Generalizing the powerset construction, coalgebraically *

Alexandra Silva[1], **Filippo Bonchi**[2], **Marcello M. Bonsangue**[1,3], **and Jan J. M. M. Rutten**[1,4]

1 **Centrum Wiskunde & Informatica**
2 **CNRS - LIP, ENS Lyon**
3 **LIACS - Leiden University**
4 **Radboud University Nijmegen**

Abstract

Coalgebra is an abstract framework for the uniform study of different kinds of dynamical systems. An endofunctor F determines both the type of systems (F-coalgebras) and a notion of behavioral equivalence ($\sim_F$) amongst them. Many types of transition systems and their equivalences can be captured by a functor F. For example, for deterministic automata the derived equivalence is language equivalence, while for non-deterministic automata it is ordinary bisimilarity. The powerset construction is a standard method for converting a nondeterministic automaton into an equivalent deterministic one as far as language is concerned. In this paper, we lift the powerset construction on automata to the more general framework of coalgebras with structured state spaces. Examples of applications include partial Mealy machines, (structured) Moore automata, and Rabin probabilistic automata.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.272

1 Introduction

Coalgebra is by now a well established general framework for the study of the behaviour of large classes of dynamical systems, including various kinds of automata (deterministic, probabilistic etc.) and infinite data types (streams, trees and the like). For a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, an F-coalgebra is a pair (X, f), consisting of a set X of states and a function $f\colon X \to F(X)$ defining the observations and transitions of the states. Coalgebras generally come equipped with a standard notion of equivalence called *F-behavioural equivalence* that is fully determined by their (functor) type F. Moreover, for most functors F there exists a *final* coalgebra into which any F-coalgebra is mapped by a unique homomorphism that identifies all F-equivalent states.

Much of the coalgebraic approach can be nicely illustrated with deterministic automata (DA), which are coalgebras of the functor $D(X) = 2 \times X^A$. In a DA, two states are D-equivalent precisely when they accept the same language. The set 2^{A^*} of all formal languages constitutes a final D-coalgebra, into which every DA is mapped by a homomorphism that sends any state to the language it accepts.

It is well-known that *non-deterministic* automata (NDA) often provide more efficient (smaller) representations of formal languages than DA's. Language acceptance of NDA's is typically defined by turning them into DA's via the *powerset construction.* Coalgebraically this works as follows. NDA's are coalgebras of the functor $N(X) = 2 \times \mathcal{P}_\omega(X)^A$, where

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

$\mathcal{P}_\omega$ is the finite powerset. An N-coalgebra $(X, f\colon X \to 2 \times \mathcal{P}_\omega(X)^A)$ is *determinized* by transforming it into a D-coalgebra $(\mathcal{P}_\omega(X), f^\sharp\colon \mathcal{P}_\omega(X) \to 2 \times \mathcal{P}_\omega(X)^A)$ (for details see Section 3). Then, the language accepted by a state s in the NDA (X, f) is defined as the language accepted by the state $\{s\}$ in the DA $(\mathcal{P}_\omega(X), f^\sharp)$.

For a second variation on DA's, we look at *partial automata* (PA): coalgebras of the functor $P(X) = 2 \times (1 + X)^A$, where for certain input letters transitions may be undefined. Again, one is often interested in the DA-behaviour (i.e., language acceptance) of PA's. This can be obtained by turning them into DA's using *totalization*. Coalgebraically, this amounts to the transformation of a P-coalgebra $(X, f\colon X \to 2 \times (1 + X)^A)$ into a D-coalgebra $(1 + X, f^\sharp\colon 1 + X \to 2 \times (1 + X)^A)$.

Although the two examples above may seem very different, they are both instances of one and the same phenomenon, which it is the goal of the present paper to describe at a general level. Both with NDA's and PA's, two things happen at the same time: (i) more (or, more generally, different types of) transitions are allowed, as a consequence of changing the functor type by replacing X by $\mathcal{P}_\omega(X)$ and $(1 + X)$, respectively; and (ii) the behaviour of NDA's and PA's is still given in terms of the behaviour of the original DA's (language acceptance).

For a large family of F-coalgebras, both (i) and (ii) can be captured simultaneously with the help of the categorical notion of *monad*, which generalizes the notion of algebraic theory. The structuring of the state space X can be expressed as a change of functor type from $F(X)$ to $F(T(X))$. In our examples above, both the functors $T_1(X) = \mathcal{P}_\omega(X)$ and $T_2(X) = 1 + X$ are monads, and NDA's and PA's are obtained from DA's by changing the original functor type $D(X)$ into $N(X) = D(T_1(X))$ and $P(X) = D(T_2(X))$. Regarding (ii), one assigns F-semantics to an FT-coalgebra (X, f) by transforming it into an F-coalgebra $(T(X), f^\sharp)$, again using the monad T. In our examples above, the determinization of NDA's and the totalization of PA's consists of the transformation of N- and P-coalgebras (X, f) into D-coalgebras $(T_1(X), f^\sharp)$ and $(T_2(X), f^\sharp)$, respectively.

We shall investigate general conditions on the functor types under which the above constructions can be applied: for one thing, one has to ensure that the FT-coalgebra map f induces a suitable F-coalgebra map $f^\sharp$. Our results will lead to a uniform treatment of all kinds of existing and new variations of automata, that is, FT-coalgebras, by an algebraic structuring of their state space through a monad T. Furthermore, we shall prove a number of general properties that hold in all situations similar to the ones above. For instance, there is the notion of N-behavioural equivalence with which NDA's, being N-coalgebras, come equipped. It coincides with the well-known notion of Park-Milner bisimilarity from process algebra. A general observation is that if two states in an NDA are N-equivalent then they are also D- (that is, language-) equivalent. For PA's, a similar statement holds. One further contribution of this paper is a proof of these statements, once and for all for all FT-coalgebras under consideration.

Coalgebras of type FT were studied in [15, 2, 11]. In [2, 11] the main concern was definitions by coinduction, whereas in [15] a proof principle was also presented. All in all, the present paper can be seen as the understanding of the aforementioned papers from a new perspective, presenting a uniform view on various automata constructions and equivalences.

The structure of the paper is as follows. After preliminaries (Section 2) and the details of the motivating examples above (Section 3), Section 4 presents the general construction as well as many more examples. In Section 5, a large family of automata (technically: functors) is characterized to which the constructions above can be applied. Section 6 discusses related work and presents pointers to future work. A technical report [27] contains all the proofs as well as further examples.

2 Background

In this section we introduce the preliminaries on coalgebras and algebras. First, we fix some notation on sets. We will denote sets by capital letters $X, Y, \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is the cartesian product of X and Y (with the usual projection maps π_1 and π_2), $X + Y$ is the disjoint union (with injection maps κ_1 and κ_2) and X^Y is the set of functions $f{:}Y \to X$. The collection of finite subsets of X is denoted by $\mathcal{P}_\omega(X)$, while the collection of full-probability distributions with finite support is $\mathcal{D}_\omega(X) = \{f\colon X \to [0,1] \mid f \text{ finite support and } \sum_{x \in X} f(x) = 1\}$. For a set of letters A, A^* denotes the set of all words over A; ϵ the empty word; and $w_1 \cdot w_2$ (and $w_1 w_2$) the concatenation of words $w_1, w_2 \in A^*$.

2.1 Coalgebras

A coalgebra is a pair $(X, f\colon X \to F(X))$, where X is a set of states and $F\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor F, together with the function f, determines the *transition structure* (or dynamics) of the F-coalgebra [22].

An F-*homomorphism* from an F-coalgebra (X, f) to an F-coalgebra (Y, g) is a function $h{:}\, X \to Y$ preserving the transition structure, *i.e.*, $g \circ h = F(h) \circ f$.

An F-coalgebra (Ω, ω) is said to be *final* if for any F-coalgebra (X, f) there exists a unique F-homomorphism $[\![-]\!]_X\colon X \to \Omega$. All the functors considered in examples in this paper have a final coalgebra.

Let (X, f) and (Y, g) be two F-coalgebras. We say that the states $x \in X$ and $y \in Y$ are *behaviourally equivalent*, written $x \sim_F y$, if and only if they are mapped into the same element in the final coalgebra, that is $[\![x]\!]_X = [\![y]\!]_Y$.

2.2 Algebras

Monads can be thought of as a generalization of algebraic theories. A *monad* $\mathbf{T} = (T, \mu, \eta)$ is a triple consisting of an endofunctor T on $\mathbf{Set}$ and two natural transformations: a *unit* $\eta\colon \mathit{Id} \Rightarrow T$ mapping a set X to its free algebra $T(X)$, and a *multiplication* $\mu\colon T^2 \Rightarrow T$. They satisfy the following commutative laws

$$\mu \circ \eta_T = id_T = \mu \circ T\eta \quad \text{and} \quad \mu \circ \mu_T = \mu \circ T\mu.$$

Sometimes it is more convenient to represent a monad $\mathbf{T}$, equivalently, as a *Kleisli triple* $(T, (\_)^\sharp, \eta)$ [17], where T assigns a set $T(X)$ to each set X, the unit η assigns a function $\eta_X\colon X \to T(X)$ to each set X, and the extension operation $(\_)^\sharp$ assigns to each $f\colon X \to T(Y)$ a function $f^\sharp\colon T(X) \to T(Y)$, such that,

$$f^\sharp \circ \eta_X = f \qquad (\eta_X)^\sharp = id_{T(X)} \qquad (g^\sharp \circ f)^\sharp = g^\sharp \circ f^\sharp\,,$$

for $g\colon Y \to T(Z)$. Monads are frequently referred to as *computational types* [18]. We list now a few examples. In what follows, $f\colon X \to T(Y)$ and $c \in T(X)$.

Nondeterminism $T(X) = \mathcal{P}_\omega(X)$; η_X is the singleton map $x \mapsto \{x\}$; $f^\sharp(c) = \bigcup_{x \in c} f(x)$.

Partiality $T(X) = 1 + X$ where $1 = \{*\}$ represents a terminating (or diverging) computation; η_X is the injection map $\kappa_2\colon X \to 1 + X$; $f^\sharp(\kappa_1(*)) = \kappa_1(*)$ and $f^\sharp(\kappa_2(x)) = f(x)$.

Further examples of monads include: exceptions ($T(X) = E + X$), side-effects ($T(X) = (S \times X)^S$), interactive output ($T(X) = \mu v.X + (O \times v) \cong O^* \times X$) and full-probability

($T(X) = \mathcal{D}_\omega(X)$). We will use all these monads in our examples and we will define η_X and $f^\sharp$ for each later in Section 4.1.

A $\mathbf{T}$-*algebra* of a monad $\mathbf{T}$ is a pair (X, h) consisting of a set X, called carrier, and a function $h\colon T(X) \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id_X$. A T-homomorphism between two $\mathbf{T}$-algebras (X, h) and (Y, k) is a function $f\colon X \to Y$ such that $f \circ h = k \circ Tf$. $\mathbf{T}$-algebras and their homomorphisms form the so-called *Eilenberg-Moore category* $\mathbf{Set}^\mathbf{T}$. There is a forgetful functor $U^\mathbf{T}\colon \mathbf{Set}^\mathbf{T} \to \mathbf{Set}$ defined by

$$U^\mathbf{T}((X,h)) = X \quad \text{and} \quad U^\mathbf{T}(f\colon (X,h) \to (Y,k)) = f\colon X \to Y\,.$$

The forgetful functor $U^\mathbf{T}$ has left adjoint $X \mapsto (T(X), \mu_X\colon TT(X) \to T(X))$, mapping a set X to its free $\mathbf{T}$-algebra. If $f\colon X \to Y$ with (Y, h) a $\mathbf{T}$-algebra, the unique $\mathbf{T}$-homomorphism $f^\sharp\colon (T(X), \mu_X) \to (Y, h)$ with $f^\sharp \circ \eta_X = f$ is given by

$$f^\sharp\colon T(X) \xrightarrow{Tf} T(Y) \xrightarrow{h} Y\,.$$

The function $f^\sharp\colon (T(X), \mu_X) \to (T(Y), \mu_Y)$ coincides with function extension for a Kleisli triple. For the monad $\mathcal{P}_\omega$ the associated Eilenberg-Moore category is the category of join semi-lattices, whereas for the monad $1 + -$ is the category of pointed sets.

3 Motivating examples

In this section, we introduce two motivating examples. We will present two constructions, the determinization of a non-deterministic automaton and the totalization of a partial automaton, which we will later show to be an instance of the same, more general, construction.

3.1 Non-deterministic automata

A deterministic automaton (DA) over the input alphabet A is a pair $(X, \langle o, t\rangle)$, where X is a set of states and $\langle o, t\rangle\colon X \to 2 \times X^A$ is a function with two components: o, the output function, determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each input letter a the next state. DA's are coalgebras for the functor $2 \times Id^A$. The final coalgebra of this functor is $(2^{A^*}, \langle \epsilon, (-)_a\rangle)$ where 2^{A^*} is the set of languages over A and $\langle \epsilon, (-)_a\rangle$, given a language L, determines whether or not the empty word is in the language ($\epsilon(L) = 1$ or $\epsilon(L) = 0$, resp.) and, for each input letter a, returns the *derivative* of L: $L_a = \{w \in A^* \mid aw \in L\}$. From any DA, there is a unique map l into 2^{A^*} which assigns to each state its behaviour (that is, the language that the state recognizes).

$$\begin{tikzcd}
X \arrow[r, dashed, "l"] \arrow[d, "\langle o,t\rangle"'] & 2^{A^*} \arrow[d, "\langle \epsilon,(-)_a\rangle"] \\
2 \times X^A \arrow[r, dashed, "id\times l^A"'] & 2 \times (2^{A^*})^A
\end{tikzcd}$$

A non-deterministic automaton (NDA) is similar to a DA but the transition function gives a set of next-states for each input letter instead of a single state. Thus, an NDA over the input alphabet A is a pair $(X, \langle o, \delta\rangle)$, where X is a set of states and $\langle o, \delta\rangle\colon X \to 2 \times (\mathcal{P}_\omega(X))^A$ is a pair of functions with o as before and where δ determines for each input letter a a set of possible next states. In order to compute the language recognized by a state x of an NDA $\mathcal{A}$, it is usual to first determinize it, constructing a DA $\mathbf{det}(\mathcal{A})$ where the state space is $\mathcal{P}_\omega(X)$, and then compute the language recognized by the state $\{x\}$ of $\mathbf{det}(\mathcal{A})$. Next, we describe in coalgebraic terms how to construct the automaton $\mathbf{det}(\mathcal{A})$.

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

$$\overline{o}(Y) = \begin{cases} 1 & \exists_{y\in Y}\, o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad t(Y)(a) = \bigcup_{y\in Y} \delta(y)(a).$$

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

We summarize the situation above with the following commuting diagram:

$$\begin{tikzcd}
X \arrow[r, "\{\cdot\}"] \arrow[d, "\langle o,\delta\rangle"'] & \mathcal{P}_\omega(X) \arrow[r, dashed, "l"] \arrow[dl, "\langle \overline{o},t\rangle"] & 2^{A^*} \arrow[d, "\langle \epsilon,(-)_a\rangle"] \\
2\times\mathcal{P}_\omega(X)^A \arrow[rr, dashed, "id\times l^A"'] & & 2\times(2^{A^*})^A
\end{tikzcd}$$

We note that the language semantics of NDA's, presented in the above diagram, can also be obtained as an instance of the abstract definition scheme of λ-coinduction [2, 11].

3.2 Partial automata

A partial automaton (PA) over the input alphabet A is a pair $(X, \langle o, \partial\rangle)$ consisting of a set of states X and a pair of functions $\langle o, \partial\rangle\colon X \to 2 \times (1 + X)^A$, with $o\colon X \to 2$ as for DA and $\partial\colon X \to (1 + X)^A$ a transition function, which for any input letter a is either undefined (no a-labelled transition takes place) or specifies the next state that is reached. PA's are coalgebras for the functor $2 \times (1 + Id)^A$. Given a PA $\mathcal{A}$, we can construct a total (deterministic) automaton $\mathbf{tot}(\mathcal{A})$ by adding an extra *sink* state to the state space: every undefined a-transition from a state x is then replaced by a a-labelled transition from x to the sink state. More precisely, given a PA $\mathcal{A} = (X, \langle o, \partial\rangle)$, we construct $\mathbf{tot}(\mathcal{A}) = (1 + X, \langle \overline{o}, t\rangle)$, where

$$\begin{array}{ll} \overline{o}(\kappa_1(*)) = 0 \qquad\qquad & t(\kappa_1(*))(a) = \kappa_1(*) \\ \overline{o}(\kappa_2(x)) = o(x) & t(\kappa_2(x))(a) = \partial(x)(a) \end{array}$$

The language $l(x)$ recognized by a state x will be precisely the language recognized by x in the original partial automaton. Moreover, the new sink state recognizes the empty language. Again we summarize the situation above with the help of following commuting diagram, which illustrates the similarities between both constructions:

$$\begin{tikzcd}
X \arrow[r, "\kappa_2"] \arrow[d, "\langle o,\partial\rangle"'] & 1+X \arrow[r, dashed, "l"] \arrow[dl, "\langle \overline{o},t\rangle"] & 2^{A^*} \arrow[d, "\langle \epsilon,(-)_a\rangle"] \\
2\times(1+X)^A \arrow[rr, dashed, "id\times l^A"'] & & 2\times(2^{A^*})^A
\end{tikzcd}$$

4 Algebraically structured coalgebras

In this section we present a general framework where both motivating examples can be embedded and uniformly studied. We will consider coalgebras for which the functor type FT

can be decomposed into a transition type F specifying the relevant dynamics of a system and a monad T providing the state space with an algebraic structure. For simplicity, we fix our base category to be **Set**.

We will study coalgebras $f\colon X \to FT(X)$ for a functor F and a monad $\mathbf{T}$ such that $FT(X)$ is a $\mathbf{T}$-algebra, that is $FT(X)$ is the carrier of a $\mathbf{T}$-algebra $(FT(X), h)$. In the motivating examples, F would be instantiated to $2 \times Id^A$ (in both) and T to $\mathcal{P}_\omega$, for NDAs, and to $1 + -$ for PAs. The condition that $FT(X)$ is a $\mathbf{T}$-algebra would amount to require that $2 \times \mathcal{P}_\omega(X)^A$ is a join-semilattice, for NDAs, and that $2 \times (1 + X)^A$ is a pointed set, for PAs. This is indeed the case, since the set 2 can be regarded both as a join-semilattice ($2 \cong \mathcal{P}_\omega(1)$) or as a pointed set ($2 \cong 1+1$) and, moreover, products and exponentials preserve the algebra structure.

The inter-play between the transition type F and the computational type $\mathbf{T}$ (more precisely, the fact that $FT(X)$ is a $\mathbf{T}$-algebra) will allow each coalgebra $f\colon X \to FT(X)$ to be extended uniquely to a T-algebra morphism $f^\sharp\colon (T(X), \mu_X) \to (FT(X), h)$ which makes the following diagram commute.

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "f"'] & T(X) \arrow[dl, "f^\sharp"] \\
FT(X) &
\end{tikzcd} \qquad f^\sharp \circ \eta_X = f$$

Intuitively, $\eta_X\colon X \to T(X)$ is the inclusion of the state space of the coalgebra $f\colon X \to FT(X)$ into the structured state space $T(X)$, and $f^\sharp\colon T(X) \to FT(X)$ is the extension of the coalgebra f to $T(X)$.

Next, we will study the behaviour of a given state or, more generally, we would like to say when two states x_1 and x_2 are equivalent. The obvious choice for an equivalence would be FT-behavioural equivalence. However, this equivalence is not exactly what we are looking for. In the motivating example of non-deterministic automata we wanted two states to be equivalent if they recognize the same language. If we would take the equivalence arising from the functor $2 \times \mathcal{P}_\omega(Id)^A$ we would be distinguishing states that recognize the same language but have difference branching types, as in the following example.

$$\begin{tikzcd}
& \bullet \arrow[d, "a"] & & & & \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
& {} \arrow[dl, "b"'] \arrow[dr, "c"] & & & {} \arrow[d, "b"'] & & {} \arrow[d, "c"] \\
\bullet & & \bullet & & \bullet & & \bullet
\end{tikzcd}$$

We now define a new equivalence, which will *absorb* the effect of the monad T.

We say that two elements x_1 and x_2 in X are *F-equivalent with respect to a monad* $\mathbf{T}$, written $x_1 \approx_F^T x_2$, if and only if $\eta_X(x_1) \sim_F \eta_X(x_2)$. The equivalence $\sim_F$ is just F-behavioural equivalence for the F-coalgebra $f^\sharp\colon T(X) \to FT(X)$.

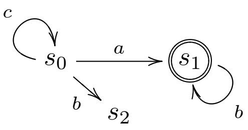
If the functor F has a final coalgebra (Ω, ω) , we can capture the semantic equivalence above in the following commuting diagram

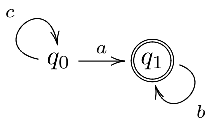
$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "f"'] & T(X) \arrow[r, dashed, "{[\![-]\!]}"] \arrow[dl, "f^\sharp"] & \Omega \arrow[d, "\omega"] \\
FT(X) \arrow[rr, dashed, "{F[\![-]\!]}"] & & F(\Omega)
\end{tikzcd} \tag{1}$$

Back to our first example, two states x_1 and x_2 of an NDA (in which T is instantiated to $\mathcal{P}_\omega$ and F to $2 \times Id^A$) would satisfy $x_1 \approx_F^T x_2$ if and only if they recognize the same language (recall that the final coalgebra of the functor $2 \times Id^A$ is 2^{A^*}).

It is also interesting to remark the difference between the two equivalences in the case of partial automata. The coalgebraic semantics of PAs [24] is given in terms of pairs of prefix-closed languages $\langle V, W\rangle$ where V contains the words that are accepted (that is, are the label of a path leading to a final state) and W contains all words that label any path (that is all that are in V plus the words labeling paths leading to non-final states). We exemplify what V and W would be in the following examples for state s_0 and q_0.

$$W = c^* + c^*b + c^*ab^* \qquad V = c^*ab^*$$





$$W = c^* + c^*ab^* \qquad V = c^*ab^*$$

Thus, state s_0 and q_0 would be distinguished by FT-equivalence (for $F = 2 \times Id^A$ and $T = 1 + -$) but they are equivalent with respect to the monad $1 + -$, $s_0 \approx_F^T q_0$, since they accept the same language.

We will show in Section 5 that the equivalence $\sim_{FT}$ is contained in $\approx_F^T$.

4.1 Examples

In this section we show more examples of applications of the framework above.

4.1.1 Partial Mealy machines

A partial Mealy machine is a set of states X together with a function $t\colon X \to (B \times (1+X))^A$, where A is a set of inputs and B is a set of output values (with a distinguished value $\perp$). For each state s and for each input a the automaton produces an output value and either terminates or continues to a next state. Applying the framework above we will be *totalizing* the automaton, similarly to what happened in the example of partial automata, by adding an extra state to the state space which will act as a sink state. The behaviour of the totalized automaton is given by the set of causal functions from A^ω (infinite sequences of A) to B^ω, which we denote by $\Gamma(A^\omega, B^\omega)$ [23]. A function $f\colon A^\omega \to B^\omega$ is causal if, for $\sigma \in A^\omega$, the n-th value of the output stream $f(\sigma)$ depends only on the first n values of the input stream σ.

$$\begin{tikzcd}
X \arrow[r] \arrow[d, "t"'] & 1 + X \arrow[r, dashed, "\llbracket - \rrbracket"] \arrow[dl, "t^\sharp"] & \Gamma(A^\omega, B^\omega) \arrow[d] \\
(B \times (1+X))^A \arrow[rr, dashed] & \begin{array}{l} \llbracket \kappa_1(*) \rrbracket(\sigma) = (\perp, \perp, \ldots) \\ \llbracket \kappa_2(s) \rrbracket(a{:}\sigma) = b{:}(\llbracket n \rrbracket(\sigma)) \\ \text{where } \langle b, n\rangle = t(s)(a) \end{array} & (B \times \Gamma(A^\omega, B^\omega))^A
\end{tikzcd}$$

4.1.2 Structured Moore automata

In the following examples we look at the functor $F(X) = T(B) \times X^A$, for B and A arbitrary sets and $\mathbf{T} = (T, \eta, (-)^\sharp)$ an arbitrary monad. This represents Moore automata with outputs in $T(B)$ and inputs in A. For any set X, $FT(X)$ has a $\mathbf{T}$-algebra lifting and the final coalgebra of F is $T(B)^{A^*}$. The final map $\llbracket - \rrbracket\colon T(X) \to T(B)^{A^*}$ is defined below.

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "\langle o,t\rangle"'] & T(X) \arrow[r, dashed, "\llbracket - \rrbracket"] \arrow[dl, "\langle o,t\rangle^\sharp"] & T(B)^{A^*} \arrow[d, "\langle \epsilon, (-)_a\rangle"] \\
T(B) \times (T(X))^A \arrow[rr, dashed] & \begin{array}{l} \llbracket m \rrbracket(\epsilon) = o^\sharp(m) \\ \llbracket m \rrbracket(aw) = \llbracket t^\sharp(m)(a) \rrbracket(w) \end{array} & T(B) \times (T(T(B)^{A^*}))^A
\end{tikzcd}$$

4.1.2.1 Moore automata with exceptions

Consider $T(X) = E + X$, with E a set of exceptions, $\eta(x) = \kappa_2(x)$ and, for a function $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is defined as $f^\sharp = [id, f]$.

An FT-coalgebra $\langle o, t\rangle\colon X \to (E+B) \times (E+X)^A$ will associate with every state s an output value (either in B or an exception in E) and, for each input a, a next state or an exception. The behaviour of a state x, given by $[\![\eta(x)]\!]$, will be a formal power series over A with output values in $E+B$ (that is, a function from A^* to $E+B$), defined as follows:

$$[\![\kappa_1(e)]\!](w) = \kappa_1(e) \qquad [\![\kappa_2(s)]\!](\epsilon) = o(s) \qquad [\![\kappa_2(s)]\!](aw) = [\![t(s)(a)]\!](w).$$

4.1.2.2 Moore automata with side effects

Consider $T(X) = (S \times X)^S$, with S a set of side-effects, $\eta(x) = \lambda s.\langle s, x\rangle$ and, for a function $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is defined as $f^\sharp(g)(s) = f(x)(s')$ where $\langle s', x\rangle = g(s)$.

Take now an FT-coalgebra $\langle o, t\rangle\colon X \to (B \times S)^S \times ((S \times X)^S)^A$ and let us explain the intuition behind this automaton type. Let S be the set of side effects (for instance, one could take $S = V^L$, functions associating memory locations to values). The set $S \times X$ can be interpreted as the configurations of the automaton, where S contains information about the state of the system and X about the control of the system. Then, we can think of $o\colon X \to (S \times B)^S$ as a function that for each configuration $S \times X$ provides an output and the new state of the system (note that $X \to (S \times B)^S \cong S \times X \to S \times B$). The transition function $t\colon X \to ((S \times X)^S)^A$ gives a new configuration for each input letter and current configuration (again we use the fact that $X \to ((S \times X)^S)^A \cong S \times X \to (S \times X)^A$).

The behaviour of a state x will be given by $[\![\eta(x)]\!]$, defined below, and it will be a function that for each configuration and for each sequence of actions returns an output value and a side effect.

$$\begin{array}{rcl} [\![g]\!](\epsilon)(s) & = & o(x)(s') \text{ where } \langle s', x\rangle = g(s) \\ [\![g]\!](aw_1) & = & [\![\lambda s.t(s)(a)(s')]\!](w_1) \text{ where } \langle s', x\rangle = g(s) \end{array}$$

4.1.2.3 Moore automata with interactive output

Consider $T(X) = \mu v.X + (O \times v) \cong O^* \times X$, with O a set of outputs, $\eta(x) = \langle \epsilon, x\rangle$ and, for $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is given by $f^\sharp(\langle w, x\rangle) = \langle ww', x'\rangle$ where $\langle w', x'\rangle = f(x)$. Take an FT-coalgebra $\langle o, t\rangle\colon X \to (O^* \times B) \times (O^* \times X)^A$. For $B = 1$, this coincides with a *(total) subsequential transducer* [8]: $o\colon X \to O^*$ is the terminal output function; $t\colon X \to (O^* \times X)^A$ is the pairing of the output function and the next state-function.

The behaviour of a state x will be given by $[\![\eta(x)]\!] = [\![\langle \epsilon, x\rangle]\!]$, where, for every $\langle w, x\rangle \in O^* \times X$, $[\![\langle w, x\rangle]\!]\colon A^* \to B^*$, is given by

$$[\![\langle w, x\rangle]\!](\epsilon) \;=\; w \cdot o(x) \qquad\qquad [\![\langle w, x\rangle]\!](aw_1) \;=\; w \cdot ([\![t(x)(a)]\!](w_1))$$

4.1.2.4 Probabilistic Moore automata

Take $T(X) = \mathcal{D}_\omega(X)$, η the Dirac distribution (defined below) and, for $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is given by

$$f^\sharp(c) = \lambda y. \sum_{d \in \mathcal{D}_\omega(Y)} \left(\sum_{x \in f^{-1}(d)} c(x) \right) \times d(y) \qquad\qquad \eta(x) = \lambda x'. \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases}$$

Take an FT-coalgebra $\langle o, t\rangle \colon X \to \mathcal{D}_\omega(B) \times \mathcal{D}_\omega(X)^A$. For $B = 2$ (note that $\mathcal{D}_\omega(2) \cong [0,1]$) this gives rise to a *(Rabin) probabilistic automaton* [21]: each state x has an output value in $o(x) \in [0,1]$ and, for each input a, $t(x)(a)$ is a probability distribution of next states. The behaviour of a state x is given by $[\![\eta(x)]\!] \colon A^* \to [0,1]$, defined below. Intuitively, one can think of $[\![\eta(x)]\!]$ as a probabilistic language: each word is associated with a value $p \in [0,1]$.

$$\begin{array}{lcl} [\![d]\!](\epsilon) & = & \sum\limits_{b\in[0,1]} (\sum\limits_{o(x)=b} d(x)) \times b \\ [\![d]\!](aw) & = & [\![\lambda x'. \sum\limits_{c\in\mathcal{D}_\omega(X)} (\sum_{b=t(x)(a)} d(x)) \times c(x')]\!](w) \end{array}$$

It is worth to note that this exactly captures the semantics of [21], while the ordinary $\sim_{FT}$ coincides with *probabilistic bisimilarity* of [14].

5 Coalgebras and T-Algebras

In the previous section we presented a framework, parameterized by a functor F and a monad $\mathbf{T}$, in which systems of type FT (that is, FT-coalgebras) can be studied using a novel equivalence $\approx_F^T$ instead of the classical $\sim_{FT}$. The only requirement we imposed was that $FT(X)$ has to be a $\mathbf{T}$-algebra.

In this section, we will present functors F for which the requirement of $FT(X)$ being a $\mathbf{T}$-algebra is guaranteed because they can be *lifted* to a functor F^* on $\mathbf{T}$-algebra. For these functors, the equivalence $\approx_F^T$ coincides with $\sim_{F^*}$. In other words, working on FT-coalgebras in **Set** under the novel $\approx_F^T$ equivalence is the same as working on F^*-coalgebras on $\mathbf{T}$-algebras under the ordinary $\sim_{F^*}$ equivalence. Next, we will prove that for this class of functors and an arbitrary monad $\mathbf{T}$ the equivalence $\sim_{FT}$ is contained in $\approx_F^T$. Instantiating this result for our first motivating example of non-deterministic automata will yield the well known fact that bisimilarity implies trace equivalence.

Let $\mathbf{T}$ be a monad. An endofunctor $F^* \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}^{\mathbf{T}}$ is said to be the $\mathbf{T}$-*algebra lifting* of a functor $F \colon \mathbf{Set} \to \mathbf{Set}$ if the following square commutes[1]:

$$\begin{tikzcd} \mathbf{Set}^{\mathbf{T}} \arrow[r, "F^*"] \arrow[d, "U^{\mathbf{T}}"'] & \mathbf{Set}^{\mathbf{T}} \arrow[d, "U^{\mathbf{T}}"] \\ \mathbf{Set} \arrow[r, "F"] & \mathbf{Set} \end{tikzcd}$$

If the functor F has a $\mathbf{T}$-algebra lifting F^* then $FT(X)$ is the carrier of the algebra $F^*(T(X), \mu)$. Functors that have a $\mathbf{T}$-algebra lifting are given, for example, by those endofunctors on **Set** constructed inductively by the following grammar

$$F ::= Id \mid B \mid F \times F \mid F^A \mid TG$$

where A is an arbitrary set, B is the constant functor mapping every set X to the carrier of a $\mathbf{T}$-algebra (B, h), and G is an arbitrary functor. Since the forgetful functor $U^{\mathbf{T}} \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ creates and preserves limits, both $F_1 \times F_2$ and F^A have a $\mathbf{T}$-algebra lifting if F, F_1, and F_2 have. Finally, TG has a $\mathbf{T}$-algebra lifting for every endofunctor G given by the assignment $(X, h) \mapsto (TGX, \mu_{GX})$. Note that we do not allow taking coproducts in the above grammar, because coproducts of $\mathbf{T}$-algebras are not preserved in general by the forgetful functor $U^{\mathbf{T}}$. Instead, one could resort to extending the grammar with the carrier of the coproduct taken

[1] This is equivalent to the existence of a distributive law $\lambda \colon TF \Rightarrow FT$ [12].

directly in $\mathbf{Set}^{\mathbf{T}}$. For instance, if $\mathbf{T}$ is the (finite) powerset monad, then we could extend the above grammar with the functor $F_1 \oplus F_2 = F_1 + F_2 + \{\top, \bot\}$.

Now, let F be a functor with a $\mathbf{T}$-algebra lifting and for which a final coalgebra Ω exists. If Ω can be constructed as the limit of the final sequence (for example assuming the functor accessible [1]), then, because the forgetful functor $U^{\mathbf{T}} \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ preserves and creates limits, Ω is the carrier of a $\mathbf{T}$-algebra, and it is the final coalgebra of the lifted functor F^*. Further, for any FT-coalgebra $f \colon X \to FT(X)$, the unique F-coalgebra homomorphism $[\![-]\!]$ as in diagram (1) is a T-algebra homomorphism between $T(X)$ and Ω. Conversely, the carrier of the final F^*-coalgebra (in $\mathbf{Set}^{\mathbf{T}}$) is the final F-coalgebra (in $\mathbf{Set}$).

Intuitively, the above means that for an accessible functor F with a $\mathbf{T}$-algebra lifting F^*, F^*-equivalence in $\mathbf{Set}^{\mathbf{T}}$ coincides with F-equivalence with respect to $\mathbf{T}$ in $\mathbf{Set}$. The latter equivalence is coarser than the FT-equivalence in $\mathbf{Set}$, as stated in the following theorem.

▶ **Theorem 1.** *Let $\mathbf{T}$ be a monad. If F is an endofunctor on* **Set** *with a $\mathbf{T}$-algebra lifting, then $\sim_{FT}$ implies $\approx_F^T$.*

The proof of this theorem (presented in [27]) relies on the fact that for every monad $\mathbf{T}$ and functor F with a $\mathbf{T}$-algebra lifting, if $h \colon (X, f) \to (Y, g)$ is an FT-coalgebra homomorphism, then $(\eta_Y \circ h)^\sharp \colon (T(X), f^\sharp) \to (T(Y), g^\sharp)$ is an F-coalgebra homomorphism.

The above theorem instantiates to the well-known facts: for NDA, where $F(X) = 2 \times X^A$ and $T = \mathcal{P}_\omega$, that bisimulation implies language equivalence; for partial automata, where $F(X) = 2 \times X^A$ and $T = 1 + -$, that equivalence of pairs of languages, consisting of defined paths and accepted words, implies equivalence of accepted words; for probabilistic automata, where $F(X) = [0,1] \times X^A$ and $T = \mathcal{D}_\omega$, that probabilistic bisimilarity implies probabilistic/weighted language equivalence. Note that, in general, the above inclusion is strict.

6 Discussion

In this paper, we lifted the powerset construction on automata to the more general framework of FT-coalgebras. Our results lead to a uniform treatment of several kinds of existing and new variations of automata (that is, FT-coalgebras) by an algebraic structuring of their state space through a monad T. We showed as examples partial Mealy machines, structured Moore automata, nondeterministic, partial and probabilistic automata. The technical report [27] shows (as further examples) several behavioural equivalences that are extremely interesting for the theory of concurrency. It is worth mentioning that the framework instantiates to many other examples, among which *weighted automata* [26]. These are simply structured Moore automata for $B = 1$ and $\mathbf{T} = \mathbb{S}_\omega^-$ (for a semiring $\mathbb{S}$) [7]. It is easy to see that $\sim_{FT}$ coincides with weighted bisimilarity [5], while $\approx_F^T$ coincides with weighted language equivalence [26].

Some of the aforementioned examples can also be coalgebraically characterized in the framework of [9]. There, instead of considering FT-coalgebras on $\mathbf{Set}$ and F^*-coalgebras on $\mathbf{Set}^{\mathbf{T}}$ (the Eilenberg-Moore category), TG-coalgebras on $\mathbf{Set}$ and $\overline{G}$-coalgebras on $\mathbf{Set}_{\mathbf{T}}$ (the *Kleisli* category) are studied. The main theorem of [9] states that under certain assumptions, the initial G-algebra is the final $\overline{G}$-coalgebra that characterizes (generalized) trace equivalence. In [27], we present a first step in exploring the connection between both frameworks. However, the exact relationship is not clear yet and further research is needed in order to make it precise. It is worth to remark that many of our examples will not fit the framework in [9]: for instance, the exception, the side effect, the full-probability and the interactive output monads do not fulfill their requirements (the first three do not have a bottom element and the latter is not commutative). Moreover, we also note that the example of partial Mealy machines is not purely trace-like, as all the examples in [9].

There are two other future research directions. On the one hand, we will try to exploit *F-bisimulations up to T* [15, 16] as a sound and complete proof technique for $\approx_F^T$. On the other hand, we would like to lift many of those coalgebraic tools that have been developed for "branching equivalences" (such as coalgebraic modal logic [6, 25] and (axiomatization for) regular expressions [3]) to work with the "linear equivalences" induced by $\approx_F^T$.

References

1 J. Adámek. Free algebras and automata realization in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

2 Falk Bartels. *On generalized coinduction and probabilistic specification formats.* PhD thesis, Vrije Universiteit Amsterdam, 2004. PhD thesis.

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

4 S. D. Brookes, C. A. R. Hoare and A. W. Roscoe. A Theory of Communicating Sequential Processes. *J. ACM.*, 31(3):560–599, 1984.

5 P. Buchholz. Bisimulation relations for weighted automata. *TCS*, 393(1-3):109–123, 2008.

6 C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. In Erol Gelenbe, Samson Abramsky, and Vladimiro Sassone, editors, *BCS Int. Acad. Conf.*, pages 128–140. British Computer Society, 2008.

7 H. P. Gumm and T. Schröder. Monoid-labeled transition systems. *ENTCS*, 44(1), 2001.

8 H.H. Hansen. Coalgebraising subsequential transducers. *ENTCS*, 203(5):109–129, 2008.

9 I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.

10 C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM.*, 21(8):666–677, 1978.

11 Bart Jacobs. Distributive laws for the coinductive solution of recursive equations. Inf. Comput. 204(4): 561-587 (2006)

12 P.T. Johnstone. Adjoint lifting theorems for categories of algebras. *Bulletin London Mathematical Society*, 7:294–297, 1975.

13 C. Jou and S. A. Smolka. Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes. In Proc. of CONCUR, *LNCS*, 458:367–383. Springer, 1990.

14 K. G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

15 M. Lenisa. From Set-theoretic Coinduction to Coalgebraic Coinduction: some results, some problems. *ENTCS*, 19, 1999.

16 M. Lenisa, J. Power and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *ENTCS*, 33, 2000.

17 E.Manes. Algebraic theories. *Graduate Texts in Mathematics*, 26, 1976.

18 E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

19 L. Monteiro. A Coalgebraic Characterization of Behaviours in the Linear Time - Branching Time Spectrum. In Proc. of WADT, *LNCS*, 5486:128–140. Springer, 2009.

20 E.-R. Olderog and C. A. R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Inf.*, 21(1):9–66, 1986.

21 M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

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

23 J. J. M. M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. *ENTCS*, 160:305–319, 2006.

24 J.J.M.M. Rutten. Coalgebra, concurrency, and control. In R. Boel and G. Stremersch, editors, *Proceedings of WODES 2000*, pages 31–38, 2000.

25 L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *TCS*, 390(2-3):230–247, 2008.

26 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

27 A. Silva, F. Bonchi, M. Bonsangue and J. Rutten. Generalizing the powerset construction, coalgebraically. Technical Report. Nr. SEN-1008, September 2010, CWI.