $F([-])(Plus_{F \lhd G}(Plus_{F \lhd G}(x_1, x_2), x_3)) = F([-])(Plus_{F \lhd G}(x_1, Plus_{F \lhd G}(x_2, x_3)))$
② $F([-])(Plus_{F \lhd G}(x_1, x_2)) = F([-])(Plus_{F \lhd G}(x_2, x_1))$
③ $F([-])(Plus_{F \lhd G}(x_1, x_1)) = F([-])(x_1)$
④ $F([-])(Plus_{F \lhd G}(Empty_{F \lhd G}, x_1)) = F([-])(x_1)$

By induction on the structure of F. We illustrate a few cases, the omitted ones are proved in a similar way.

$\boxed{F = Id}$ $x_1, x_2, x_3 \in Exp_G$

$$\begin{array}{lll} ① & [Plus_{Id \lhd G}(Plus_{Id \lhd G}(x_1, x_2), x_3)] & \\ = & [(x_1 \oplus x_2) \oplus x_3] & (\text{def. } Plus) \\ = & [x_1 \oplus (x_2 \oplus x_3)] & (Associativity) \\ = & [Plus_{Id \lhd G}(x_1, Plus_{Id \lhd G}(x_2, x_3))] & (\text{def. } Plus) \end{array}$$

$$\begin{array}{lll} ④ & [Plus_{Id \lhd G}(Empty_{Id \lhd G}, x_1)] & \\ = & [\emptyset \oplus x_1] & (\text{def. } Plus \text{ and } Empty) \\ = & [x_1] & (Empty) \end{array}$$

[1]This is equivalent to prove that $Exp_{F \lhd G}/_{\equiv_{ACIE}}$, together with $[-]_{ACIE}$, is the coequalizer of the projection morphisms from $\equiv_{ACIE}$ to $Exp_{F \lhd G}$.

$\boxed{F = F_1 \times F_2}$ $x_1 = \langle u_1, v_1\rangle, x_2 = \langle u_2, v_2\rangle \in F_1 \times F_2(Exp_G)$

$$
\begin{array}{lll}
\textcircled{2} & (F_1 \times F_2)([-])(Plus_{F_1\times F_2 \lhd G}(\langle u_1, v_1\rangle, \langle u_2, v_2\rangle)) & \\
= & \langle F_1([-])(Plus_{F_1\lhd G}(u_1, u_2)), F_2([-])(Plus_{F_2\lhd G}(v_1, v_2))\rangle & \text{(def. } Plus\text{)} \\
= & \langle F_1([-])(Plus_{F_1\lhd G}(u_2, u_1)), F_2([-])(Plus_{F_2\lhd G}(v_2, v_1))\rangle & \text{(ind. hyp.)} \\
= & (F_1 \times F_2)([-])(Plus_{F_1\times F_2 \lhd G}(\langle u_2, v_2\rangle, \langle u_1, v_1\rangle)) & \text{(def. } Plus\text{)}
\end{array}
$$

$$
\begin{array}{lll}
\textcircled{3} & (F_1 \times F_2)([-])(Plus_{F_1\times F_2 \lhd G}(\langle u_1, v_1\rangle, \langle u_1, v_1\rangle)) & \\
= & \langle F_1([-])(Plus_{F_1\lhd G}(u_1, u_1)), F_2([-])(Plus_{F_2\lhd G}(v_1, v_1))\rangle & \text{(def. } Plus\text{)} \\
= & \langle F_1([-])(u_1), F_2([-])(v_1)\rangle & \text{(ind. hyp.)} \\
= & (F_1 \times F_2)([-])(\langle u_1, v_1\rangle) &
\end{array}
$$

$\boxed{F = \mathscr{P}_\omega F_1}$ $x_1, x_2, x_3 \in \mathscr{P}_\omega F_1(Exp_G)$

$$
\begin{array}{lll}
\textcircled{1} & \mathscr{P}_\omega F_1([-])(Plus_{\mathscr{P}_\omega F_1\lhd G}(x_1, Plus_{\mathscr{P}_\omega F_1\lhd G}(x_2, x_3))) & \\
= & \mathscr{P}_\omega F_1([-])(x_1 \cup (x_2 \cup x_3)) & \text{(def. } Plus\text{)} \\
= & \mathscr{P}_\omega F_1([-])((x_1 \cup x_2) \cup x_3) & \\
= & \mathscr{P}_\omega F_1([-])(Plus_{\mathscr{P}_\omega F_1\lhd G}(Plus_{\mathscr{P}_\omega F_1\lhd G}(x_1, x_2), x_3)) & \text{(def. } Plus\text{)}
\end{array}
$$

It is interesting to remark that in the last but one step we use the fact that, for any set X, $(\mathscr{P}_\omega X, \cup, \{\})$ is a join-semilattice (hence, $x_1 \cup (x_2 \cup x_3) = (x_1 \cup x_2) \cup x_3$). In fact, due to this fact, in the case $F = \mathscr{P}_\omega F_1$ the induction hypothesis will never be used. □

Thus, we have a well-defined function

$$\overline{\delta}_{F\lhd G} : Exp_{F\lhd G}/_{\equiv_{ACIE}} \to F(Exp_G/_{\equiv_{ACIE}})$$

such that $\overline{\delta}_{F\lhd G}([\varepsilon]_{ACIE}) = (F[-]_{ACIE})(\delta_{F\lhd G}(\varepsilon))$.

We are now ready to state and prove the second half of Kleene's theorem.

3.14. **Theorem.** *Let G be a non-deterministic functor. For every $\varepsilon \in Exp_G$, $\Delta_G(\varepsilon) = (S, g)$ is such that S is finite and there exists $s \in S$ with $\varepsilon \sim s$.*

Proof. For every $\varepsilon \in Exp_G$, we set $\Delta_G(\varepsilon) = \langle [\varepsilon]_{ACIE}\rangle$ (recall that $\langle s\rangle$ denotes the subcoalgebra generated by s). First note that, by Lemma 3.13, the map $[-]_{ACIE}$ is a homomorphism and thus $\varepsilon \sim [\varepsilon]_{ACIE}$. We prove, for every $\varepsilon \in Exp_G$, that the subcoalgebra $\langle [\varepsilon]_{ACIE}\rangle$ is finite. Again, in order to improve readability, in this proof we will use $[-]$ to denote $[-]_{ACIE}$.

More precisely, we prove, for all $\varepsilon \in Exp_{F\lhd G}$ that

$$\langle [\varepsilon]\rangle \subseteq \{[\varepsilon_1 \oplus \ldots \oplus \varepsilon_k] \mid \varepsilon_1, \ldots, \varepsilon_k \in cl(\varepsilon) \text{ all distinct}\} \tag{3.3}$$

by double induction on $N(\varepsilon)$ (Definition 3.9) and on the height of proofs for typing ε. Here, $cl(\varepsilon)$ denotes the smallest set containing all subformulas of ε and the unfoldings of μ (sub)formulas. For this proof it is not relevant the precise definition of cl, but only the fact that, for every ε, the set $cl(\varepsilon)$ is finite (note that because ε is guarded, the number of unfoldings is finite) and has the property: $\varepsilon' \in cl(\varepsilon) \Rightarrow cl(\varepsilon') \subseteq cl(\varepsilon)$.

$\boxed{N(\varepsilon) = 0}$

We have to prove equation (3.3) for all ε except $\varepsilon = \mu x.\varepsilon_1$. For $\varepsilon = \emptyset$ the result follows trivially, since $\langle [\emptyset]\rangle = \{[\emptyset]\}$. For any $\varepsilon \in \{l\langle\varepsilon_1\rangle, r\langle\varepsilon_1\rangle, l[\varepsilon_1], r[\varepsilon_1], a(\varepsilon_1), \{\varepsilon_1\}\}$ the result follows easily by induction (on the height of proofs for typing ε) because $\langle [\varepsilon]\rangle \subseteq \langle [\varepsilon_1]\rangle \cup \{[\varepsilon]\}$. The most interesting case is when $\varepsilon = \varepsilon_1 \oplus \varepsilon_2$. We have

$$\langle [\varepsilon_1 \oplus \varepsilon_2]\rangle \subseteq \{[\varepsilon_1' \oplus \varepsilon_2'] \mid \varepsilon_1' \in \langle [\varepsilon_1]\rangle \text{ and } \varepsilon_2' \in \langle [\varepsilon_2]\rangle\} \cup \{[\varepsilon_1 \oplus \varepsilon_2]\}$$

and the result follows because

$$\varepsilon_1' \oplus \varepsilon_2' \equiv_{ACIE} \varepsilon_1'' \oplus \ldots \oplus \varepsilon_k'' \text{ with all } \varepsilon_1'', \ldots, \varepsilon_k'' \in cl(\varepsilon_1' \oplus \varepsilon_2') \text{ distinct}$$

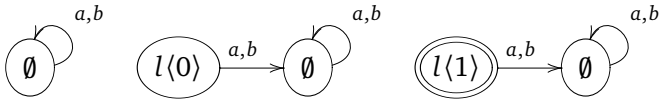
and $cl(\varepsilon_1' \oplus \varepsilon_2') \subseteq cl(\varepsilon_1 \oplus \varepsilon_2)$.

$\boxed{N(\varepsilon) \leq k+1}$

We need to prove the desired inclusion for every expression. Everything follows as in the base case and thus we prove just the inclusion for $\varepsilon = \mu x.\varepsilon_1$. We have $\langle [\mu x.\varepsilon_1] \rangle = \langle [\varepsilon_1[\mu x.\varepsilon_1/x]] \rangle$ and $cl(\varepsilon_1[\mu x.\varepsilon_1/x]) \subseteq cl(\mu x.\varepsilon_1)$, which yields the intended result. □

3.3.1. *Examples.* In this subsection we will illustrate the construction described in the proof of Theorem 3.14: given an expression $\varepsilon \in Exp_G$ we construct a G-coalgebra (S, g) such that there is $s \in S$ with $s \sim \varepsilon$. For simplicity, we will consider deterministic and partial automata expressions over $A = \{a, b\}$.

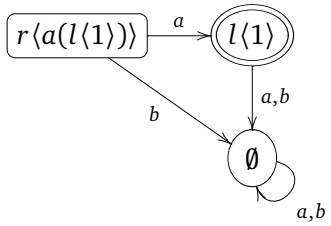
Let us start by showing the synthesised automata for the most simple deterministic expressions – $\emptyset$, $l\langle 0 \rangle$ and $l\langle 1 \rangle$.



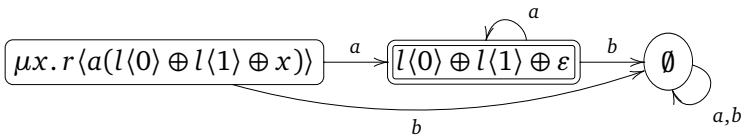
It is interesting to make the parallel with the traditional regular expressions and remark that the first two automata recognize the empty language {} and the last the language $\{\epsilon\}$ containing only the empty word.

An important remark is that the automata generated are not minimal (for instance, the automata $l\langle 0 \rangle$ and $\emptyset$ are bisimilar). Our goal has been to generate a finite automaton from an expression. From this the minimal automaton can always be obtained by identifying bisimilar states.

The following automaton, generated from the expression $r\langle a(l\langle 1 \rangle) \rangle$, recognizes the language {a},

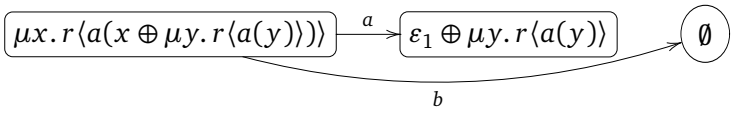


For an example of an expression containing fixed-points, consider $\varepsilon = \mu x.\, r\langle a(l\langle 0 \rangle \oplus l\langle 1 \rangle \oplus x) \rangle$. One can easily compute the synthesised automaton:



and observe that it recognizes the language aa^*. Here, the role of the join-semilattice structure is also visible: $l\langle 0 \rangle \oplus l\langle 1 \rangle \oplus \varepsilon$ specifies that this state is supposed to be non-final ($l\langle 0 \rangle$) and final ($l\langle 1 \rangle$). The conflict of these two specifications is solved, when they are combined with $\oplus$, using the join-semilattice structure: because $1 \vee 0 = 1$ the state is set to be final.

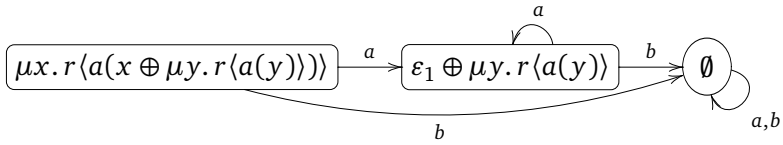
As a last example of deterministic expressions consider $\varepsilon_1 = \mu x.\, r\langle a(x \oplus \mu y.r\langle a(y)\rangle)\rangle$. Applying δ_D to ε_1 one gets the following (partial) automaton:



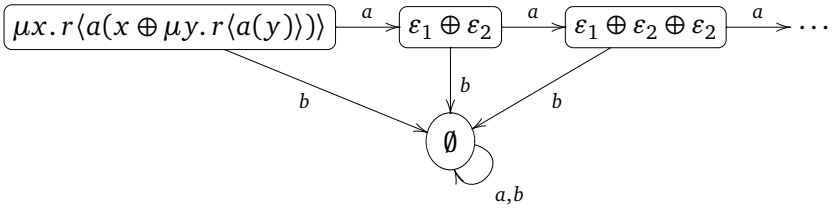
Calculating $\delta_D(\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle)(a)$ yields

$$\delta_D(\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle)(a) \quad = \quad \langle 0, \varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle\rangle$$

Now, note that the expression $\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle \oplus \mu y.\, r\langle a(y)\rangle$ is in the same equivalence class as $\varepsilon_1 \oplus \mu y.\, r\langle a(y)\rangle$, which is a state that already exists. As we saw in the beginning of Section 3.1, by only applying δ_D, without *ACI*, one would always generate syntactically different states which instead of the automata computed now:

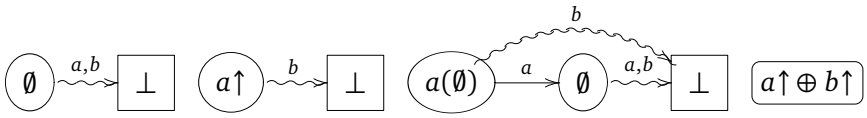


would yield the following infinite automaton (with $\varepsilon_2 = \mu y.\, r\langle a(y)\rangle$):



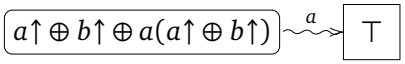
Let us next see a few examples of synthesis for partial automata expressions, where we will illustrate the role of $\bot$ and $\top$. In the graphical representation of a partial automata (S, p), we draw $(s) \overset{a}{\rightsquigarrow} \boxed{g(s)(a)}$ whenever $g(s)(a) \in \{\bot, \top\}$. Note however that $\bot \notin S$ and $\top \notin S$ and thus will have no defined transitions.

As before, let us first present the corresponding automata for simple expressions – $\emptyset$, $a\!\uparrow$, $a(\emptyset)$ and $a\!\uparrow \oplus b\!\uparrow$.



Note how $\bot$ is used to encode underspecification, working as a kind of deadlock state. In the first three expressions the behaviour for one or both of the inputs is missing, whereas in the last expression the specification is complete.

The element $\top$ is used to deal with inconsistent specifications. For instance, consider the expression $a\!\uparrow \oplus b\!\uparrow \oplus a(a\!\uparrow \oplus b\!\uparrow)$. All inputs are specified, but note that at the outermost level input a appears in two different sub-expressions – $a\!\uparrow$ and $a(a\!\uparrow \oplus b\!\uparrow)$ – specifying at the same time that input a leads to successful termination and that it leads to a state where $a\!\uparrow \oplus b\!\uparrow$ holds, which is contradictory, giving rise to the following automaton.



4. A SOUND AND COMPLETE AXIOMATIZATION

In the previous section, we have shown how to derive from the type of a system, given by a functor G, a language Exp_G that allows for specification of G-behaviours. Analogously to Kleene's theorem, we have proved the coincidence between the behaviours denoted by Exp_G and locally finite G-coalgebras. In this section, we will show how to provide Exp_G with a sound and complete axiomatization. Again, the functor G will serve as a main guide for the definition. The defined axiomatization is closely related to Kleene algebra (the set of expressions has a join semilattice structure) and to the axiomatization provided by Milner for CCS (uniqueness of fixed-points will be required). When instantiating the definition below to concrete functors one will recover known axiomatizations, such as the one for CCS mentioned above or the one for labelled transition systems (with explicit termination) presented in [1]. The latter will be discussed in detail in Section 4.2.

We now introduce an equational system for expressions of type $F \lhd G$. We define the relation $\equiv \subseteq Exp_{F\lhd G} \times Exp_{F\lhd G}$, written infix, as the least reflexive and transitive relation containing the following identities:

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

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

(2) μ is the unique fixed-point.

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

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

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

(4) $\equiv$ is a congruence.

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

(5) α-equivalence

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

It is important to remark that in the third group of rules there does not exist any rule applicable to expressions of type $\mathscr{P}_\omega F$.

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

Applying $\langle\!\langle - \rangle\!\rangle$ (as defined in the proof of Theorem 3.12) one can easily compute the expressions corresponding to s_1 and s'_1:

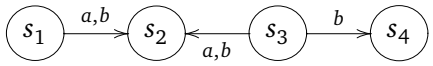
$$\begin{aligned}\varepsilon_1 &= \langle\!\langle s_1 \rangle\!\rangle = \mu x_1.l\langle 0\rangle \oplus r\langle a(\{x_1\})\rangle \\ \varepsilon'_1 &= \langle\!\langle s_2 \rangle\!\rangle = \mu y_1.l\langle 0\rangle \oplus r\langle a(\{\mu y_2.l\langle 0\rangle \oplus r\langle a(\{\mu y_1.l\langle 0\rangle \oplus r\langle a(\{y_2\})\rangle\})\rangle\})\rangle\end{aligned}$$

We now prove that $\varepsilon'_1 \equiv \varepsilon_1$. In the following calculations let $\varepsilon = \mu x_1.r\langle a(\{x_1\})\rangle$.

$$\begin{array}{lll} & \varepsilon'_1 \equiv \varepsilon_1 & \\ \iff & r\langle a(\{\mu y_2.r\langle a(\{r\langle a(\{y_2\})\rangle\})\rangle\})\rangle \equiv \varepsilon & ((B-\emptyset),(\times-\emptyset-L),(FP) \text{ and } (Empty)) \\ \iff & \mu y_2.r\langle a(\{r\langle a(\{y_2\})\rangle\})\rangle \equiv \varepsilon & ((FP) \text{ on } \varepsilon \text{ and } (Cong) \text{ twice}) \\ \Leftarrow & r\langle a(\{r\langle a(\{\varepsilon\})\rangle\})\rangle \equiv \varepsilon & (\text{uniqueness of fixed-points}) \\ \iff & r\langle a(\{\varepsilon\})\rangle \equiv \varepsilon & (\text{fixed-point axiom}) \\ \iff & \varepsilon \equiv \varepsilon & (\text{fixed-point axiom}) \end{array}$$

Note that the (*Cong*) rule was used in almost every step.

For another example, consider the non-deterministic automata over the alphabet $A = \{a, b\}$:



Using the definition of $\langle\!\langle - \rangle\!\rangle$ one can compute the following expressions for s_1, s_2, s_3 and s_4:

$$\begin{aligned}\varepsilon_1 &= \langle\!\langle s_1 \rangle\!\rangle = \mu x_1.l\langle 0\rangle \oplus r\langle a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\})\rangle \\ \varepsilon_2 &= \langle\!\langle s_2 \rangle\!\rangle = \mu x_2.l\langle 0\rangle \oplus \emptyset \\ \varepsilon_3 &= \langle\!\langle s_3 \rangle\!\rangle = \mu x_3.l\langle 0\rangle \oplus r\langle a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\} \oplus \{\varepsilon_4\})\rangle \\ \varepsilon_4 &= \langle\!\langle s_4 \rangle\!\rangle = \mu x_4.l\langle 0\rangle \oplus \emptyset\end{aligned}$$

For ε_2 we calculate:

$$\begin{array}{lll} \varepsilon_2 & \equiv & l\langle 0\rangle \oplus \emptyset \quad (FP) \\ & \equiv & l\langle \emptyset\rangle \quad (Empty) \text{ and } (B-\emptyset) \\ & \equiv & \emptyset \quad (\times-\emptyset-L) \end{array}$$

Similarly, one has that $\varepsilon_4 \equiv \emptyset$. Now, we prove $\varepsilon_1 \equiv \varepsilon_3$:

$$\begin{array}{lll} & \varepsilon_1 \equiv \varepsilon_3 & \\ \iff & l\langle 0\rangle \oplus r\langle a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\})\rangle \equiv l\langle 0\rangle \oplus r\langle a(\{\varepsilon_2\}) \oplus b(\{\varepsilon_2\} \oplus \{\varepsilon_4\})\rangle & (FP) \\ \iff & l\langle 0\rangle \oplus r\langle a(\{\emptyset\}) \oplus b(\{\emptyset\})\rangle \equiv l\langle 0\rangle \oplus r\langle a(\{\emptyset\}) \oplus b(\{\emptyset\} \oplus \{\emptyset\})\rangle & (\varepsilon_2 \equiv \emptyset \equiv \varepsilon_4) \\ \iff & l\langle 0\rangle \oplus r\langle a(\{\emptyset\}) \oplus b(\{\emptyset\})\rangle \equiv l\langle 0\rangle \oplus r\langle a(\{\emptyset\}) \oplus b(\{\emptyset\})\rangle & (Idempotency) \end{array}$$

♠

The equivalence relation $\equiv$ gives rise to the equivalence map $[-]\colon Exp_{F\lhd G} \to Exp_{F\lhd G}/_{\equiv}$, defined by $[\varepsilon] = \{\varepsilon' \mid \varepsilon \equiv \varepsilon'\}$. The following diagram summarizes the maps we have defined so far:

$$\begin{array}{ccc} Exp_{F\lhd G} & \xrightarrow{[-]} & Exp_{F\lhd G}/_{\equiv} \\ {\scriptstyle \delta_{F\lhd G}}\downarrow & & \\ F(Exp_{F\lhd G}) & \xrightarrow[F([-])]{} & F(Exp_G/_{\equiv}) \end{array}$$

In order to complete the diagram, we next prove that $\equiv$ is contained in the kernel of $F([-])\circ \delta_{F\lhd G}$. This will guarantee the existence of a well-defined function $\partial_{F\lhd G}\colon Exp_{F\lhd G}/_{\equiv} \to F(Exp_G/_{\equiv})$ which, when $F = G$, provides Exp_G with a coalgebraic structure $\partial_G\colon Exp_G \to G(Exp_G)$ (as before, we write ∂_G to abbreviate $\partial_{G\lhd G}$) and which makes $[-]$ a homomorphism of coalgebras.

4.2. **Lemma.** *Let G and F be non-deterministic functors, with $F \lhd G$. For all $\varepsilon_1, \varepsilon_2 \in Exp_{F\lhd G}$ with $\varepsilon_1 \equiv \varepsilon_2$,*

$$F([-]) \circ \delta_{F\lhd G}(\varepsilon_1) = F([-]) \circ \delta_{F\lhd G}(\varepsilon_2)$$

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

First, let us consider derivations of length 1. We need to prove the result for all the axioms in items 1. and 3. plus the axioms *FP* and $(\alpha - equiv)$.

For the axioms in 1. the result follows by Lemma 3.13. The axiom *FP* follows trivially because of the definition of δ_G, since $\delta_G(\mu x.\gamma) = \delta_G(\gamma[\mu x.\gamma/x])$ and thus $G([-])\circ\delta_G(\mu x.\gamma) = G([-]) \circ \delta_G(\gamma[\mu x.\gamma/x])$.

For the axiom $(\alpha - equiv)$ we use the (*Cong*) rule, which is proved below:

$$\begin{array}{lll} & G([-]) \circ \delta_G(\mu x.\gamma) & \\ = & G([-]) \circ \delta_G(\gamma[\mu x.\gamma/x]) & \text{(def. of } \delta_G) \\ = & G([-]) \circ \delta_G(\gamma[\mu y.\gamma[y/x]/x]) & \text{(by (}Cong\text{))} \\ = & G([-]) \circ \delta_G(\gamma[y/x][\mu y.\gamma[y/x]/y]) & (A[B[y/x]/x] = A[y/x][B[y/x]/y],\ y \text{ not free in } \gamma) \\ = & G([-]) \circ \delta_G(\mu y.\gamma[y/x])) & \text{(def. of } G([-]) \circ \delta_G) \end{array}$$

Let us now show the proof for some of the axioms in 3.. The omitted cases are similar. We show for each axiom $\varepsilon_1 \equiv \varepsilon_2$ that $\delta_{F\lhd G}(\varepsilon_1) = \delta_{F\lhd G}(\varepsilon_2)$.

$\boxed{\bot_B \equiv \emptyset}$

$\delta_{B\lhd G}(\bot_B) = \bot_B = \delta_{B\lhd G}(\emptyset)$

$\boxed{b_1 \oplus b_2 \equiv b_1 \vee_B b_2}$

$\delta_{B\lhd G}(b_1 \vee_B b_2) = b_1 \vee_B b_2 = \delta_{B\lhd G}(b_1 \oplus b_2)$

$\boxed{l(\emptyset) \equiv \emptyset}$

$\delta_{F_1\times F_2\lhd G}(l(\emptyset)) = \langle Empty_{F_1\lhd G}, Empty_{F_2\lhd G}\rangle = \delta_{F_1\times F_2\lhd G}(\emptyset)$

$\boxed{l(\varepsilon_1 \oplus \varepsilon_2) \equiv l(\varepsilon_1) \oplus l(\varepsilon_2)}$

$$\begin{array}{ll} & \delta_{F_1\times F_2\lhd G}(l(\varepsilon_1 \oplus \varepsilon_2)) \\ = & \langle \delta_{F_1\lhd G}(\varepsilon_1 \oplus \varepsilon_2), Empty_{F_2\lhd G}\rangle) \\ = & \langle Plus_{F_1\lhd G}(\delta_{F_1\lhd G}(\varepsilon_1), \delta_{F_1\lhd G}(\varepsilon_2)), Plus_{F_2\lhd G}(Empty_{F_2\lhd G}, Empty_{F_2\lhd G})\rangle) \\ = & Plus_{F_1\times F_2}(\langle \delta_{F_1\lhd G}(\varepsilon_1), Empty_{F_2\lhd G}\rangle, \langle \delta_{F_1\lhd G}(\varepsilon_2), Empty_{F_2\lhd G}\rangle \\ = & \delta_{F_1\times F_2\lhd G}(l(\varepsilon_1) \oplus l(\varepsilon_2))) \end{array}$$

$\boxed{l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2]}$

$$\begin{array}{ll} & \delta_{F_1+F_2\lhd G}(l[\varepsilon_1 \oplus \varepsilon_2]) \\ = & \kappa_1(\delta_{F_1\lhd G}(\varepsilon_1 \oplus \varepsilon_2)) \\ = & Plus_{F_1+F_2}(\kappa_1(\delta_{F_1\lhd G}(\varepsilon_1)), \kappa_1(\delta_{F_1\lhd G}(\varepsilon_2)) \\ = & \delta_{F_1+F_2\lhd G}(l[\varepsilon_1] \oplus l[\varepsilon_2]) \end{array}$$

$\boxed{l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset]}$

$$\begin{array}{ll} & \delta_{F_1+F_2\lhd G}(l[\varepsilon_1] \oplus r[\varepsilon_2]) \\ = & Plus_{F_1+F_2}(\kappa_1(\delta_{F_1\lhd G}(\varepsilon_1)), \kappa_2(\delta_{F_2\lhd G}(\varepsilon_2))) \\ = & \top \\ = & Plus_{F_1+F_2}(\kappa_1(\delta_{F_1\lhd G}(\emptyset)), \kappa_2(\delta_{F_2\lhd G}(\emptyset))) \\ = & \delta_{F_1+F_2\lhd G}(l[\emptyset] \oplus r[\emptyset]) \end{array}$$

It is interesting to remark here that if we would have the axioms $l[\emptyset] \equiv \emptyset$ and $r[\emptyset] \equiv \emptyset$ in the axiomatizatiation presented above this theorem would not hold.

$$\delta_{F_1+F_2\triangleleft G}(l[\emptyset]) = \kappa_1([\bot]) \neq \bot = \delta_{F_1+F_2\triangleleft G}(\emptyset)$$
$$\delta_{F_1+F_2\triangleleft G}(r[\emptyset]) = \kappa_2([\bot]) \neq \bot = \delta_{F_1+F_2\triangleleft G}(\emptyset)$$

Derivations with length $k>1$ can be obtained by two rules: (*Unique*) or (*Cong*). For the first (which uses the second), suppose that we have derived $\mu x.\gamma \equiv \varepsilon$ and that we have already proved $\gamma[\varepsilon/x] \equiv \varepsilon$. Then, we have:

$$\begin{array}{rcll} G([-])\circ\delta_G(\mu x.\gamma) & = & G([-])\circ\delta_G(\gamma[\mu x.\gamma/x]) & \text{(def. } \delta_G\text{)} \\ & = & G([-])\circ\delta_G(\gamma[\varepsilon/x]) & \text{(by (}\textit{Cong}\text{))} \\ & = & G([-])\circ\delta_G(\varepsilon) & \text{(induction hypothesis)} \end{array}$$

For (*Cong*), suppose that we have derived $\varepsilon[\varepsilon_1/x] \equiv \varepsilon[\varepsilon_2/x_2]$ and that we have already derived $\varepsilon_1 \equiv \varepsilon_2$, which gives us, as induction hypothesis, the equality

$$(F[-])(\delta_{F\triangleleft G}(\varepsilon_1)) = (F[-])(\delta_{F\triangleleft G}(\varepsilon_2)) \tag{4.1}$$

Then, assuming (4.1), we prove the equality by induction on the height of proofs for typing ε. The most interesting case is $\varepsilon = x$ (and thus $\varepsilon{:}G \triangleleft G$). All the other cases follow easily by induction and thus we omit them here.

$$\begin{array}{rcll} (G[-])(\delta_G(x[\varepsilon_1/x]) & = & (G[-])(\delta_G(\varepsilon_1)) & \\ & = & (G[-])(\delta_G(\varepsilon_2)) & (4.1) \\ & = & (G[-])(\delta_G(x[\varepsilon_2/x])) & \end{array}$$

□

Thus, we have now a well-defined function $\partial_{F\triangleleft G}\colon Exp_{F\triangleleft G}/_{\equiv} \to F(Exp_G/_{\equiv})$, which makes the diagram above commute, that is $\partial_{F\triangleleft G}([\varepsilon]) = (F[-])\circ\delta_{F\triangleleft G}(\varepsilon)$. This provides the set $Exp_G/_{\equiv}$ with a coalgebraic structure $\delta_G\colon Exp_G \to G(Exp_G)$ which makes $[-]$ a homomorphism between the coalgebras (Exp_G, δ_G) and $(Exp_G/_{\equiv}, \partial_G)$.

4.1. **Soundness and Completeness.** We now show that the axiomatization introduced in the previous section is sound and complete.

Soundness is a direct consequence of the fact that the equivalence map $[-]$ is a coalgebra homomorphism.

4.3. **Theorem** (Soundness)**.** *Let G be a non-deterministic functor. For all $\varepsilon_1, \varepsilon_2 \in Exp_G$,*

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

Proof. Let G be a non-deterministic functor, let $\varepsilon_1, \varepsilon_2 \in Exp_G$ and suppose that $\varepsilon_1 \equiv \varepsilon_2$. Then, $[\varepsilon_1] = [\varepsilon_2]$ and, thus

$$\mathbf{beh}_{Exp_G/_{\equiv}}([\varepsilon_1]) = \mathbf{beh}_{Exp_G/_{\equiv}}([\varepsilon_2])$$

where $\mathbf{beh}_S$ denotes, for any G-coalgebra (S, g), the unique map into the final coalgebra. The uniqueness of the map into the final coalgebra and the fact that $[-]$ is a coalgebra homomorphism implies that $\mathbf{beh}_{Exp_G/_{\equiv}} \circ [-] = \mathbf{beh}_{Exp_G}$ which then yields

$$\mathbf{beh}_{Exp_G}(\varepsilon_1) = \mathbf{beh}_{Exp_G}(\varepsilon_2)$$

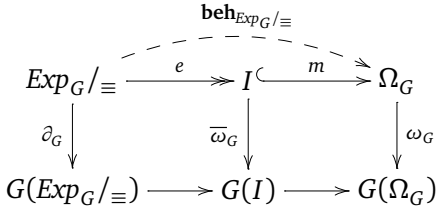
Since in the final coalgebra only the bisimilar elements are identified, $\varepsilon_1 \sim \varepsilon_2$ follows. □

For completeness a bit more of work is required. Let us explain upfront the key ingredients of the proof. The goal is to prove that $\varepsilon_1 \sim \varepsilon_2 \Rightarrow \varepsilon_1 \equiv \varepsilon_2$. First, note that we have

$$\varepsilon_1 \sim \varepsilon_2 \Leftrightarrow \mathbf{beh}_{Exp_G}(\varepsilon_1) = \mathbf{beh}_{Exp_G}(\varepsilon_2) \Leftrightarrow \mathbf{beh}_{Exp_G/_\equiv}([\varepsilon_1]) = \mathbf{beh}_{Exp_G/_\equiv}([\varepsilon_2]) \tag{4.2}$$

We then prove that $\mathbf{beh}_{Exp_G/_\equiv}$ is injective, which is a sufficient condition to guarantee that $\varepsilon_1 \equiv \varepsilon_2$ (since it implies, together with (4.2), that $[\varepsilon_1] = [\varepsilon_2]$).

We proceed as follows. First, we factorize $\mathbf{beh}_{Exp_G/_\equiv}$ into an epimorphism and a monomorphism [28, Theorem 7.1] as shown in the following diagram (where $I = \mathbf{beh}_{Exp_G/_\equiv}(Exp_G/_\equiv)$):



Then, we prove that (1) $(Exp_G/_\equiv, \partial_G)$ is a locally finite coalgebra (Lemma 4.4) and (2) both coalgebras $(Exp_G/_\equiv, \partial_G)$ and $(I, \overline{\omega}_G)$ are final in the category of locally finite G-coalgebras (Lemmas 4.7 and 4.8, respectively). Since final coalgebras are unique up to isomorphism, it follows that $e\colon Exp_G/_\equiv \to I$ is in fact an isomorphism and therefore $\mathbf{beh}_{Exp_G/_\equiv}$ 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).

We now proceed with presenting and proving the extra lemmas needed in order to prove completeness. We start by showing that the coalgebra $(Exp_G/_\equiv, \partial_G)$ is locally finite (note that this implies that $(I, \overline{\omega}_G)$ is also locally finite) and that ∂_G is an isomorphism.

4.4. **Lemma.** *The coalgebra $(Exp_G/_\equiv, \partial_G)$ is a locally finite coalgebra. Moreover, ∂_G is an isomorphism.*

Proof. Locally finiteness is a direct consequence of the generalized Kleene's theorem (Theorem 3.14). In the proof of Theorem 3.14 we showed that, given $\varepsilon \in Exp_G$, the subcoalgebra $\langle [\varepsilon]_{ACIE} \rangle$ is finite. Thus, the subcoalgebra $\langle [\varepsilon] \rangle$ is also finite (since $Exp_G/_\equiv$ is a quotient of $Exp_G/_{\equiv_{ACIE}}$).

To see that ∂_G is an isomorphism, first define, for every $F \lhd G$,

$$\partial^{-1}_{F \lhd G}(s') = [\overline{\gamma}^F_{s'}] \tag{4.3}$$

where $\overline{\gamma}^F_{s'}$ is defined, for $F \neq Id$, as $\gamma^F_{s'}$ in the proof of Theorem 3.12, and for $F = Id$ as $\overline{\gamma}^{Id}_{[\varepsilon]} = \varepsilon$. Then, we prove that $\partial^{-1}_{F \lhd G}$ has indeed the properties ① $\partial^{-1}_{F \lhd G} \circ \partial_{F \lhd G} = id_{Exp_{F \lhd G}/_\equiv}$ and ② $\partial_{F \lhd G} \circ \partial^{-1}_{F \lhd G} = id_{F(Exp_{F \lhd G}/_\equiv)}$. Instantiating $F = G$ one derives that δ_G is an isomorphism. It is enough to prove for ① that $\overline{\gamma}^F_{\partial_{F \lhd G}([\varepsilon])} \equiv \varepsilon$ and for ② that $\partial_{F \lhd G}([\overline{\gamma}^F_{s'}]) = s'$. We illustrate a couple of cases. The omitted ones are similar.

① By double induction on $N(\varepsilon)$ and the height of proofs for typing ε.

$\boxed{N(\varepsilon)=0}$ $\quad \overline{\gamma}^{Id}_{\partial_{Id\lhd G}([\varepsilon])} = \varepsilon$

$$\overline{\gamma}^{F_1\times F_2}_{\partial_{F_1\times F_2\lhd G}([r\langle\varepsilon\rangle])} = l\langle\overline{\gamma}^{F_1}_{\partial_{F_1\lhd G}(\emptyset)}\rangle \oplus r\langle\overline{\gamma}^{F_2}_{\partial_{F_2\lhd G}(\varepsilon)}\rangle \overset{IH}{\equiv} l\langle\emptyset\rangle \oplus r\langle\varepsilon\rangle \equiv r\langle\varepsilon\rangle$$

$\boxed{N(\varepsilon)\leq k+1}$ $\quad \overline{\gamma}^{F}_{\partial_{F\lhd G}([\mu x.\varepsilon])} = \overline{\gamma}^{F}_{\partial_{F\lhd G}([\varepsilon[\mu x.\varepsilon/x]])} \overset{IH}{\equiv} \varepsilon[\mu x.\varepsilon/x] \equiv \mu x.\varepsilon$

It is interesting to remark that the cases $\varepsilon=\emptyset$ and $\varepsilon=\varepsilon_1\oplus\varepsilon_2$ require an extra proof (by induction on F). More precisely, one needs to prove that ⓐ $\overline{\gamma}^{F}_{F[-](Empty_{F\lhd G})} \equiv \emptyset$ and ⓑ $\overline{\gamma}^{F}_{F[-](Plus_{F\lhd G}(x_1,x_2))} \equiv \overline{\gamma}^{F}_{F[-](x_1)} \oplus \overline{\gamma}^{F}_{F[-](x_2)}$. It is an easy proof by induction. We illustrate here only the cases $F=Id$, $F=B$ and $F=F_1\times F_2$.

ⓐ $\overline{\gamma}^{Id}_{[\emptyset]} = \emptyset$

$\overline{\gamma}^{B}_{[\bot_B]} = \bot_B \equiv \emptyset$

$$\overline{\gamma}^{F_1\times F_2}_{\langle F_1[-](Empty_{F_1\lhd G}),F_2[-](Empty_{F_2\lhd G})\rangle} = l\langle\overline{\gamma}^{F_1}_{F_1[-](Empty_{F_1\lhd G})}\rangle \oplus r\langle\overline{\gamma}^{F_2}_{F_2[-](Empty_{F_2\lhd G})}\rangle \overset{IH}{\equiv} l\langle\emptyset\rangle \oplus r\langle\emptyset\rangle \equiv \emptyset$$

ⓑ $\overline{\gamma}^{Id}_{[x_1\oplus x_2]} = x_1\oplus x_2 = \overline{\gamma}^{Id}_{[x_1]} \oplus \overline{\gamma}^{Id}_{[x_2]}$

$\overline{\gamma}^{B}_{[x_1\vee_B x_2]} = x_1\vee_B x_2 \equiv x_1\oplus x_2 = \overline{\gamma}^{B}_{[x_1]} \oplus \overline{\gamma}^{B}_{[x_2]}$

$$\overline{\gamma}^{F_1\times F_2}_{F_1\times F_2[-](Plus_{F_1\times F_2\lhd G}(\langle u_1,v_1\rangle,\langle u_2,v_2\rangle))} = \overline{\gamma}^{F_1\times F_2}_{\langle Plus_{F_1}(u_1,v_1),Plus_{F_2}(u_2,v_2)\rangle} = l\langle\overline{\gamma}^{F_1}_{Plus_{F_1}(u_1,v_1)}\rangle \oplus r\langle\overline{\gamma}^{F_2}_{Plus_{F_2}(u_2,v_2)}\rangle$$

$$\overset{IH}{\equiv} l\langle\gamma^{F_1}_{u_1}\oplus\gamma^{F_1}_{v_1}\rangle \oplus r\langle\gamma^{F_2}_{u_2}\oplus\gamma^{F_2}_{v_2}\rangle \equiv (l\langle\gamma^{F_1}_{u_1}\rangle \oplus r\langle\gamma^{F_2}_{u_2}\rangle) \oplus (l\langle\gamma^{F_1}_{v_1}\rangle \oplus r\langle\gamma^{F_2}_{v_2}\rangle) = \overline{\gamma}^{F_1\times F_2}_{\langle u_1,u_2\rangle} \oplus \overline{\gamma}^{F_1\times F_2}_{\langle v_1,v_2\rangle}$$

② By induction on the structure of F.

$$\partial_{F_1+F_2\lhd G}([\overline{\gamma}^{F_1+F_2}_{s'}]) = \begin{cases} \partial_{F_1+F_2\lhd G}([l[\overline{\gamma}^{F_1}_{s}]]) = \kappa_1(\partial_{F_1\lhd G}([\overline{\gamma}^{F_1}_{s}])) & s' = \kappa_1(s) \\ \partial_{F_1+F_2\lhd G}([r[\overline{\gamma}^{F_2}_{s}]]) = \kappa_2(\partial_{F_2\lhd G}([\overline{\gamma}^{F_2}_{s}])) & s' = \kappa_2(s) \\ \partial_{F_1+F_2\lhd G}([\emptyset]) = \bot & s' = \bot \\ \partial_{F_1+F_2\lhd G}([l[\emptyset]\oplus r[\emptyset]]) = \top & s' = \top \end{cases}$$

$$\overset{IH}{=} s'$$

$$\partial_{\mathscr{P}_\omega F\lhd G}([\overline{\gamma}^{\mathscr{P}_\omega F}_{S}]) = \begin{cases} \partial_{\mathscr{P}_\omega F\lhd G}([\emptyset]) = \{\} & S=\{\} \\ \partial_{\mathscr{P}_\omega F\lhd G}([\bigoplus_{s\in S}\overline{\gamma}^{F_1}_{s}]) = \{\partial_{F\lhd G}([\overline{\gamma}^{F_1}_{s}]) \mid s\in S\} & \text{otherwise} \end{cases}$$

$$\overset{IH}{=} S$$

□

We now prove the analogue of the following useful and intuitive equality on regular expressions. Given a deterministic automaton $\langle o,t\rangle\colon S\to 2\times S^A$ and a state $s\in S$, the associated regular expression r_s can be written as

$$r_s = o(s) + \sum_{a\in A} a\cdot r_{t(s)(a)} \tag{4.4}$$

using the axioms of Kleene algebra [11, Theorem 4.4].

4.5. **Lemma.** *Let (S,g) be a locally finite G-coalgebra, with $G\neq Id$, and let $s\in S$, with $\langle s\rangle = \{s_1,\dots,s_n\}$ (where $s_1=s$). Then:*

$$\langle\!\langle s_i\rangle\!\rangle \equiv \gamma^{G}_{g(s_i)}\{\langle\!\langle s_1\rangle\!\rangle/x_1\}\dots\{\langle\!\langle s_n\rangle\!\rangle/x_n\} \tag{4.5}$$

Proof. Let A_i^k, where i and k range from 1 to n, be the terms defined as in the proof of Theorem 3.12. Recall that $\langle\!\langle s_i \rangle\!\rangle = A_i^n$. We calculate:

$$
\begin{array}{lll}
 & \langle\!\langle s_i \rangle\!\rangle & \\
= & A_i^n & \\
= & (\mu x_i.\gamma^G_{g(s_i)})\{A_1^0/x_1\}\dots\{A_n^{n-1}/x_n\} & \\
= & \mu x_i.(\gamma^G_{g(s_i)}\{A_1^0/x_1\}\dots\{A_{i-1}^{i-2}/x_{i-2}\}\{A_{i+1}^{i}/x_{i+1}\}\dots\{A_n^{n-1}/x_n\}) & \\
\equiv & \gamma^G_{g(s_i)}\{A_1^0/x_1\}\dots\{A_{i-1}^{i-2}/x_{i-2}\}\{A_{i+1}^{i}/x_{i+1}\}\dots\{A_n^{n-1}/x_n\}\{A_i^n/x_i\} & \text{(fixed-point axiom}^2\text{)} \\
= & \gamma^G_{g(s_i)}\{A_1^0/x_1\}\dots\{A_n^{n-1}/x_n\} & \text{(by 3.1)} \\
= & \gamma^G_{g(s_i)}\{A_1^0\{A_2^1/x_2\}\dots\{A_n^{n-1}/x_n\}/x_1\}\dots\{A_n^{n-1}/x_n\} & \text{(by 3.2)} \\
= & \gamma^G_{g(s_i)}\{A_1^n/x_1\}\{A_2^1/x_2\}\dots\{A_n^{n-1}/x_n\} & \text{(def. } A_1^n\text{)} \\
\vdots & & \text{(last 2 steps for } A_2^1,\dots,A_{n-1}^{n-2}\text{)} \\
= & \gamma^G_{g(s_i)}\{A_1^n/x_1\}\{A_2^n/x_2\}\dots\{A_n^n/x_n\} & (A_{n-1}^n = A_n^n) \\
= & \gamma^G_{g(s_i)}[A_1^n/x_1][A_2^n/x_2]\dots[A_n^n/x_n] & \text{(all } A_i^n \text{ are closed)}
\end{array}
$$

□

Instantiating (4.5) for $\langle o, t\rangle : S \to 2 \times S^A$, one can easily spot the similarity with equation (4.4) above:

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

Next, we prove that there exists a coalgebra homomorphism between any locally finite G- coalgebra (S, g) and $(Exp_G/_{\equiv}, \partial_G)$.

4.6. **LEMMA.** *Let (S, g) be a locally finite G-coalgebra. There exists a coalgebra homomorphism $\lceil - \rceil : S \to Exp_G/_{\equiv}$.*

Proof. We define $\lceil - \rceil = [-] \circ \langle\!\langle - \rangle\!\rangle$, where $\langle\!\langle - \rangle\!\rangle$ is as in the proof of Theorem 3.12, associating to a state s of a locally finite coalgebra an expression $\langle\!\langle s \rangle\!\rangle$ with $s \sim \langle\!\langle s \rangle\!\rangle$. To prove that $\lceil - \rceil$ is a homomorphism we need to verify that $(G\lceil - \rceil) \circ g = \partial_G \circ \lceil - \rceil$.

If $G = Id$, then $(G\lceil - \rceil) \circ g(s_i) = [\emptyset] = \partial_G(\lceil s_i \rceil)$. For $G \neq Id$ we calculate, using Lemma 4.5:

$$\partial_G \circ \lceil s_i \rceil = \partial_G([\gamma^G_{g(s_i)}[\langle\!\langle s_1 \rangle\!\rangle/x_1]\dots[\langle\!\langle s_n \rangle\!\rangle/x_n]])$$

and we then prove the more general equality, for $F \lhd G$ and $m \in F\langle s\rangle$:

$$\partial_{F\lhd G}([\gamma^F_m[\langle\!\langle s_1 \rangle\!\rangle/x_1]\dots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) = F\lceil - \rceil(m) \tag{4.6}$$

The intended equality then follows by taking $F = G$ and $m = g(s_i)$. Let us prove the equation (4.6) by induction on F.

$\boxed{F = Id}$ $m = s_j \in \langle s\rangle$

$$\partial_{Id\lhd G}([\gamma^{Id}_{s_j}[\langle\!\langle s_1 \rangle\!\rangle/x_1]\dots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) = [\langle\!\langle s_j \rangle\!\rangle] = \lceil s_j \rceil$$

$\boxed{F = B}$ $m = b \in B$

$$\partial_{B\lhd G}([\gamma^B_b[\langle\!\langle s_1 \rangle\!\rangle/x_1]\dots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) = [b] = B\lceil - \rceil(b)$$

[2]Note that the fixed point axiom can be formulated using syntactic replacement rather than substitution – $\gamma\{\mu x.\gamma/x\} \equiv \mu x.\gamma$ – since $\mu x.\gamma$ is a closed term.

$\boxed{F = F_1 \times F_2}$ $m = \langle m_1, m_2 \rangle \in F_1 \times F_2\langle s \rangle$

$$\begin{array}{lll} & \partial_{F_1\times F_2 \lhd G}([\gamma^{F_1\times F_2}_{\langle m_1,m_2\rangle}[\langle\!\langle s_1 \rangle\!\rangle/x_1]\ldots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) & \\ = & \partial_{F_1\times F_2 \lhd G}([l(\gamma^{F_1}_{m_1}) \oplus r(\gamma^{F_2}_{m_2})[\langle\!\langle s_1 \rangle\!\rangle/x_1]\ldots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) & \\ = & \langle \partial_{F_1 \lhd G}([\gamma^{F_1}_{m_1}[\langle\!\langle s_1 \rangle\!\rangle/x_1]\ldots[\langle\!\langle s_n \rangle\!\rangle/x_n]]), \partial_{F_2 \lhd G}([\gamma^{F_2}_{m_2}[\langle\!\langle s_1 \rangle\!\rangle/x_1]\ldots[\langle\!\langle s_n \rangle\!\rangle/x_n]]) \rangle & \\ \overset{IH}{=} & \langle F_1\lceil - \rceil(m_1), F_2\lceil - \rceil(m_2)\rangle & \\ = & F_1 \times F_2\lceil - \rceil(m) & \end{array}$$

$\boxed{F = F_1 + F_2}$, $\boxed{F = F_1^A}$ and $\boxed{F = \mathscr{P}_\omega F_1}$: similar to $F_1 \times F_2$.

□

We can now prove that the coalgebras $(Exp_G/_\equiv, \partial_G)$ and $(I, \overline{\omega}_G)$ are both final in the category of locally finite G-coalgebras.

4.7. **LEMMA.** *The coalgebra $(I, \overline{\omega}_G)$ is final in the category $Coalg(G)_{\mathbf{LF}}$.*

Proof. We want to show that for any locally finite G-coalgebra (S, g), there exists a *unique* homomorphism $(S, g) \to (I, \overline{\omega}_G)$. The existence is guaranteed by Lemma 4.6, where $\lceil - \rceil : S \to Exp_G/_\equiv$ is defined. Postcomposing this homomorphism with e (defined above) we get a coalgebra homomorphism $e \circ \lceil - \rceil : S \to I$. If there is another homomorphism $f : S \to I$, then by postcomposition with the inclusion $m : I \hookrightarrow \Omega$ we get two homomorphisms ($m \circ f$ and $m \circ e \circ \lceil - \rceil$) into the final G-coalgebra. Thus, f and $e \circ \lceil - \rceil$ must be equal. □

4.8. **LEMMA.** *The coalgebra $(Exp_G/_\equiv, \partial_G)$ is final in the category $Coalg(G)_{\mathbf{LF}}$.*

Proof. We want to show that for any locally finite G-coalgebra (S, g), there exists a *unique* homomorphism $(S, g) \to (Exp_G/_\equiv, \partial_G)$. We only need to prove uniqueness, since the existence is guaranteed by Lemma 4.6, where $\lceil - \rceil : S \to Exp_G/_\equiv$ is defined.

Suppose we have another homomorphism $f : S \to Exp_G/_\equiv$. Then, we shall prove that $f = \lceil - \rceil$. First, observe that because f is a homomorphism the following holds for every $s \in S$ (without any risk of confusion, in this proof that follows we denote equivalence classes $[\varepsilon]$ by expressions ε representing them):

$$f(s) = \partial_G^{-1} \circ G(f) \circ g(s) = [\gamma^G_{g(s)}[f(s_1)/x_1]\ldots[f(s_n)/x_n]] \tag{4.7}$$

where $\langle s \rangle = \{s_1, \ldots, s_n\}$, with $s_1 = s$ (recall that ∂_G^{-1} was defined in (4.3) and note that $\gamma^G_{G(f)\circ g(s)} = \gamma^G_{g(s)}[f(s_i)/x_i]$).

We now prove that $f(s_i) = \lceil s_i \rceil$, for all $i = 1, \ldots n$. For simplicity we will here prove the case $n = 3$. The general case is identical but notationally heavier. First, we prove that $f(s_1) = A_1[f(s_2)/x_2][f(s_3)/x_3]$.

$$\begin{array}{lll} & f(s_1) = \gamma^G_{g(s_1)}[f(s_1)/x_1][f(s_2)/x_2][f(s_3)/x_3] & \text{(by (4.7))} \\ \Leftrightarrow & f(s_1) = \gamma^G_{g(s_1)}[f(s_2)/x_2][f(s_3)/x_3][f(s_1)/x_1] & \text{(all } f(s_i) \text{ are closed)} \\ \Rightarrow & f(s_1) = \mu x_1.\gamma^G_{g(s_1)}[f(s_2)/x_2][f(s_3)/x_3] & \text{(by uniqueness of fixed points)} \\ \Leftrightarrow & f(s_1) = A_1[f(s_2)/x_2][f(s_3)/x_3] & \text{(def. of } A_1) \end{array}$$

Now, using what we have computed for m_1 we prove that $m_2 = A_2^1[f(s_3)/x_3]$.

$$\begin{array}{lll} & f(s_2) = \gamma^G_{g(s_2)}[f(s_1)/x_1][f(s_2)/x_2][f(s_3)/x_3] & \text{(by (4.7))} \\ & f(s_2) = \gamma^G_{g(s_2)}[A_1/x_1][f(s_2)/x_2][f(s_3)/x_3] & \text{(expressions for } f(s_1) \text{ and (3.2))} \\ \Leftrightarrow & f(s_2) = \gamma^G_{g(s_2)}[A_1/x_1][f(s_3)/x_3][f(s_2)/x_2] & \text{(all } f(s_i) \text{ are closed)} \\ \Rightarrow & f(s_2) = \mu x_2.\gamma^G_{g(s_2)}[A_1/x_1][f(s_3)/x_3] & \text{(by uniqueness of fixed points)} \\ \Leftrightarrow & f(s_2) = A_2^1[f(s_3)/x_3] & \text{(def. of } A_2^1) \end{array}$$

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

$$f(s_1) = A_1[A_2^1[f(s_3)/x_3]/x_2][f(s_3)/x_3] = A_1[A_2^1/x_2][f(s_3)/x_3]$$

Finally, we prove that $f(s_3) = A_3^2$:

$$\begin{array}{lll} & f(s_3) = \gamma^G_{g(s_3)}[f(s_1)/x_1][f(s_2)/x_2][f(s_3)/x_3] & \text{(by (4.7))} \\ \Leftrightarrow & f(s_3) = \gamma^G_{g(s_3)}[A_1/x_1][A_2^1/x_2][f(s_3)/x_3] & \text{(expr. for } f(s_i) \text{ and (3.2))} \\ \Rightarrow & f(s_3) = \mu x_3.\gamma^G_{g(s_3)}[A_1/x_1][A_2^1/x_2] & \text{(by uniqueness of fixed points)} \\ \Leftrightarrow & f(s_3) = A_3^2 & \text{(def. of } A_3^2) \end{array}$$

Thus, we have $f(s_1) = A_1[A_2^1/x_2][A_3^2/x_3]$, $f(s_2) = A_2^1[A_3^2/x_3]$ and $f(s_3) = A_3^2$. Note that $A_2^1[A_3^2/x_3] = A_2^1\{A_3^2/x_3\}$ since x_2 is not free in A_3^2. Similarly, $[A_2^1/x_2][A_3^2/x_3] = \{A_2^1/x_2\}\{A_3^2/x_3\}$. Thus $f(s_i) = \lceil s_i \rceil$, for all $i = 1,2,3$. □

It is interesting to remark that as a consequence of Lemma 4.8, we have that if G_1 and G_2 are isomorphic functors then $Exp_{G_1}/_{\equiv}$ and $Exp_{G_2}/_{\equiv}$ are also isomorphic. At this point, because final objects are unique up-to isomorphism, we know that $e : Exp_G/_{\equiv} \to I$ is an isomorphism and hence we can conclude that the map $\mathbf{beh}_{Exp_G/_{\equiv}}$ is injective, since it factorizes into an isomorphism followed by a mono. This fact is the last ingredient we need to prove completeness.

4.9. **Theorem** (Completeness). *Let G be a non-deterministic functor. For all $\varepsilon_1, \varepsilon_2 \in Exp_G$,*

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

Proof. Let G be a non-deterministic functor, let $\varepsilon_1, \varepsilon_2 \in Exp_G$ 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, since the equivalence class map $[-]$ is a homomorphism, $\mathbf{beh}_{Exp_G/_{\equiv}}([\varepsilon_1]) = \mathbf{beh}_{Exp_G/_{\equiv}}([\varepsilon_2])$. Now, because $\mathbf{beh}_{Exp_G/_{\equiv}}$ is injective we have that $[\varepsilon_1] = [\varepsilon_2]$. Hence, $\varepsilon_1 \equiv \varepsilon_2$. □

4.2. **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 canonically derived one.

Labelled transition systems. Labelled transition systems (with explicit termination) are coalgebras for the functor $1+(\mathscr{P}_\omega Id)^A$. As we will show below, instantiating our framework for this functor produces a language that is equivalent to the closed and guarded expressions generated by the following grammar, where $a \in A$ and $x \in X$ (X is a set of fixed-point variables):

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

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

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

Note that, as expected, there is no law that allows us to prove $a.(P+Q) \equiv a.P + a.Q$. Moreover, it is interesting to observe that this axiomatization is very similar to the one presented in [1]. The difference is only in the fact that we consider action prefixing instead of sequential composition. In the syntax above, δ represents deadlock and $\surd$ successuful termination.

We will now show how the beautified syntax above was derived from the canonically derived syntax for the expressions $\varepsilon \in Exp_{1+(\mathscr{P}_\omega Id)^A}$, which is given by the set of closed and guarded expressions defined by the following BNF:

$$\begin{array}{l} \varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid x \mid \mu x.\varepsilon \mid l[\varepsilon_1] \mid r[\varepsilon_2] \\ \varepsilon_1 ::= \emptyset \mid \varepsilon_1 \oplus \varepsilon_1 \mid * \\ \varepsilon_1 ::= \emptyset \mid \varepsilon_2 \oplus \varepsilon_2 \mid a(\varepsilon') \\ \varepsilon' ::= \emptyset \mid \varepsilon' \oplus \varepsilon' \mid \{\varepsilon\} \end{array}$$

We define two maps between this grammar and the grammar presented above. Let us start to show how to translate P's into ε's, by defining a map $(-)^\dagger$ by induction on the structure of P:

$$\begin{array}{lcllcl} (\mathbf{0})^\dagger & = & r[\emptyset] & (a.P)^\dagger & = & r[a(\{P^\dagger\})] \\ (P_1 + P_2)^\dagger & = & (P_1)^\dagger \oplus (P_2)^\dagger & (\surd)^\dagger & = & l[*] \\ (\mu x.P)^\dagger & = & \mu x.P^\dagger & (\delta)^\dagger & = & r[a(\emptyset)] \\ x^\dagger & = & x & & & \end{array}$$

And now the converse translation:

$$\begin{array}{lcllcl} (\emptyset)^\ddagger & = & \mathbf{0} & (l[*])^\ddagger & = & \surd \\ (\varepsilon_1 \oplus \varepsilon_2)^\ddagger & = & (\varepsilon_1)^\ddagger + (\varepsilon_2)^\ddagger & (r[\emptyset])^\ddagger & = & \delta \\ (\mu x.\varepsilon)^\ddagger & = & \mu x.\varepsilon^\ddagger & (r[\varepsilon_2 \oplus \varepsilon_2'])^\ddagger & = & (r[\varepsilon_2])^\ddagger \oplus (r[\varepsilon_2'])^\ddagger \\ x^\ddagger & = & x & (r[a(\emptyset)])^\ddagger & = & \delta \\ (l[\emptyset])^\ddagger & = & \surd & (r[a(\varepsilon_1' \oplus \varepsilon_2')])^\ddagger & = & (r[a(\varepsilon_1')])^\ddagger + (r[a(\varepsilon_2')])^\ddagger \\ (l[\varepsilon_1 \oplus \varepsilon_1'])^\ddagger & = & (l[\varepsilon_1])^\ddagger \oplus (l[\varepsilon_1'])^\ddagger & (r[a(\{\varepsilon\})])^\ddagger & = & a.\varepsilon^\ddagger \end{array}$$

One can now prove that if $P_1 \equiv P_2$ (using the equations above) then $(P_1)^\dagger \equiv (P_2)^\dagger$ (using the automatically derived equations for the functor) and also that $\varepsilon_1 \equiv \varepsilon_2$ implies $(\varepsilon_1)^\ddagger \equiv (\varepsilon_2)^\ddagger$.

Automata on guarded strings. It has recently been shown [20] 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)

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

We will not present a full comparison of this syntax to the one of Kleene algebra with tests [20] (and propositional Hoare triples). The differences between our syntax and that of KAT are similar to the ones between regular expressions and the language Exp_D for the functor representing deterministic automata (see Definition 3.5). Similarly to the LTS example one can define maps between the beautified syntax and the automatically generated one and prove its correctness.

5. Polynomial and finitary coalgebras

In this section,we will sketch how our framework can be extended to polynomial functors and finitary functors.

We start by introducing the definition of polynomial and finitary functors, which we take from [2].

5.1. **Definition** (Polynomial Functor)**.** Sums of the cartesian power functors are called polynomial functors:

$$P_\Sigma(X) = \coprod_{\sigma \in \Sigma} X^{ar(\sigma)}$$

Here, $\coprod$ stands for ordinary coproduct and the indexing set Σ is a signature, that is a possibly infinite collection of symbols σ, each of which is equipped with a finite cardinal $ar(\sigma)$, called the arity of σ. ♣

In order to be able to state and prove Kleene's theorem for polynomial functors we have to define an infinitary version of the operator $\oplus$ and extend the framework accordingly.

We start by defining the aforementioned operator on sets: $\bigoplus_{i \in I} X_i = \left(\coprod_{i \in I} X_i\right) \cup \{\bot, \top\}$ and the corresponding functor, for which we shall use the same symbol, is defined pointwise in the same way as for $\oplus$. Note that $\oplus$ is a special case of this operator (resp. functor) for I a two element set. In fact, for simplicity, we shall only consider this operator for index sets I with more than two elements.

Note that there is a natural injection between polynomial functors and the class of non-deterministic functors extended with $\oplus$: every polynomial functor $P_\Sigma(X)$ is mapped to $\overline{P}_\Sigma(X) = \bigoplus_{\sigma\in\Sigma} X^{ar(\sigma)}$.

Now, we slightly alter the definition of expressions. Instead of the expressions $l[-]$ and $r[-]$ we had before for $\oplus$ we now add an expression $i[-]$ for each $i \in I$ and the expected typing rule:

$$\frac{\vdash \varepsilon : F_j \lhd G \qquad j \in I}{\vdash j[\varepsilon] : \bigoplus_{i\in I} F_i \lhd G}$$

All the other ingredients in our story are adjusted in the expected way. We show what happens in the axiomatization. For $\oplus$ we had the rules

$$l[\varepsilon_1 \oplus \varepsilon_2] \equiv l[\varepsilon_1] \oplus l[\varepsilon_2] \qquad r[\varepsilon_1 \oplus \varepsilon_2] \equiv r[\varepsilon_1] \oplus r[\varepsilon_2] \qquad l[\varepsilon_1] \oplus r[\varepsilon_2] \equiv l[\emptyset] \oplus r[\emptyset]$$

which are now replaced by

$$i[\varepsilon_1] \oplus i[\varepsilon_2] \equiv i[\varepsilon_1 \oplus \varepsilon_2] \qquad i[\varepsilon_1] \oplus j[\varepsilon_2] \equiv i[\emptyset] \oplus j[\emptyset], \quad i \neq j$$

At this point we can define expressions for polynomial functors and state Kleene's theorem.

5.2. **Definition** (Expressions for polynomial functors)**.** Let P_Σ be a polynomial functor and $\overline{P}_\Sigma$ the corresponding non-deterministic functor as defined above. The set of expressions for P_Σ are defined as

$$Exp_{P_\Sigma} = \{\varepsilon \in Exp_{\overline{P}_\Sigma} \mid \varepsilon \not\equiv \emptyset \text{ and } \varepsilon \not\equiv i[\emptyset] \oplus j[\emptyset], \quad i \neq j\}$$

♣

It is interesting to remark that in the expressions for polynomial functors we are eliminating those expressions relative to the $\oplus$ functor which would lead to underspecification (represented by $\bot$) and overspecification (represented by $\top$). These features are not representable in polynomial functors because of the use of ordinary coproduct instead of $\oplus$. Note that because of the simple format of $\overline{P}$ the two restrictions above could be easily expressed by refining the type system in Definition 3.2 in order to not allow $\emptyset$ and $\varepsilon_1 \oplus \varepsilon_2$ to be of type $\bigoplus_{\sigma\in\Sigma} X^{ar(\sigma)} \lhd G$.

5.3. **Theorem** (Kleene's theorem for polynomial functors)**.** *Let P_Σ be a polynomial functor.*

1. *For every locally finite coalgebra $(S, g : S \to P_\Sigma(S))$ and for every $s \in S$ there exists an expression $\varepsilon \in Exp_{P_\Sigma}$ such that $\varepsilon \sim s$.*
2. *For every expression $\varepsilon \in Exp_{P_\Sigma}$ there is a finite coalgebra $(S, g : S \to P_\Sigma(S))$ with $s \in S$ such that $s \sim \varepsilon$.*

The proof of this theorem is an adaptation of the proofs of theorems 3.12 and 3.14.

The axiomatization of Exp_{P_Σ} is the same as the one for $Exp_{\overline{P}_\Sigma}$ (which we used above to define Exp_{P_Σ}) apart from the axiom $i[\varepsilon_1] \oplus j[\varepsilon_2] \equiv i[\emptyset] \oplus j[\emptyset], \quad i \neq j$, which is removed.

We now repeat the exercise above for finitary functors. A finitary functor $\mathbf{F}$ is a functor that is a quotient of a polynomial functor, *i.e.* there exists a natural transformation $\eta : P_\Sigma \to \mathbf{F}$, whose components $\eta : P_\Sigma(X) \to \mathbf{F}(X)$ are epimorphisms. We define $Exp_{\mathbf{F}} = Exp_P$.

5.4. **Theorem** (Kleene's theorem for finitary functors)**.** *Let $\mathbf{F}$ be a finitary functor.*

1. *Let (S, f) be a locally-finite $\mathbf{F}$-coalgebra. Then, for any $s \in S$, there exists an expression $\langle\!\langle s \rangle\!\rangle \in Exp_{\mathbf{F}}$ such that $s \sim \langle\!\langle s \rangle\!\rangle$.*

(2) *Let $\varepsilon \in Exp_{\mathbf{F}}$. Then, there exists a finite* **F***-coalgebra (S, f) with $s \in S$ such that $s \sim \varepsilon$.*

Proof. Let **F** be a finitary functor (quotient of a polynomial functor P). For every **F**-coalgebra (T, t), we have:

$$\begin{array}{ccc} T & \xrightarrow{id} & T \\ {\scriptstyle f^\sharp}\downarrow & & \downarrow{\scriptstyle f} \\ P(T) & \xrightarrow[\eta_S]{} & \mathbf{F}(T) \end{array}$$

Also note that as a consequence of naturality $t_1 \sim_P t_2 \Rightarrow t_1 \sim_{\mathbf{F}} t_2$.

① Let (S, f) be a locally finite **F**-coalgebra, with $f^\sharp(s) = r(\eta^{-1}(f(s)))$ where r picks a representative from $\eta^{-1}(f(s)) \subseteq \mathscr{P}_\omega(P(S))$. We then build $\langle\!\langle s \rangle\!\rangle$ w.r.t $f^\sharp$ just as in Theorem 3.12 (note that $(S, f^\sharp)$ is also locally-finite) and the result follows because $\langle\!\langle s \rangle\!\rangle \sim_{\mathbf{F}} s \Leftarrow \langle\!\langle s \rangle\!\rangle \sim_{\mathbf{P}} s$ (consequence of naturality).

② Let $\varepsilon \in Exp_{\mathbf{F}}$. By Theorem 3.14, there exists a finite P-coalgebra (S, f) with $s \in S$ such that $s \sim_P \varepsilon$. Thus, we take $(S, \eta_S \circ f)$ and we have a finite **F**-coalgebra with $s \in S$ such that $\varepsilon \sim_{\mathbf{F}} s$. □

For the axiomatization a bit more ingenuity is required. One needs to derive which extra axioms are induced by the epimorphism and then prove that they are sound and complete.

For instance, the finite powerset functor (which we included in the syntax of nondeterministic functors) is the classical example of a finitary functor. It is the quotient of the polynomial functor $P(X) = 1 + X + X^2 + \ldots$ (this represents lists of length n) by identifying lists that contain precisely the same elements (that is, eliminating repeated elements). Thus, this restriction would have to be translated to the syntax (which is not always obvious how to achieve).

In this particular case the syntax for Exp_P is the set of closed and guarded expressions given by the following BNF:

$$\begin{array}{l} \varepsilon ::= \emptyset \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon \mid x \mid i[\varepsilon_i], \quad i \in \mathbb{N} \\ \varepsilon_0 ::= \emptyset \mid \varepsilon_0 \oplus \varepsilon_0 \\ \varepsilon_i ::= \emptyset \mid \varepsilon_i \oplus \varepsilon_i \mid l\langle\varepsilon\rangle \mid r\langle\varepsilon_{i-1}\rangle \end{array}$$

such that $\varepsilon \not\equiv \emptyset$ and $\varepsilon \not\equiv i[\emptyset] \oplus j[\emptyset]$, $i \neq j$. Taking into account the restriction mentioned above, plus the two conditions stemming from the expressions for polynomial functors, would lead to the following simplified syntax

$$\varepsilon ::= \mu x.\varepsilon \mid x \mid 0[*] \mid 1[\varepsilon] \mid k[\bigoplus_{i=1,\ldots,k'} \varepsilon_i]$$

for all $k = 2, \ldots$ and $k' \leq k$, together with the axioms for the fixed-point, $(\alpha - equiv)$, (*Commutativity*), (*Associativity*), (*Idempotency*) and the extra axiom:

$$k[\bigoplus_{i=1,\ldots,k'} \varepsilon_i] \equiv k'[\bigoplus_{i=1,\ldots,k'} \varepsilon_i] \text{ if all } \varepsilon_1, \ldots, \varepsilon_{k'} \text{ are distinct}$$

In this case, it is easy to see that this set of axioms is sound and complete. The restricted syntax and axioms needs to be derived for each concrete finitary functor. Finding a uniform way of defining such restricted syntax/axioms and also uniformly proving soundness and completeness is an interesting problem and it is left as future work.

6. DISCUSSION

We presented a systematic way of deriving, from the type of a system, a language of (generalized) regular expressions and a sound and complete axiomatization thereof. We presented the analogue of Kleene's theorem, proving the coincidence of the behaviours captured by the expressions and the systems under consideration. The whole approach was illustrated with five examples: deterministic finite automata, partial deterministic automata, non-deterministic automata, labelled transition systems and automata on guarded strings. Moreover, all the results presented in [7] can be recovered as a particular instance of the present framework.

Iterative algebras [3] formalise potentially infinite computations as unique solutions of recursive equations. The language Exp_G can be reviewed as a syntactic characterization of the free iterative G-algebra. It would be interesting to investigate the connections with iterative algebras further. Concretely, complete iterative algebras [24] might allow for extending our work further than locally finite coalgebras, since they provide a way of computing solutions for infinite systems of equations (which would be needed for Kleene's theorem).

In [17], a bialgebraic review of deterministic automata and regular expressions was presented. We expect that these results extend neatly to our framework, but fully working this out is left as future work.

In this paper we studied coalgebras for **Set** functors. It is an interesting and challenging question to extend our results to other categories. In particular, it would be interesting to study functors over the metric spaces [32, 10].

In his seminal paper [18], S. Kleene introduced an algebraic description of regular languages: regular expressions. This was the precursor of many papers, including this one. Salomaa [30] presented a sound and complete axiomatization for proving the equivalence of regular expressions. This was later refined by Kozen in [19]. In [25], Milner introduced a set of expressions for finite LTS's and proved an analogue of Kleene's theorem: each expression denotes the behaviour of a finite LTS and, conversely, the behaviour of a finite LTS can be specified by an expression. He also provided an axiomatization for his expressions, with the property that two expressions are provably equivalent if and only if they are bisimilar.

Our approach is inspired by the work of Kleene and Milner. For that reason we have $\emptyset$ and $\oplus$ in the syntax of our expressions. One could have taken a different approach and eliminate the need of these two operators. The main changes would be to consider an expression $\langle -, - \rangle$ for the product instead of the two expressions $l\langle - \rangle$ and $r\langle - \rangle$ which we considered and an expression $\langle a_1(-), a_2(-), \ldots, a_n(-) \rangle$ for the exponential (with $A = \{a_1, \ldots a_n\}$). This would also lead to the use of constant sets (instead of join semilatices) and normal coproduct (instead of $\oplus$) in our functor definitions (the class of non-deterministic functors would then coincide with the so-called Kripke polynomial functors). As an example take the functor $D(X) = 2 \times X^A$ of deterministic automata. The expressions corresponding to this functor would then be the set of closed and guarded expressions given by the following BNF:

$$\varepsilon ::= x \mid \mu x.\varepsilon \mid \langle 0, \langle a_1(\varepsilon), a_2(\varepsilon), \ldots, a_n(\varepsilon) \rangle \rangle \mid \langle 1, \langle a_1(\varepsilon), a_2(\varepsilon), \ldots, a_n(\varepsilon) \rangle \rangle$$

This syntax can be perceived as an explicit and complete description of the automaton. This means that underspecification is inexistent and the compactness of regular expressions is lost. As an example of the verbosity present in this new language, take $A = \{a, b, c\}$ and consider the language that accepts words with only a's and has at last one a (described by

aa^* in Kleene's regular expressions). In the language Exp_D it would be written as $\mu x.a(l\langle 1\rangle \oplus x)$. Using the approach described above it would be encoded as the expression

$$\mu x.\langle 0, \langle a(\langle 1, \langle a(x), b(empty), c(empty)\rangle\rangle), b(empty), c(empty)\rangle\rangle$$

where $empty = \mu y.\langle 0, \langle a(y), b(y), c(y)\rangle$ is the expression denoting the empty language. For this new syntax, the proof of Kleene's theorem follows a similar strategy as the one we presented and the axiomatization would only contain the axioms for the fixed-point. The approach we took provides a more user-friendly syntax and stays close to the know syntaxes for deterministic automata and LTSs.

The connection between regular expressions and coalgebras was first explored in [29]. There deterministic automata, the set of formal languages and regular expressions are all presented as coalgebras of the functor $2 \times Id^A$ (where A is the alphabet, and 2 is the two element set). It is then shown that the standard semantics of language acceptance of automata and the assignment of languages to regular expressions both arise as the unique homomorphism into the final coalgebra of formal languages. The coalgebra structure on the set of regular expressions is determined by their so-called *Brzozowski* derivatives [11]. In the present paper, the set of expressions for the functor $F(S) = 2 \times S^A$ differs from the classical definition in that we do not have Kleene star and full concatenation (sequential composition) but, instead, the least fixed point operator and action prefixing. Modulo that difference, the definition of a coalgebra structure on the set of expressions in both [29] and the present paper is essentially the same. All in all, one can therefore say that standard regular expressions and their treatment in [29] can be viewed as a special instance of the present approach. This is also the case for the generalization of the results in [29] to automata on guarded strings [20]. Finally, the present paper extends the results in our FoSSaCS'08 paper [7], where a sound and complete specification language and a synthesis algorithm for Mealy machines is given. Mealy machines are coalgebras of the functor $(B \times Id)^A$, where A is a finite input alphabet and B is a finite semilattice for the output alphabet. Part of the material of the present paper is based on two conference papers: our FoSSaCS'09 paper [9] and our LICS'09 paper [8].

In the last few years several proposals of specification languages for coalgebras appeared [26, 27, 15, 14, 12, 5, 6, 21]. Our approach is similar in spirit to that of Goldblatt [14] in that we use the ingredients of a functor for typing expressions, and differs from [27, 15] because we do not need an explicit "next-state" operator, as we can deduce it from the type information. Apart from [21], the logics presented do not include fixed-point operators. Our language of regular expressions can be seen as an extension with fixed point operators of the coalgebraic logic of [5] and it is similar to a fragment of the logic presented in [21]. However, our goal is rather different: we want a *finitary* language that characterizes exactly all *finite* coalgebras. Our language is minimal, as it contains only the operators necessary for this goal. This restriction does not decrease the expressiveness of the language with respect to bisimulation and it allows for a simple and direct algorithm for the synthesis of a finite coalgebra.

All the results presented in this paper can be extended in order to accommodate systems with *quantities*, such as probability or costs [4]. This allows for the derivation of specification languages and axiomatizations for a wide variety of systems, which include weighted automata, simple probabilistic systems (also known as Markov chains) and systems with mixed probability and non-determinism (such as Segala systems). Here is when generality of our approach really brings new results. For instance, we have derived a language and an

axiomatization for the so-called stratified systems. The language is equivalent to the one presented in [13], but no axiomatization was known.

The derivation of the syntax and axioms associated with each non-deterministic functor has been implemented in the coinductive prover CIRC [23]. This allows for automatic reasoning about the equivalence of expressions specifying systems.

Acknowledgements. The authors are grateful for useful comments from several people: Filippo Bonchi, Helle Hansen, Bartek Klin, Dexter Kozen, Clemens Kupke, Stefan Milius, Prakash Panagaden, Ana Sokolova, Yde Venema and Erik de Vink. The title of this paper was inspired by the title of a section of a paper of Dexter Kozen [20]. The proof of soundness and completeness was simplified (when compared with the one presented in our LICS paper [8]) inspired by recent work of Stefan Milius on expressions for linear systems (personal communication).

REFERENCES

[1] Luca Aceto and Matthew Hennessy. Termination, deadlock, and divergence. *J. ACM*, 39(1):147–187, 1992. pages 19, 28

[2] Jirí Adámek, Dominik Lücke, and Stefan Milius. Recursive coalgebras of finitary functors. *ITA*, 41(4):447–462, 2007. pages 29

[3] Jirí Adámek, Stefan Milius, and Jiri Velebil. Iterative algebras at work. *Mathematical Structures in Computer Science*, 16(6):1085–1131, 2006. pages 32

[4] Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Deriving syntax and axioms for quantitative regular behaviours. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2009. pages 33

[5] Marcello M. Bonsangue and Alexander Kurz. Duality for logics of transition systems. In Vladimiro Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 2005. pages 33

[6] Marcello M. Bonsangue and Alexander Kurz. Presenting functors by operations and equations. In Luca Aceto and Anna Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *LNCS*, pages 172–186. Springer, 2006. pages 33

[7] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Coalgebraic logic and synthesis of Mealy machines. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2008. pages 1, 3, 32, 33

[8] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. An algebra for Kripke polynomial coalgebras. In *LICS*, pages 49–58. IEEE Computer Society, 2009. pages 33, 34

[9] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. A Kleene theorem for polynomial coalgebras. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2009. pages 33

[10] Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006. pages 32

[11] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964. pages 10, 14, 24, 33

[12] Corina Cîrstea and Dirk Pattinson. Modular construction of modal logics. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2004. pages 33

[13] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995. pages 34

[14] Robert Goldblatt. Equational logic of polynomial coalgebras. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic 4*, pages 149–184. King's College Publications, 2002. pages 33

[15] Bart Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *ITA*, 35(1):31–59, 2001. pages 33

[16] Bart Jacobs. *Introduction to Coalgebra. Towards Mathematics of States and Observations*. 2004. Draft. pages 4

[17] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006. pages 32

[18] Stephen Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956. pages 1, 32

[19] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Logic in Computer Science*, pages 214–225, 1991. pages 1, 32

[20] Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report http://hdl.handle.net/1813/10173, Computing and Information Science, Cornell University, March 2008. pages 29, 33, 34

[21] Clemens Kupke and Yde Venema. Coalgebraic automata theory: basic results. Technical Report SEN-E0701, CWI, The Netherlands, 2007. pages 33

[22] Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors. *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7-10, 2009. Proceedings*, volume 5728 of *Lecture Notes in Computer Science*. Springer, 2009. pages 35

[23] Dorel Lucanu, Eugen-Ioan Goriac, Georgiana Caltais, and Grigore Rosu. Circ: A behavioral verification tool based on circular coinduction. In Kurz et al. [22], pages 433–442. pages 34

[24] Stefan Milius. Completely iterative algebras and completely iterative monads. *Inf. Comput.*, 196(1):1–41, 2005. pages 32

[25] Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. pages 1, 7, 32

[26] Larry Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96, 1999. pages 33

[27] Martin Rößiger. Coalgebras and modal logic. *Electronic Notes in Theoretical Computer Science*, 33, 2000. pages 33

[28] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. pages 2, 4, 5, 23

[29] Jan J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218, 1998. pages 33

[30] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. pages 32

[31] Sam Staton. Relating coalgebraic notions of bisimulation. In Kurz et al. [22], pages 191–205. pages 5

[32] Daniele Turi and Jan J. M. M. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998. pages 32

Centrum Wiskunde & Informatica (CWI) is the national research institute for mathematics and computer science in the Netherlands. The institute's strategy is to concentrate research on four broad, societally relevant themes: earth and life sciences, the data explosion, societal logistics and software as service.

Centrum Wiskunde & Informatica (CWI) is het nationale onderzoeksinstituut op het gebied van wiskunde en informatica. De strategie van het instituut concentreert zich op vier maatschappelijk relevante onderzoeksthema's: aard- en levenswetenschappen, de data-explosie, maatschappelijke logistiek en software als service.

Bezoekadres:
Science Park 123
Amsterdam

Postadres:
Postbus 94079, 1090 GB Amsterdam
Telefoon 020 592 93 33
Fax 020 592 41 99
info@cwi.nl
www.cwi.nl

CWI

Centrum Wiskunde & Informatica