# Coalgebraic Determinization of Alternating Automata

Report on a M1 Internship

Meven Bertrand, under the supervision of Jurriaan Rot

September 25, 2017

**Abstract**

Coalgebra is a currently quite active field, which aims to look at generic state-based systems (most prominently automata) from a very abstract point of view, mainly using tools from category theory. One of its achievements is to give a completely generic approach of determinization, unifying in an elegant manner non-deterministic automata, probabilistic automata or non-deterministic pushdown automata in one and the same model.

However, the case of alternating automata fails to easily fit in this model. The aim of this internship was therefore to tackle this problem: can alternating automata also be determinized in the coalgebraic way? Does this give semantics that coincides with the concretely defined one?

In this report, we give a positive answer to both questions. The main element of our construction is a distributive law, the definition of which has been for some time an open question.

## Contents

# 1 Introduction

A big part of computer science is about studying models of computation, for various purposes. A lot of these models (automata, Turing machines, stream systems, Mealy machines, etc.) share a common structure: they consist of a set of states, together with some kind of transition structure. The "user" does not have access to the states, but only to some output: the system behaves like a black box.

Coalgebra is a field that uses tools coming from category theory to describe systems of this kind in a generic way. Its biggest achievement is the notion of bisimulation, a sound reasoning principle for behavioural equivalence that exists for any of these state-based systems as soon as they are described in the coalgebraic setting.

However, this notion of bisimulation will not be our main concern here. Rather, we will look at another quite interesting construction: determinization. A lot of state based systems include some kind of branching structure: it can be choice between multiple transitions (non-deterministic automata, non-deterministic Turing machines, and so forth), but also probabilistic transitions (probabilistic automata), and even more exotic models (weighted automata, alternating automata, etc.). In most of these cases, one can construct a deterministic version of the system, by making the state space bigger. Thanks to the coalgebraic setting, this determinization is now understood at an abstract level, and can be performed on generic non-deterministic systems, as surveyed in [6].

However, there are a few cases that somehow do not easily fit in this setting. The case of alternating automata is one, and it has resisted attempts to describe it coalgebraically for some time. In this report we give a coalgebraic description of alternating automata in the setting of determinization. This finally solves the aforementioned difficulty, and makes alternating automata another example of the power of the coalgebraic theory. Moreover, this definition gives a semantics that is similar to the concrete one, which makes it very satisfactory.

The crucial point of this determinization is a categorical construction called distributive law. The law itself was already known, but it was used in a completely different setting. Thus, our work was mostly to acknowledge that this law could be used in our setting, and construct the frame around it to make it fit into the general picture of determinization. After this, we still had to verify that the semantics corresponding to this construction is indeed the usual, concrete one.

**Literature review** The search for this distributive law has caused a lot of mistakes and unsatisfactory trials: in [5] and [6], the problem of alternating automata is tackled, but each time the model is less interesting than the original one; concerning the law itself, a construction that is erroneously claimed to be a distributive law appears in different papers, for instance [8] (corrected in [9]), or [11]. The case of [9] is rather interesting, as they use an imperfect law, and are still able to draw some results out of it, but this is not enough to really solve the problem. A precise description of the errors can be found in [7]. The use of our the distributive law was suggested by Luigi Santocanale. It seems to be somewhat a folklore result, so it is hard to trace back, but it can for example be found in [12, p. 220-221], where it is just a simple example of a much more complex construction.

**Outline** The report is divided in three parts: the first one (section 2) gives some generic categorical notions, the second one (section 3) introduces the needed notions of coalgebra, including the generic determinization procedure, and the last one (section 4) exposes the special case of alternating automata. This last section is the one containing the original work, the two others are mostly there to give context. At the end of the report, we include unpublished notes to which we refer a few times: [7].

# 2 Category Theory

In this section, we give a small overview of category theory, with the aim of introducing the main subjects the internship was dealing with. Since we mean it to be an overview, and not a proper course of any kind, we will not give proofs, but just state the interesting properties, and try to give intuition on the reason they are true. Complete definitions and proofs can be found in [1].

## 2.1 Basic Definitions

In almost every field of mathematics, one is concerned with a certain type of objects, and with mappings that preserve these objects: sets and functions, vectors spaces and linear maps, groups and group homomorphisms, ordered sets and monotone maps, topological spaces and continuous functions, and so on. This is what category theory tries to define, on a very abstract level.

**Definition 2.1 (Category)**
A category consists of

- a collection of objects ($A$, $B$, $C$ ...),
- a collection of arrows ($f$, $g$, $h$ ...),

and the following are also given

- for each arrow, two objects are given, its domain and codomain, written respectively as $\mathrm{dom}(f)$ and $\mathrm{codom}(f)$, and we usually write $f : A \to B$ to indicate that $\mathrm{dom}(f) = A$ and $\mathrm{codom}(f) = B$,
- given any three objects $A$, $B$ and $C$ and arrows $f : A \to B$ and $g : B \to C$, an arrow $g \circ f : A \to C$ is given, called the composite of $f$ and $g$,
- for each object, an arrow $\mathrm{id}_A$ is given, called the identity of $A$,

and these must satisfy

- associativity: $h \circ (g \circ f) = (h \circ g) \circ f$ whenever the composition is defined,
- unit: for any arrow $f : A \to B$, we have $\mathrm{id}_B \circ f = f = f \circ \mathrm{id}_A$.

All the examples mentioned above are categories according to this definition. Some much weirder constructions are also categories, but we will not present these, as they are of little interest for this report.

We will mostly work in the following category:

**Definition 2.2 (Category of sets)**
The sets (as objects) and functions (as arrows), with the usual compositions and identities, form a category, that we will denote by **Set**.

Another category we will make use of is the following:

**Definition 2.3 (Category of posets)**
A poset (partially ordered set) is a set together with an order relation (transitive, reflexive and antisymmetric relation).
A monotone map $f$ between two posets $(X, \leq)$ and $(Y, \preceq)$ is a function $f : X \to Y$ such that

$$\forall\, x, x' \in X,\ x \leq x' \Rightarrow f(x) \preceq f(x')$$

The posets and monotone maps (with the usual composition and identity) form a category, that we will denote as **Poset**.

Of course, we want maps between categories. This is possible using the next definition.

**Definition 2.4 (Functor)**
Given two categories $\mathbb{C}$ and $\mathbb{D}$, a functor $\mathcal{F} :\ \mathbb{C} \to \mathbb{D}$ is a mapping from objects of $\mathbb{C}$ to objects of $\mathbb{D}$ and from arrows of $\mathbb{C}$ to arrows of $\mathbb{D}$ that preserves domains, codomains, composition and identities, that is:

- for any arrow $f : A \to B$ in $\mathbb{C}$, $\mathcal{F}(f) :\ \mathcal{F}(A) \to \mathcal{F}(B)$,
- for any two arrows $f : A \to B$ and $g : B \to C$, $\mathcal{F}(g \circ f) = \mathcal{F}(g) \circ \mathcal{F}(f)$,
- for any object $A$, $\mathcal{F}(\mathrm{id}_A) = \mathrm{id}_{\mathcal{F}(A)}$.

This definition can apply to an extremely large amount of contexts and usual constructions, and this is the main reason why category theory is so handy in formulating coalgebra, as we will see in section 3.1.

Here are some examples that we will use later in this report.

**Example 2.1 (Product as a functor)**
Given a set $X$, one can define the Cartesian product with $X$ as a functor $\mathcal{F}$ in the category **Set**:

- on objects, we define $\mathcal{F}(Y) = X \times Y$,
- on arrows, we define $\mathcal{F}(f)(x, y) = (x, f(y))$, that is, $\mathcal{F}(f)$ is the identity on the $X$ component, and $f$ on the other component.

One can easily check that this fits into the definition above. This functor will be denoted as $X \times (-)$ later in this report.

**Example 2.2 (Exponential as a functor)**
Given a set $X$, one can define the exponential with respect to $X$ as a functor $\mathcal{F}$ in the category **Set**:

- on objects, we define $\mathcal{F}(Y) = Y^X$, the set of functions from $X$ to $Y$,
- on arrows, we define $\mathcal{F}(f)(g) = f \circ g$, that is $\mathcal{F}(f)$ is the post-composition with $f$.

Again, it is easy to check that this fits into the definition above. This functor will be denoted as $(-)^X$ later in this report.

**Example 2.3 (Identity functor)**
Given a category $\mathbb{C}$, there is a so-called identity functor on $\mathbb{C}$, written as $\mathrm{id}_{\mathbb{C}}$, defined as the identity both on objects and arrows.

**Example 2.4 (Powerset functor)**
The usual powerset construction can be seen as a functor $\mathcal{P} : \mathbf{Set} \to \mathbf{Poset}$, defined as follows:

- on objects, we define $\mathcal{P}(X)$ as the set of all subsets of $X$, with the order given by inclusion,
- on arrows, we define $\mathcal{P}(f)(P) = \{f(x), x \in P\}$, that is $\mathcal{F}(f)$ takes the direct image of a subset by $f$.

If one forgets about the order structure, one can also see the same $\mathcal{P}$ as a functor $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$. In the rest of the report, we will abuse notation and make no distinction between both functors, as the correct type can be inferred from the context.

**Example 2.5 (Forgetful functor)**
A large and very useful class of functors is the class of forgetful functors, that is functors between categories that "forget" some of the structure. For instance, we can look at the category **Poset**. Its objects are ordered sets, but one can "forget" that these objects have some order structure, and rather look at them purely as sets. That is what the following forgetful functor $\mathcal{U}$ does:

- on objects, we define $\mathcal{U}((X, \leq)) = X$,

- on arrows, we define $\mathcal{U}(f) = f$ (but $f$ which was a monotone function is now considered as a mere function between sets).

A last definition, which characterize to objects that are in a certain sense "universal".

**Definition 2.5 (Final object)**
A final object of a category $\mathbb{C}$ is an object $\mathbf{1}$ of $\mathbb{C}$ such that for any other object $A$ of $\mathbb{C}$, there exists a unique arrow $!_A : A \to \mathbf{1}$.

## 2.2 More Advanced Constructions

Until here, the constructions we defined were rather simple, and though they may be useful as such, the real interest in category theory comes from more complex constructions. These constructions are also the ones that are at the core of the coalgebraic setting we will work in, mostly the notion of monad, that we will use to encapsulate non-determinism.

The definitions will be given fully and illustrated with examples, in order to give some grasp on them, but the details of the definitions are not needed to understand the rest of the report past this section.

**Definition 2.6 (Natural transformation)**
Given two functors $\mathcal{F} : \mathbb{C} \to \mathbb{D}$ and $\mathcal{G} : \mathbb{C} \to \mathbb{D}$, a natural transformation $\lambda : \mathcal{F} \Rightarrow \mathcal{G}$ is a family of arrows (in $\mathbb{D}$) $\lambda_X : \mathcal{F}(X) \to \mathcal{G}(X)$ (one for each object $X$ of $\mathbb{C}$) such that for any arrow $f : X \to Y$ in $\mathbb{C}$, the following diagram commutes:

$$\begin{array}{ccc} \mathcal{F}(X) & \xrightarrow{\lambda_X} & \mathcal{G}(X) \\ {\scriptstyle \mathcal{F}(f)}\downarrow & & \downarrow{\scriptstyle \mathcal{G}(f)} \\ \mathcal{F}(Y) & \xrightarrow[\lambda_Y]{} & \mathcal{G}(Y) \end{array}$$

that is, such that $\lambda_Y \circ \mathcal{F}(f) = \mathcal{G}(f) \circ \lambda_X$.

The idea of a natural transformation is that it is a transformation between two functors that can be indifferently applied before of after a function application. In a sense, it transforms $\mathcal{F}(X)$ into $\mathcal{G}(X)$ without looking at the content of $X$, but only at the structure given by $\mathcal{F}$ and $\mathcal{G}$.

**Example 2.6 (Cartesian Product)**
Recall the functor $X \times (-)$ that we already defined on **Set**. We could also define in a very similar way the functor $(-) \times X$. Then there is a natural transformation $\lambda : X \times (-) \Rightarrow (-) \times X$ between these two functors, defined by the components:

$$\begin{array}{rccc} \lambda_Y : & X \times Y & \to & Y \times X \\ & (x, y) & \mapsto & (y, x) \end{array}$$

This notion of natural transformation is already useful as such, but it is also a building element in a lot of other categorical notions, such as the one of monad.

**Definition 2.7 (Monad)**
A monad is a functor $\mathcal{T}$ from a category $\mathbb{C}$ to itself, together with two natural transformations $\eta : \mathrm{id} \Rightarrow \mathcal{T}$ and $\mu : \mathcal{T} \circ \mathcal{T} \to \mathcal{T}$ such that the following diagrams commute for any object $X$:

$$\begin{array}{ccc} \mathcal{T}(X) & \xrightarrow{\eta_{\mathcal{T}(X)}} & \mathcal{T}(\mathcal{T}(X)) \\ {\scriptstyle \mathcal{T}(\eta_X)}\downarrow & \searrow^{\mathrm{id}_X} & \downarrow{\scriptstyle \mu_X} \\ \mathcal{T}(\mathcal{T}(X)) & \xrightarrow[\mu_X]{} & \mathcal{T}(X) \end{array} \qquad \begin{array}{ccc} \mathcal{T}(\mathcal{T}(\mathcal{T}(X))) & \xrightarrow{\mathcal{T}(\mu_X)} & \mathcal{T}(\mathcal{T}(X)) \\ {\scriptstyle \mu_{\mathcal{T}(X)}}\downarrow & & \downarrow{\scriptstyle \mu_X} \\ \mathcal{T}(\mathcal{T}(X)) & \xrightarrow[\mu_X]{} & \mathcal{T}(X) \end{array}$$

that is we have $\mu_X \circ \eta_{\mathcal{T}(X)} = id_X = \mu_X \circ \mathcal{T}(\eta_X)$ and $\mu_X \circ F(\mu_X) = \mu_X \circ \mu_{F(X)}$.
The transformation $\eta$ is called the unit of the monad, and the transformation $\mu$ is called its multiplication.

The idea is that a monad is some kind of structure, along with a way to put an object into this structure (the unit), and a way to collapse two levels of the structure together. These must respect some easy rules: if you use the unit to put a second level, and collapse this level, then you get the thing you started with, and if you collapse three levels into one, you can collapse the outer most or inner most two first, with the same result.

**Example 2.7 (Powerset as a monad)**
We already described the powerset as a functor $\mathcal{P}$ from the category **Set** to itself. This functor can be made into a monad, by taking:

- the unit $\begin{array}{rccc} \eta_X : & X & \to & \mathcal{P}(X) \\ & x & \mapsto & \{x\} \end{array}$,
- the multiplication $\begin{array}{rccc} \mu_X : & \mathcal{P}(\mathcal{P}(X)) & \to & \mathcal{P}(X) \\ & S & \mapsto & \cup S = \cup_{s \in S} s \end{array}$ (that is, take a set of sets to their union).

The last important definition is the one of adjoints. This is a quite difficult notion to grasp, although it is very powerful by itself. However, in this report we will mostly be interested in their link with monads, as adjunctions are a usual way to construct monads.

**Definition 2.8 (Adjoints)**
Two functors $\mathcal{F} : \mathbb{C} \to \mathbb{D}$ and $\mathcal{G} : \mathbb{D} \to \mathbb{C}$ are said to be adjoints (this is written as $\mathcal{F} \dashv \mathcal{G}$, $\mathcal{F}$ is called the left adjoint and $\mathcal{G}$ is called the right adjoint) if there are two natural transformation $\eta : \mathrm{id}_{\mathbb{C}} \Rightarrow \mathcal{G} \circ \mathcal{F}$ and $\varepsilon : \mathcal{F} \circ \mathcal{G} \Rightarrow \mathrm{id}_{\mathbb{D}}$ such that for any objects $C$ of $\mathbb{C}$ and $D$ of $\mathbb{D}$, we have

$$\mathcal{G}(\varepsilon_D) \circ \eta_{\mathcal{G}(D)} = \mathrm{id}_{\mathcal{G}(D)}$$

and

$$\varepsilon_{\mathcal{F}(C)} \circ \mathcal{F}(\eta_C) = \mathrm{id}_{\mathcal{F}(C)}$$

that is, such that the two following diagrams commute:

$$\begin{array}{ccc} \mathcal{G}(D) & \xrightarrow{\eta_{\mathcal{G}(D)}} & \mathcal{G}(\mathcal{F}(\mathcal{G}(D))) \\ & \searrow^{\mathrm{id}_{\mathcal{G}(D)}} & \downarrow{\scriptstyle \mathcal{G}(\varepsilon_D)} \\ & & \mathcal{G}(D) \end{array} \qquad \begin{array}{ccc} \mathcal{F}(C) & \xrightarrow{\mathcal{F}(\eta_C)} & \mathcal{F}(\mathcal{G}(\mathcal{F}(C))) \\ & \searrow^{\mathrm{id}_{\mathcal{F}(C)}} & \downarrow{\scriptstyle \varepsilon_{\mathcal{F}(C)}} \\ & & \mathcal{F}(C) \end{array}$$

There are many equivalent ways to define adjunctions. This one is good for our purposes, as it emphasizes the role of the unit and counit, which we will use in section 4.1 to define a monad. Although this definition might not seem of importance, adjoints, together with monads, form two of the most interesting objects of category theory: a lot of mathematical interesting propositions and definitions can be seen as instances of adjunctions, be it formal polynomials, quantifiers, interior and closure operators in topology, free objects over a set...

Here is an example of adjunction, of which we will make use later to construct a monad.

**Definition 2.9 (Discrete order functor)**
The functor $\mathcal{O}$ is the functor from the category **Set** to the category **Poset** that maps

- a set $X$ to the discrete order on $X$, that is $\mathcal{O}(X) = (X, =)$ (in the discrete order, the only comparable elements are the ones that are equal),
- an arrow $f$ to itself.

Note that if $f : X \to Y$ is a function in **Set**, then $f$ is also a monotone function between $\mathcal{O}(X)$ and $\mathcal{O}(Y)$, so that this definition makes sense.

**Proposition 2.1 (Adjunction for $\mathcal{O}$)**
There is an adjunction $\mathcal{O} \dashv \mathcal{U}$ between the discrete order functor we just defined and the forgetful functor $\mathcal{U} : \mathbf{Poset} \to \mathbf{Set}$ as defined in section 2.1.

Its unit is just the identity, and its counit is defined on a preorder $X = (S, \leq)$ by

$$\begin{array}{rccc} \varepsilon_X : & \mathcal{O}(\mathcal{U}(X)) & \to & X \\ & x & \mapsto & x \end{array}$$

so it is the identity of $S$, but it goes from the preorder $(S, =)$ to the preorder $(S, \leq)$.

# 3 Coalgebras And Determinization

In this section, we present the main subject of the internship, namely coalgebra and the view it gives on determinization. Once again, we do not provide the proofs of the facts we state, but these can be found in the papers we refer to.

## 3.1 State-Based Systems as Coalgebras

In this subsection we define our core construction: coalgebras. We also show how final objects can be used to give some sort of semantics. A more complete approach (with proofs!) can be found in [4].

**Definition 3.1 (Coalgebra)**
Given a category $\mathbb{C}$, a coalgebra for a functor $\mathcal{F} : \mathbb{C} \to \mathbb{C}$ consists of an object $X$ together with an arrow $f : X \to \mathcal{F}X$.

Coalgebras are meant to represent state-based system, where $X$ represents the states of the system, and $f$ is the transition function. Then the functor $\mathcal{F}$ characterizes the type of system we are looking at.

**Example 3.1 (Stream system)**
A stream system over an alphabet (set of letters) $A$ is a set of states $X$ together with two functions $o : X \to A$ (output) and $t : X \to X$ (transition). It is a coalgebra for the functor $A \times (-)$, the transition function being $\langle o, t \rangle : X \to A \times X$, where $\langle o, t \rangle$ is defined by: $\begin{array}{rccc} \langle o, t \rangle : & X & \to & A \times X \\ & x & \mapsto & (o(x), t(x)) \end{array}$

**Example 3.2 (Deterministic automaton)**
A deterministic automaton over an alphabet $A$ is a set of states $Q$, together with a function $o : X \to \mathbf{2}$ (where $\mathbf{2}$ is the two-element set $\{\mathbf{0}, \mathbf{1}\}$, returning $\mathbf{1}$ if the state is accepting and $\mathbf{0}$ otherwise) and a transition function $\delta : Q \to Q^A$. It is a coalgebra for the functor $\mathbf{2} \times (-)^A$, the transition function being $\langle o, \delta \rangle$ where $\chi_F$.

This way of looking at things is a bit unusual, but this is really the same as the usual definition of non-deterministic automata: the function $o$ corresponds to the characteristic function of the set of final states, and the function $\delta$ can be seen as a function of the type $X \times A \to X$, as for any $(x, a) \in X \times A$, $\delta(x)(a)$ is an element of $X$ (the $\delta$ of the coalgebraic definition is the curried version of the usual one).

Since we will reuse this $\mathbf{2} \times (-)^A$ functor, we will denote it as $\mathcal{D}$.

Note that in the above examples we are not taking into account any starting state. This is because the aim is really to describe a state-based **system**, not just one "computation" into it (for instance, the computation of a stream for a stream system, or of the acceptance or rejection of a word in an automaton).

**Definition 3.2 (Homomorphism of coalgebras)**
Given two coalgebras $(X, f)$ and $(Y, g)$ for a functor $\mathcal{F}$, a coalgebra homomorphism is an arrow $h : X \to Y$ such that the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \mathcal{F}X & \xrightarrow[\mathcal{F}h]{} & \mathcal{F}Y \end{array}$$

That is, we can either make a transition, and then use the arrow, or use the arrow first, and then make a transition.

With this we can construct a new category.

**Proposition 3.1**
Given a (fixed) functor, the coalgebras and homomorphism of coalgebras for this functor form a category.

**Definition 3.3 (Final coalgebra)**
Given a functor $\mathcal{F}$, a coalgebra $(Z, \xi)$ is called a final coalgebra if it is a final object in the category of coalgebras. This means that given a coalgebra $(X, f)$ for $\mathcal{F}$, there is a unique arrow $\mathrm{beh}_f$ (we might call it simply beh if there is no possible confusion) such that the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{beh}} & Z \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \xi} \\ \mathcal{F}X & \xrightarrow[\mathcal{F}\,\mathrm{beh}]{} & \mathcal{F}Z \end{array}$$

This definition is crucial, as it gives semantics for states: if we work in the category of sets, and $x$ is an element of $X$, then $\mathrm{beh}_f(x)$ is the semantics of the state $x$, as the following examples illustrate.

**Example 3.3 (Final stream system)**
Given an alphabet $A$, the coalgebra $(A^\omega, \langle o_z, t_z \rangle)$, where $o_z(s) = s(0)$ (the output is the first element of the stream) and $t_z(s)(i) = s(i+1)$ (the transition is a shift of the stream), is a final coalgebra for the functor $A \times (-)$ of stream systems. Moreover, if $(X, \langle o, t \rangle)$ is a stream system, then $\text{beh}(x) = (o(x), o(t(x)), o(t^2(x)), o(t^3(x)), \dots)$.

This corresponds to the stream one would naturally associate with $x$ in the stream system $(X, \langle o, t \rangle)$.

**Example 3.4 (Final deterministic automaton)**
Given an alphabet $A$, the coalgebra $(\mathbf{2}^{A^*}, \langle O, D \rangle)$ where $\mathbf{2}^{A^*}$ is the set of languages over $A$,

$$O(L) = \begin{cases} \mathbf{0} \text{ if } \varepsilon \notin L \\ \mathbf{1} \text{ if } \varepsilon \in L \end{cases}$$

with $\varepsilon$ denoting the empty word, and

$$D(L)(a) = \{w \in A^* \mid a \cdot w \in L\}$$

is a final coalgebra for $\mathbf{2} \times (-)^A$.

As expected, if $(Q, \langle o, \delta \rangle)$ is an automaton and $q$ is a state of $Q$, then $\text{beh}(q)$ is the language accepted by the automaton with starting state $q$, as defined usually:

- $\varepsilon \in \text{beh}(q)$ if and only if $o(q) = 1$,
- for any word $w$ and letter $a$, $a \cdot w \in \text{beh}(q)$ if and only if $w \in \text{beh}(\delta(q)(a))$.

## 3.2 The Problem Of Determinization

The main subject of this internship is the determinization procedure for an automaton. There is a well-known example, namely the one of non-deterministic automata. In that case, a new (deterministic) automaton is created, whose states are sets of states of the original automaton, and the semantics (in this case, the language denoted by the automaton) is preserved by the construction.

**Definition 3.4 (Non-deterministic automaton)**
Given an alphabet $A$, a non-deterministic automaton consists of a set of states $Q$ together with a function $o : Q \to \mathbf{2}$ and a transition function $\delta : Q \to \mathcal{P}(Q)^A$. It is a coalgebra for the functor $\mathbf{2} \times (\mathcal{P}(-))^A$.
A word $w$ is accepted by a state $q \in Q$ if

- $w = \varepsilon$ and $o(q) = 1$,
- $w = a \cdot w'$ and $w'$ is accepted by one of the states in $\delta(q)(a)$.

**Definition 3.5 (Determinization of an automaton)**
Given a non-deterministic automaton $(Q, o, \delta)$ over an alphabet $A$, its determinization is a deterministic automaton, with

- state set $\mathcal{P}(Q)$,
- accepting function $o^\#$ defined by $o^\#(S) = 1 \Leftrightarrow \exists\, s \in S,\ o(s) = 1$, a subset of $Q$ is accepting if it contains an accepting state,
- transition function $\delta^\# :\ \mathcal{P}(Q) \to \mathcal{P}(Q)^A$, defined by $\delta^\#(S)(a) = \cup_{q \in P} \delta(q)(a)$.

**Theorem 3.2 (Preservation of the semantics)**
A state $q$ of a non-deterministic automaton accepts a word if and only if the state $\{q\}$ of the determinization of the automaton accepts the word.

Following this motivating example, we wand to find a general way to perform the same kind of determinization. But first, we need to define what we wish to determinize.

**Definition 3.6 ($\mathcal{T}$-automaton)**
Given a monad $(\mathcal{T}, \eta, \mu)$ on the category of sets, a $\mathcal{T}$-automaton for an alphabet $A$ is a coalgebra for the functor $\mathbf{2} \times (\mathcal{T}(-))^A$.

Here the monad represents the "non-determinisic" component of the automaton, as the example of non-deterministic automata illustrates. The $\mathbf{2} \times (-)^A$, as we already saw, corresponds to the automaton structure.

**Example 3.5 (Non-deterministic automaton as a $\mathcal{P}$-automaton)**
A non-deterministic automaton is a $\mathcal{P}$-automaton, that is, a coalgebra for the functor $\mathbf{2} \times (\mathcal{P}(-))^A$.

But for instance the case of probabilistic automata also fits into this definition, with a proper monad (whose functorial component involves probability distributions).

The aim is then, given a monad $\mathcal{T}$ and a $\mathcal{T}$-automaton $(X, f)$, to find a semantics for $(X, f)$, that is a function from $X$ to the final coalgebra $\mathbf{2}^{A^*}$, that should arise "naturally" in some sense, and correspond with the concrete semantics that one already has on the motivating examples like the non-deterministic automata.

The next two sections provide two different ways to do this. The first one gives some semantics directly, without really resorting to determinization as such. The second one is the direct abstraction of the determinization of non-deterministic automata, in the sense that from a $\mathcal{T}$-automaton with state space $X$ it constructs a deterministic automaton with state space $\mathcal{T}(X)$.

## 3.3 Bialgebraic Semantics

The construction presented here can be found in all details in [5]. Its main ingredient is a so called Eilenberg-Moore algebra, that is an arrow $\beta : \mathcal{T}(\mathbf{2}) \to \mathbf{2}$ satisfying some axioms.

**Definition 3.7 (Eilenberg-Moore algebra)**
Given a monad $(\mathcal{T}, \eta, \mu)$, an Eilenberg-Moore algebra is an object $X$ together with an arrow $\beta : \mathcal{T}(X) \to X$ such that the following diagrams commute:

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & \mathcal{T}(X) \\ & \searrow^{\mathrm{id}_X} & \downarrow{\scriptstyle \beta} \\ & & X \end{array} \qquad \begin{array}{ccc} \mathcal{T}(\mathcal{T}(X)) & \xrightarrow{\mu_X} & \mathcal{T}(X) \\ {\scriptstyle \mathcal{T}(\beta)}\downarrow & & \downarrow{\scriptstyle \beta} \\ \mathcal{T}(X) & \xrightarrow[\beta]{} & X \end{array}$$

Given such an Eilenberg-Moore algebra, one can construct a new algebra on functions, using the so-called strength of the monad.

**Definition 3.8 (Strength)**
Given a monad $\mathcal{T}$, the strength $\mathrm{st} : \mathcal{T}(X^Y) \to \mathcal{T}(X)^Y$ is defined by $\mathrm{st}(f)(y) = \mathcal{T}(\lambda h \in X^Y \cdot h(y))(f)$.

Note that this construction only works because in the category **Set**, the function $\lambda h \in X^Y \cdot h(y)$ always exists, no matter what $y$ is chosen, which is not the case in all categories.

We will use this strength operator (which actually is a natural transformation $\mathrm{st} : \mathcal{T}(X^Y) \Rightarrow \mathcal{T}(X)^Y$) again later, as it is a very useful construction: it allows to put a monad "inside" a set of functions.

Now we can construct our algebra on functions.

**Proposition 3.3 (Pointwise Eilenberg-Moore algebra)**
Given a Eilenberg-Moore algebra $(X, \beta)$ for a monad $\mathcal{T}$ in the category of sets and a set $Y$, one can define a new Eilenberg-Moore algebra $(X^Y, \hat{\beta})$, defined, given $f \in \mathcal{T}(X^Y)$, by

$$\begin{array}{rccc} \hat{\beta}(f): & Y & \to & X \\ & y & \mapsto & \beta(\mathrm{st}(f)(y)) \end{array}$$

With this algebra, we can state the main theorem of this section, found in [5].

**Theorem 3.4 (Bialgebraic semantics)**
Given some monad $(\mathcal{T}, \eta, \mu)$, an Eilenberg-Moore algebra $\beta : \mathcal{T}(\mathbf{2}) \to \mathbf{2}$, and a $\mathcal{T}$-automaton $(X, f)$, there exists a unique map $\mathrm{beh}_1 : X \to \mathbf{2}^{A^*}$ that makes the following diagram commute:

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{beh}_1} & \mathbf{2}^{A^*} \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle \langle O, D\rangle} \\ \mathbf{2} \times (\mathcal{T}(X))^A = \mathcal{D}(\mathcal{T}(X)) & \xrightarrow{\mathcal{D}(\mathcal{T}(\mathrm{beh}_1))} & \mathcal{D}(\mathcal{T}(\mathbf{2}^{A^*})) \end{array} \qquad \mathcal{D}(\mathbf{2}^{A^*}) \uparrow{\scriptstyle \mathcal{D}(\hat{\beta})}$$

(recall that $\mathcal{D}$ is the functor for deterministic automata, that is $\mathcal{D} = \mathbf{2} \times (-)^A$).

This means that given only an Eilenberg-Moore algebra on $\mathbf{2}$, we can get a semantics for any $\mathcal{T}$-automaton, and moreover this semantics is the only one "compatible" with the algebra. When one considers in detail the above diagram for $\mathrm{beh}_1$, this "compatibility" says that one can either make a transition in the automaton, then go to the semantics world and use $\beta$ to aggregate the states together, or go to the semantics world and make the transition there, and that these two yield the same result.

When looking at examples, spelling out this diagram in concrete terms usually corresponds to an inductive definition of acceptance of a word, similar to the one we gave for non-deterministic automata in section 3.2: there is a rule for acceptance of the empty word, and a rule for acceptance of a word $a \cdot w$, based on the transitions made from the current state using $a$. We will have another example of this with alternating automata once we have a suitable $\mathcal{T}$-automaton structure for it.

In this theorem, the algebra $\beta$ should be seen as the formal translation of the accept condition after one step. The following example, that uses the same monad but different algebras, illustrates that.

**Example 3.6 (Universal and existential non-deterministic automata)**
Given a $\mathcal{P}$-automaton, one has two "natural" ways to define $\beta : \mathcal{P}(\mathbf{2}) \to \mathbf{2}$: one can take $\beta(P) = \max(P)$ or $\beta(P) = \min(P)$.

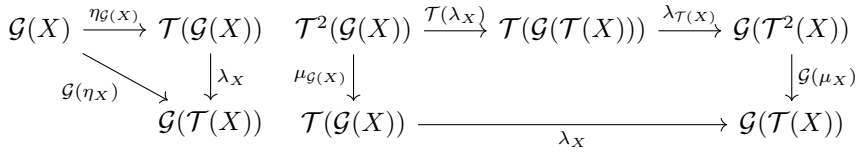
The map $\mathrm{beh}_{\max}$ obtained by the first one is the usual semantics, where we require that **there exists** a transition $q \xrightarrow{a} q'$ where $q'$ accepts $w$ for $q$ to accept $a \cdot w$. This is the usual definition of a non-deterministic automaton (as appearing in section 3.2), but we could also call it an existential non-deterministic automaton.

On the contrary, the map $\mathrm{beh}_{\min}$ obtained by the second one corresponds to the semantics where we require that **all** transitions $q \xrightarrow{a} q'$ lead to a state $q'$ accepting $w$ for $q$ to accept $a \cdot w$. This is also called a universal non-deterministic automaton.

## 3.4 Semantics Via Determinization

The content of this section is drawn out of [6]. Its main ingredient is a so called distributive law.

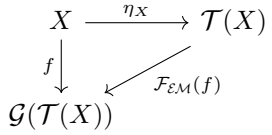
**Definition 3.9 (Distributive law)**
Given a monad $(\mathcal{T}, \eta, \mu)$ and a functor $\mathcal{G}$, a distributive law is a natural transformation $\lambda : \mathcal{TG} \Rightarrow \mathcal{GT}$, that is compatible with the monad structure, that is such that the following diagrams commute for any object $X$:




**Theorem 3.5 (Determinization via a distributive law)**
Given a monad $(\mathcal{T}, \eta, \mu)$, a functor $\mathcal{G}$, a distributive law $\lambda$ and a $\mathcal{GT}$-coalgebra $(X, f)$, one can construct a determinization of $(X, f)$, namely the $\mathcal{G}$-coalgebra $(\mathcal{T}(X), \mathcal{F}_{\mathcal{EM}}(f))$, where

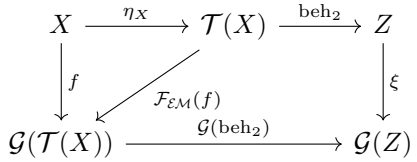
$$\mathcal{F}_{\mathcal{EM}}(f) : \mathcal{T}(X) \to \mathcal{G}(\mathcal{T}(X)) = \left(\mathcal{T}(X) \overset{\mathcal{T}(f)}{\to} \mathcal{T}(\mathcal{G}(\mathcal{T}(X))) \overset{\lambda_{\mathcal{T}(X)}}{\to} \mathcal{G}(\mathcal{T}^2(X)) \overset{\mathcal{G}(\mu_X)}{\to} \mathcal{G}(\mathcal{T}(X))\right)$$

Moreover, this determinized coalgebra makes the following diagram commute:




This commutative triangle is also called the generalized powerset construction, and has appeared for the first time is [14].

**Corollary 3.6**
In the context of the theorem above, if the functor $\mathcal{G}$ has a final coalgebra $(Z, \xi)$, one gets a semantics for $f$ via the unique arrow $\text{beh}_2 : \mathcal{T}(X) \to Z$, that is the unique arrow making the following diagram commute:




In the case where $\mathcal{G}$ is the functor for automata, the situation is even better, as the distributive law can be constructed from a $\mathcal{T}$-algebra, similar to the one used in section 3.3, as shown in [6].

**Proposition 3.7 (Distributive law arising from an algebra)**
Given a monad $(\mathcal{T}, \eta, \mu)$ and a $\mathcal{T}$-algebra $\beta : \mathcal{T}(\mathbf{2}) \to \mathbf{2}$, there is a distributive law $\lambda$ between $\mathcal{T}$ and $\mathcal{D}$, constructed as follows:

$$\lambda_X : \mathcal{T}(\mathbf{2} \times X^A) \to \mathbf{2} \times \mathcal{T}(X)^A = \left(\mathcal{T}(\mathbf{2} \times X^A) \overset{\langle \mathcal{T}(\pi_1), \mathcal{T}(\pi_2) \rangle}{\longrightarrow} \mathcal{T}(\mathbf{2}) \times \mathcal{T}(X^A) \overset{\beta \times st}{\longrightarrow} \mathbf{2} \times \mathcal{T}(X)^A\right)$$

where $\pi_1$ and $\pi_2$ are the two projection from the product $\mathbf{2} \times X^A$.

So as before, given only a $\mathcal{T}$-algebra on $\mathbf{2}$ we are able to fully define a semantics.

**Example 3.7 (Correspondance with concrete determinization)**
In the case where the monad is $\mathcal{P}$, the functor is $\mathcal{D}$ and $\beta$ is max, the obtained determinized coalgebra is exactly the same as in the usual determinization: if $(Q, \langle o, \delta \rangle)$ is a non-deterministic automaton, then $\mathcal{F}_{\mathcal{EM}}(\langle o, \delta \rangle) = \langle o^{\#}, \delta^{\#} \rangle$, with $o^{\#}$ and $\delta^{\#}$ defined as in 3.2.

Then the theorem of preservation of semantics (theorem 3.2) just states that the semantics defined by $\text{beh}_2 \circ \eta_Q$ is the same as the one defined concretely, because for $\mathcal{P}$, the unit $\eta$ is defined by $\eta_Q(q) = \{q\}$.

The most interesting fact, however, is the following, which appears in [5], but relies on more high-level results from [2].

**Theorem 3.8 (Correspondance of the two semantics)**
Given a monad $(\mathcal{T}, \eta, \mu)$ and a $\mathcal{T}$-algebra $\beta : \mathcal{T}(\mathbf{2}) \to \mathbf{2}$, the two semantics $\text{beh}_1$ of section 3.3 and $\text{beh}_2$ are such that $\text{beh}_2 \circ \eta_X = \text{beh}_1$.

In other words, the bialgebraic semantics and the semantics via determinization are essentially the same, even if the way they are presented is quite different. This can be interpreted in two different ways: one could argue the fact that those two constructs yield in the end the same result, is an argument in favor of thinking that they represent the "natural" way to associate a semantics to a $\mathcal{T}$-automaton.

The second way, and maybe more interesting way, is to view the previous theorem as stating that the determinization procedure yields the only semantics that is compatible with $\beta$ (in the sense we developed at the end of section 3.3), so that it is in a way the only correct way to determinize with respect to $\beta$.

The two different constructions also have different interests: $\text{beh}_1$ is useful in getting a semantics, and obtaining in concrete terms the definition of this semantics, while $\text{beh}_2$ is a way to construct a new automaton, that recognizes the same language as the $\mathcal{T}$-automaton we were considering.

# 4 The Case Of Alternating Automata

## 4.1 Alternating Automata

In this section, we present the state-based system we are interested in, the alternating automaton. This model is a kind of extension of the non-deterministic automaton: in a non-deterministic automaton, a word $a \cdot w$ is accepted from a state $q$ if there is a transition $q \xrightarrow{a} q'$ to a state $q'$ that accepts the word $w$. However, one could choose a different rule, for instance say that **every** transition labeled with $a$ should lead to a state accepting $w$. These are the simplest examples, but one could wish to use other logical rule to relate acceptation by a state to acceptation by other states. This is the idea that lead to the model of alternating automata.

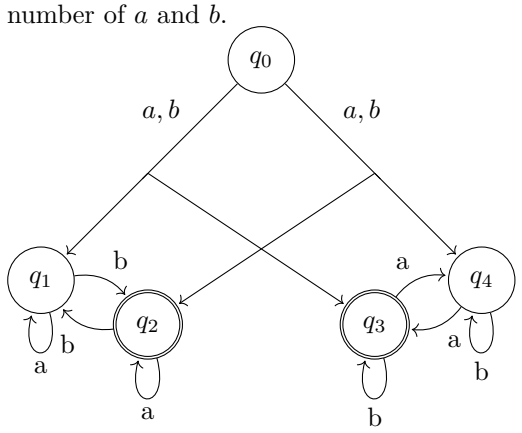
Alternating automata were first introduced in [3] with slightly more general features than the ones we consider, and in a quite different presentation. Our model has been chosen over the original one because of the ease to translate it in a categorical fashion.

**Definition 4.1 (Alternating automata (preliminary))**
An alternating automaton with respect to an alphabet $A$ is a coalgebra for the functor $\mathcal{D} \circ \mathcal{P} \circ \mathcal{P}$, that is a set $Q$ together with a function $\langle o, \delta \rangle : Q \to \mathbf{2} \times \mathcal{P}(\mathcal{P}(Q))^A$. As for a deterministic automaton, the function $o : Q \to \mathbf{2}$ represents the accepting states. Concerning the transition function, $\mathcal{P}(\mathcal{P}(Q))$ is seen as a set of "forks".
Acceptation of a word is then defined by induction, as follows:

- a state $q$ accepts the empty word if and only if $o(q) = \mathbf{1}$,
- a state $q$ accepts the word $a \cdot w$ if and only if there is a fork in $\delta(q)(a)$ such that every state of the fork accepts $w$, that is $accepts(q, a \cdot w) \Leftrightarrow \exists\, F \in \delta(q)(a),\ \forall\, q' \in F,\ accepts(q, w)$.

Note how we decomposed our functor in two parts: one corresponding to the automaton structure, and the other one to the non-deterministic part. This is the same decomposition we already studied in section 3. Our hope is to give a monadic structure to $\mathcal{P} \circ \mathcal{P}$, so that our definition of alternating automaton fits in the generic picture of determinization of a $\mathcal{T}$-automaton in section 3.

**Example 4.1 (Example of alternating automata)**
The following automaton, with start state $q_0$, recognizes words with an even number of $a$ and $b$, or an odd number of $a$ and $b$.




## 4.2 A Monadic Structure For Alternating Automata

This section contains most of the original work of the report. For reasons of clarity, we did not include the proofs of the original theorems we state in the section. However we added them in appendix A, so that the interested reader can still have a look at them.

As we want to give a monadic structure to $\mathcal{PP}$, and we already know $\mathcal{P}$ has a monadic structure, we are looking for a way to compose monads. The following proposition is the usual way it is done. It involves a distributive law between monads, which is defined in a similar way to a distributive law as defined in section 3.4, with two more properties for the compatibility of the law with the second monad.

**Proposition 4.1 (Composition of monads)**
Given two monads $(\mathcal{T}_1, \eta^1, \mu^1)$ and $(\mathcal{T}_2, \eta^2, \mu^2)$ and a distributive law between monads $\lambda : \mathcal{T}_1\mathcal{T}_2 \Rightarrow \mathcal{T}_2\mathcal{T}_1$, one can create a new composite monad, with the components:

- functorial part $\mathcal{T} = \mathcal{T}_2 \circ \mathcal{T}_1$,
- unit $\eta$ defined by the components

$$\eta_X : X \to \mathcal{T}(X) = \left( X \xrightarrow{\eta^1_X} \mathcal{T}_1(X) \xrightarrow{\eta^2_{\mathcal{T}_1(X)}} \mathcal{T}_2(\mathcal{T}_1(X)) \right)$$

- multiplication $\mu$ defined by the components

$$\mu_X : \mathcal{T}^2(X) \to \mathcal{T}(X) = \left( \mathcal{T}_2(\mathcal{T}_1(\mathcal{T}_2(\mathcal{T}_1(X)))) \xrightarrow{\mathcal{T}_2(\lambda_{\mathcal{T}_1(X)})} \mathcal{T}_2^2(\mathcal{T}_1^2(X)) \xrightarrow{\mu^2_{\mathcal{T}_1^2(X)}} \mathcal{T}_2(\mathcal{T}_1^2(X)) \xrightarrow{\mathcal{T}_2(\mu^1_X)} \mathcal{T}_2(\mathcal{T}_1(X)) \right)$$

Note that naturality of the components of both monads ensure that the order in which the units and multiplications are performed are irrelevant, for instance one could apply $\eta^2$ first and the $\eta^1$, the resulting $\eta$ would be the same.

One could then try to construct a suitable distributive law for powerset over powerset, that should somehow reflect the fact that the outside powerset is considered disjunctively and the inside one is considered conjunctively. The natural candidate is the following law:

$$\begin{array}{rccl} \lambda_X : & \mathcal{P}(\mathcal{P}(X)) & \to & \mathcal{P}(\mathcal{P}((X)) \\ & S & \mapsto & \{V \subseteq \cup S \mid \forall\, U \in S,\ \mathrm{Card}(V \cap U) = 1\} \end{array}$$

that corresponds to the conversion of a disjunctive normal form into a conjunctive normal form.

However this is sadly not even a natural transformation. Patching it to

$$\begin{array}{rccl} \lambda_X : & \mathcal{P}(\mathcal{P}(X)) & \to & \mathcal{P}(\mathcal{P}((X)) \\ & S & \mapsto & \{V \subseteq \cup S \mid \forall\, U \in S,\ \mathrm{Card}(V \cap U) \geq 1\} \end{array}$$

(instead of taking exactly one element in every set, we take at least one) yields a natural transformation, however this natural transformation is not a distributive law (of monads over monads), and actually there are examples in [13] showing the natural transformation does not lead to a monad structure on $\mathcal{PP}$.

A way to explain it is to observe that in the functor $\mathcal{PP}$ we wish to turn into a monad, the outside $\mathcal{P}$ is in some way too relaxed. Indeed, if we have two forks $f \subseteq f' \subseteq X$, then the fork $f'$ accepts a word only if the fork $f$ accepts that word as well, so $\{f, f'\}$ accepts the exact same words as $\{f\}$. This is not very annoying for our concrete definition of acceptance, but as we just saw, when we move to category theory, the outside $\mathcal{P}$ causes troubles.

There are at least two different ways to patch this and get a unique set of fork to represent an acceptance condition:

- require that no two forks are comparable; this replaces $\mathcal{P}(\mathcal{P}(X))$ by $\mathcal{A}(\mathcal{P}(X))$, the set of antichains (set of pairwise incomparable elements) of $\mathcal{P}(X)$,
- close the set of forks with respect to inclusion; this replaces $\mathcal{P}(\mathcal{P}(X))$ by $\tilde{\mathcal{U}p}(\mathcal{P}(X))$, the set of upward closed sets of $\mathcal{P}(X)$.

These two solutions are actually really close, as for any ordered set, there is a bijection between antichains over the set and upsets over the set (taking the minimal elements of an upset, and the upwards closure of an antichain). This is why we only explore one of the two idea — the one using upsets. It will enable us to give a correct distributive law, and following, also a correct monadic structure. This is why we alter the coalgebraic definition of alternating automata to the following:

**Definition 4.2 (Alternating automata)**
An alternating automaton with respect to an alphabet $A$ is a coalgebra for the functor $\mathcal{D} \circ \widetilde{\mathcal{U}p} \circ \mathcal{P}$, where $\mathcal{P}$ is the powerset monad (with codomain the category **Poset**), and $\widetilde{\mathcal{U}p}$ is a functor from the category **Poset** to the category **Set** that takes a poset to the set of its upwards closed sets.
We will denote the functor $\widetilde{\mathcal{U}p} \circ \mathcal{P}$ by $\mathcal{A}lt$.

Now, if we want to make this definition work with the results from section 3, we need to give $\mathcal{A}lt$ a monadic structure. This is where category theory proves useful, because it will enable us to combine simple constructions in a reasonably easy way.

**Definition 4.3 (Upwards closure, downwards closure)**
Given a poset $X$ and a subset $P$ of $X$, the upwards closure of $P$, denoted as $\uparrow P$ is the set defined by

$$\uparrow P = \{x \in X \mid \exists\, y \in P,\ y \leq x\}$$

The downwards closure of $P$, denoted as $\downarrow P$, is defined similarly as

$$\downarrow P = \{x \in X \mid \exists\, y \in P,\ y \geq x\}$$

Note that the upwards (resp. downwards) closure of a set is always upwards (resp. downwards) closed.

Using this, we can define two monads on **Poset**.

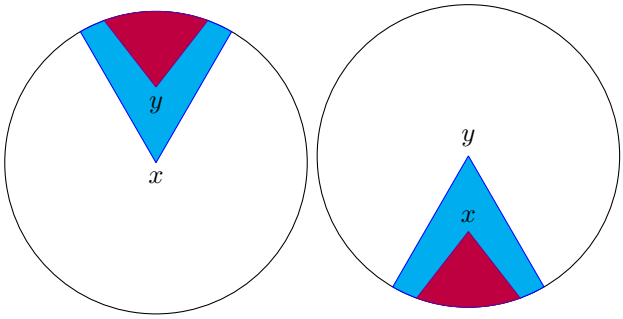
**Definition 4.4 (Upset monad)**
We define the monad $(\mathcal{Up}, \eta^{\mathcal{Up}}, \mu^{\mathcal{Up}})$ on the category of posets as follows:

- on objects, $\mathcal{Up}((X, \preceq))$ is the set of upwards closed sets of $X$, ordered by **reversed** inclusion order, that is $\mathcal{Up}(X) = (\{P \subseteq X \mid \forall\, x, y \in X,\ x \preceq y \wedge x \in P \Rightarrow y \in P\}, \supseteq)$,
- on arrows, $\mathcal{Up}(f)(P) = \uparrow f(P) = \{y \in X \mid \exists\, x \in P,\ f(x) \leq y\}$,
- the unit is $\eta^{\mathcal{Up}}(x) = \uparrow\{x\}$,
- the multiplication is $\mu^{\mathcal{Up}}(S) = \cup S$, as for the powerset monad.

**Definition 4.5 (Downset monad)**
We define the monad $(\mathcal{Dn}, \eta^{\mathcal{Dn}}, \mu^{\mathcal{Dn}})$ on the category of posets as follows:

- on objects, $\mathcal{Dn}(X)$ is the set of downwards closed sets of $X$, ordered by inclusion order, that is $\mathcal{Dn}(X) = (\{P \subseteq X \mid \forall\, x, y \in X,\ x \geq y \wedge x \in P \Rightarrow y \in P\}, \subseteq)$,
- on arrows, $\mathcal{Dn}(f)(P) = \downarrow f(P)$,
- the unit is $\eta^{\mathcal{Dn}}(x) = \downarrow\{x\}$,
- the multiplication is $\mu^{\mathcal{Dn}}(S) = \cup S$.

The reason of the reversion of the order for the inclusion can be understood with the two following diagrams:




On the leftmost, $x \leq y$ but $\uparrow x \supseteq \uparrow y$, whereas on the rightmost $x \leq y$ and $\uparrow x \subseteq \uparrow y$.

Note that the $\mathcal{Up}$ functor is of type $\mathcal{Up}$ : **Poset** $\rightarrow$ **Poset**, so the functor $\tilde{\mathcal{Up}}$ that we use in the definition of alternating automata is really just $\mathcal{U} \circ \mathcal{Up}$.

We also have a relation between $\mathcal{Dn}$, $\mathcal{O}$ and $\mathcal{P}$:

**Proposition 4.2 (Downwards closed sets of a discrete order)**
We have the equality

$$\mathcal{Dn} \circ \mathcal{O} = \mathcal{P}$$

**Proof**
Take a set $X$. By definition of $\mathcal{Dn}$, it holds that $\mathcal{Dn} \circ \mathcal{O}(X)$ is a subset of $\mathcal{P}(X)$, ordered by inclusion. So we only need to show that any subset of $X$ is a downwards closed set of $\mathcal{O}(X)$. Now, take $P$ a subset of $X$, $x$ and $y$ elements of $X$ and suppose $x \in P$. Because of the definition of the discrete order, if $x \geq y$ then $x = y$, and so $y \in P$. So $P$ is indeed downwards closed on $\mathcal{O}(X)$.

Thus, we can now rewrite the functor $\mathcal{Alt}$ as $\mathcal{U} \circ \mathcal{Up} \circ \mathcal{Dn} \circ \mathcal{O}$. This does not seem much of an improvement, but we can now use the adjunction $\mathcal{O} \dashv \mathcal{U}$ mentioned in section 2.2 and the monadic composition of $\mathcal{Up}$ and $\mathcal{Dn}$ to turn $\mathcal{Alt}$ into a monad. But first, we need a last ingredient. This is the distributive law we have been

advertising throughout the paper. It has not been created for this paper, although tracing it back is hard to do. In our case, we found its description in [12, p. 220-221].

**Theorem 4.3 (Distributive law)**
The following $\lambda : \mathcal{D}n\,\mathcal{U}p \Rightarrow \mathcal{U}p\,\mathcal{D}n$ is a distributive law between monads:

$$\begin{array}{rccl} \lambda_X : & \mathcal{D}n(\mathcal{U}p(X)) & \to & \mathcal{U}p(\mathcal{D}n(X)) \\ & S & \mapsto & \{T \in \mathcal{D}n(X) \mid \forall\, s \in S,\ s \cap T \neq \emptyset\} \end{array}$$

This transformation is similar to the way one transforms a disjunctive normal form into a conjunctive normal form in logic: to form a disjunction of conjunction from a conjunction of disjunction, one makes a big disjunction of all the different ways to pick one literal in each disjunction of the disjunctive normal form. Here, because of the upwards and downwards closure, at least one literal is taken rather than exactly one, but the idea is the same. Also, the functors $\mathcal{U}p$ and $\mathcal{D}n$ encode this logical view on the powerset directly in the type, by making a clear difference between the monad interpreted disjunctively and the one interpreted conjunctively.

Using the first proposition of this section about the composition of monads, this distributive law yields a monad structure.

**Corollary 4.4 (Monad structure for $\mathcal{U}p\,\mathcal{D}n$)**
The functor $\mathcal{U}p\,\mathcal{D}n$ can be given a monad structure $(\mathcal{U}p\,\mathcal{D}n, \eta^1, \mu^1)$.

This distributive law is the most important element of this report: giving a categorical semantics to alternating automata amounts to give a monad structure to some functor (either $\mathcal{PP}$ or a modified version of it), which in turn amounts to finding a correct distributive law. Quite a lot of errors have been made trying to define this distributive law: a list appears in [7], itself being a correction of an error in [8], tracing back the error to reference books such as [11]. A lot of patches have been found, but ours is the first completely satisfactory one: it preserves the properties of powerset (idempotency, commutativity, associativity), which was not the case in certain attempts (for instance, the lists/languages used in [5]), while keeping the full power of alternating automata (contrary to the simpler version covered in [6]), and it is an actual distributive law between monads (unlike the one proposed in [9], a corrected version of [8]).

The main idea that had not been used before is to define the distributive law on **Poset** instead of **Set**, and to use the adjunction $\mathcal{O} \dashv \mathcal{U}$ to turn the obtained monad into a monad on **Set**.

Using another category and adjunctions to construct a correct monad structure for alternating automata is not a new idea: an idea of how to do this (using semi-lattices and distributive lattices) is given in [13]. We tried this before devising our current solution, but the attempt was not successful.

The exact definition of the complete monad is then as follows:

**Theorem 4.5 (Monad structure for $\mathcal{A}lt$)**
Let $\eta^2$ (resp. $\varepsilon^2$) be the unit (resp. counit) of the adjunction $\mathcal{O} \dashv \mathcal{U}$. Then the functor $\mathcal{A}lt$ (equal to $\widetilde{\mathcal{U}p}\,\mathcal{P}$ or $\mathcal{U}\,\mathcal{U}p\,\mathcal{D}n\,\mathcal{O}$) is a monad, which unit $\eta$ has components

$$\eta_X : \mathrm{id}_X \xrightarrow{\eta^2_X} \mathcal{U}(\mathcal{O}(X)) \xrightarrow{\mathcal{U}(\eta^1_{\mathcal{O}(X)})} \mathcal{A}lt(X)$$

and multiplication $\mu$ has components

$$\mu_X : \mathcal{A}lt(\mathcal{A}lt(X)) \xrightarrow{\mathcal{U}\,\mathcal{U}p\,\mathcal{D}n\,\varepsilon^2_{\mathcal{U}p\,\mathcal{D}n\,\mathcal{O}(X)}} \mathcal{U}\,\mathcal{U}p\,\mathcal{D}n\,\mathcal{U}p\,\mathcal{D}n\,\mathcal{O}(X) \xrightarrow{\mathcal{U}(\mu^1_{\mathcal{O}(X)})} \mathcal{A}lt(X)$$

This construction is not ad-hoc, but it comes from the link between monads and adjunctions. See appendix A.4 for details.

One can easily compute the unit of this monad, and get

$$\eta_X(x) = \uparrow\{x\} = \{T \in \mathcal{P}X \mid x \in T\}$$

For the multiplication, the computation gives the following:

$$\mu_X(S) = \{T \in \mathcal{P}(X) \mid \exists\, s \in S,\ \forall\, t \in s,\ \exists\, u \in t,\ \forall\, v \in u,\ v \in T\}$$

Because we constructed this multiplication stepwise, we can also give an intuition of how it works:

1. use $\varepsilon^2$ to get rid of the $\mathcal{O}\mathcal{U}$ in the middle without really modifying the object (recall from section 2.2 that $\varepsilon^2$ is merely just the identity),
2. use the distributive law to exchange the position of $\mathcal{U}p$ and $\mathcal{D}n$, similarly as the transformation of a disjunctive normal form into a conjunctive normal form,
3. flatten two levels of $\mathcal{U}p$ into one and two levels of $\mathcal{D}n$ into one using the union.

### 4.3 Induced Semantics

Now that we have a monad, we need an algebra for this monad to be able to define a semantics as in section 3.

**Proposition 4.6 (Algebra for $\mathcal{A}lt$)**
The pair $(\mathbf{2}, \beta)$ where

$$\begin{array}{rccl} \beta: & \mathcal{A}lt(\mathbf{2}) & \to & \mathbf{2} \\ & S & \mapsto & \begin{cases} \mathbf{1} \text{ if } \{\mathbf{1}\} \in S \\ \mathbf{0} \text{ otherwise} \end{cases} \end{array}$$

is an algebra for $\mathcal{A}lt$.

On a side note, one has $\mathcal{A}lt(\emptyset) = \widetilde{\mathcal{U}p}(\mathcal{P}(\emptyset)) = \widetilde{\mathcal{U}p}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\} = \mathbf{2}$, and the algebra $(\mathbf{2}, \beta)$ we just gave is actually $(\mathcal{A}lt(\emptyset), \mu_\emptyset)$, which is usually called the free algebra on $\emptyset$.

Given this $\beta$ and a alternating automaton $(X, \langle o, \delta\rangle)$, there is a unique map $beh_1 : X \to \mathbf{2}^{A^*}$ that makes the following diagram commute:

$$\begin{array}{ccc} X & \xrightarrow{\quad\mathrm{beh}_1\quad} & \mathbf{2}^{A^*} \\ {\scriptstyle\langle o,\delta\rangle}\downarrow & & \downarrow{\scriptstyle\langle O,D\rangle} \\ & & \mathcal{D}(\mathbf{2}^{A^*}) \\ & & \uparrow{\scriptstyle\mathcal{D}(\hat{\beta})} \\ \mathbf{2}\times(\mathcal{A}lt(X))^A = \mathcal{D}(\mathcal{A}lt(X)) & \xrightarrow{\mathcal{D}(\mathcal{A}lt(\mathrm{beh}_1))} & \mathcal{D}(\mathcal{A}lt(\mathbf{2}^{A^*})) \end{array}$$

If we spell out $\hat{\beta}$, we get $\hat{\beta}(S)(w) = \mathbf{1} \Leftrightarrow \exists\, s \in S,\ \forall\, L \in S,\ L(w) = \mathbf{1}$, and we can translate the above diagram into two conditions, corresponding to the first and second components of the products:

- for $q \in X$, $\mathrm{beh}_1(q)(\varepsilon) = o(q)$, so $\mathrm{beh}_1(q)$ accepts the empty word if and only if $o(q)$ is $\mathbf{1}$,
- for $q \in X$, $a \in A$ and $w \in A^*$, $\mathrm{beh}_1(q)(a \cdot w) = \mathbf{1} \Leftrightarrow \exists\, F \in \delta(q)(a),\ \forall\, q' \in F,\ \mathrm{beh}_1(q')(w) = \mathbf{1}$.

This corresponds exactly to the concrete definition of acceptance that we gave for an alternating automaton in section 4.1! So our whole construction is sound, as the categorical approach to alternating automata is equivalent to the usual one.

## 5 Conclusion

Using the construction described in this report, we are finally able to fully fit alternating automata in the large picture of determinization, and show that the problem there was not a failure of the theory, but rather a lack of a proper monad. A problem that we solved by finding a happy detour through order structures.

A nice thing about this construction is that it really shows the power of category theory: without it, formulating just the right monad would have been really hard. But using categorical tools like distributive law and adjunctions, breaking the problem in small, handleable pieces, and then putting these pieces together, makes the problem reasonable.

# References


[1] S. Awodey. **Category Theory**. Oxford: Clarendon Press, 2006.

[2] F. Bartels. "On Generalised Coinduction and Probabilistic Specification Formats: Distributive Laws in Coalgebraic Modelling". PhD thesis. F.U. Amsterdam, 2004.

[3] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. "Alternation". In: **J. ACM** 28.1 (Jan. 1981), pp. 114–133. DOI: 10.1145/322234.322243.

[4] B. Jacob. **Introduction to Coalgebra. Towards Mathematics of States and Observation**. Cambridge University Press, 2016.

[5] B. Jacobs. "A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages". In: **Algebra, Meaning, and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday**. Ed. by K. Futatsugi, J.-P. Jouannaud, and J. Meseguer. Springer Berlin Heidelberg, 2006, pp. 375–404. DOI: 10.1007/11780274_20.

[6] B. Jacobs, A. Silva, and A. Sokolova. "Trace semantics via determinization". In: **Journal of Computer and System Sciences** 81.5 (2015), pp. 859–879. DOI: 10.1016/j.jcss.2014.12.005.

[7] B. Klin. "An Erroneous Monad Structure On Double Covariant Powerset". 2016.

[8] B. Klin and J. Rot. "Coalgebraic Trace Semantics via Forgetful Logics". In: **FOSSACS 2015, Proceedings**. Ed. by A. Pitts. Springer Berlin Heidelberg, 2015, pp. 151–166. DOI: 10.1007/978-3-662-46678-0_10.

[9] B. Klin and J. Rot. "Coalgebraic trace semantics via forgetful logics". In: **Logical Methods in Computer Science** 12.4 (2016). DOI: 10.2168/LMCS-12(4:10)2016.

[10] S. MacLane. **Categories For The Working Mathematician**. 2nd ed. Springer New York, 1978.

[11] E. Manes. "Monads of sets". In: **Handbook of Algebra** 3 (2003), pp. 67–153. DOI: 10.1016/S1570-7954(03)80059-1.

[12] F. Marmolejo, R. Rosebrugh, and R. Wood. "A basic distributive law". In: **Journal of Pure and Applied Algebra** 168.2 (2002), pp. 209–226. DOI: 10.1016/S0022-4049(01)00097-4.

[13] J. Moerman. "$\mathcal{PP}$ is not a monad?!" 2017. URL: http://joshuamoerman.nl/notes/17pp-is-not-a-monad.pdf.

[14] A. Silva et al. "Generalizing determinization from automata to coalgebras". In: **Logical Methods in Computer Science** 9.1 (2013). DOI: 10.2168/LMCS-9(1:9)2013.

# Appendices

## A Proofs

Because of the reversed inclusion order used in the monad $\mathcal{U}p(P)$, given a set of states $X$ ordered with some order $\leq$ and a subset $P$ of $X$, we need to make a difference between the upwards (resp. downwards) closure of $P$ with respect to $\leq$, and the upwards (resp. downwards) closure of $P$ with respect to the usual inclusion order. We will use the arrows $\uparrow$ (resp. $\downarrow$) for the first one, and $\restriction$ (resp. $\downarrow$) for the second.

Also, we will write $\{x \in S \mid P(x)\}$ for the set of elements of $S$ having the property $P$ and $\{f(s), s \in S\}$ for the set $\{x \in T \mid \exists\, s \in S,\ t = f(s)\}$ if $T$ is the codomain of $f$.

Finally, to avoid confusion due to the many levels of intricate sets, we will write $f_*$ for the direct image, that is $f_*(S) = \{f(s), s \in S\}$.

### A.1 Preliminary Order Lemmas

**Lemma A.1 (Downwards closure and union)**
Given a set of sets $S$, we have $\cup \downarrow S = \cup S$.

**Lemma A.2 (Order closure and direct image)**
Given two posets $(X, \leq)$ and $(Y, \preceq)$, a monotone map $f : X \to Y$ and a set $P \subseteq X$, we have $\uparrow f_*(\uparrow P) = \uparrow f_*(P)$, and similarly $\downarrow f_*(\downarrow P) = \downarrow f_*(P)$.

**Lemma A.3 (Upwards closure and intersection)**
If $S$ is a set of sets and $t$ is a set, then $\forall\, s \in \restriction S,\ t \cap s \neq \emptyset$ is equivalent to $\forall\, s \in S,\ t \cap s \neq \emptyset$.

### A.2 Monad Structure of $\mathcal{U}p$ and $\mathcal{D}n$

**Proposition A.4 (Upset monad)**
The triple $(\mathcal{U}p, \eta^{\mathcal{U}p}, \mu^{\mathcal{U}p})$, defined as follows:

- on objects, $\mathcal{U}p((X \leq))$ is the set of upwards closed sets of $X$, ordered by **reversed** inclusion order, that is $\mathcal{U}p((X, \leq)) = (\{P \subseteq X \mid \forall\, x, y \in X,\ x \leq y \wedge x \in P \Rightarrow y \in P\}, \supseteq)$,
- on arrows, $\mathcal{U}p(f)(P) = \uparrow f_*(P)$,
- the unit is $\eta^{\mathcal{U}p}(x) = \uparrow \{x\}$,
- the multiplication is $\mu^{\mathcal{U}p}(S) = \cup S$,

is a monad.

**Proof**
In this proof we write $\eta$ for $\eta^{\mathcal{U}p}$ and $\mu$ for $\mu^{\mathcal{U}p}$.

First, we need to prove that $\mathcal{U}p$ is a functor. Since direct image and upwards closure preserve inclusion, if $f : X \to Y$ is an arrow in **Poset**, then $\mathcal{U}p(f)$ is monotone, and so it is an arrow $\mathcal{U}p(X) \to \mathcal{U}p(Y)$. Checking that $\mathcal{U}p$ preserves identity is easy. Finally, if $f$ and $g$ are two arrows with the correct types, $\uparrow g_*(\uparrow f_*(S)) = \uparrow g_*(f_*(S)) = \uparrow (g \circ f)_*(S)$, using lemma A.2.

Next, the unit. The upwards closure of the image by $\eta$ is obvious. Next, suppose $(X, \leq)$ is a poset, and $x, y \in X$ are such that $x \leq y$. Then $y \in \uparrow \{x\}$, so $\uparrow \{y\} \subseteq \uparrow \{x\}$, and so $\eta_{(X,\leq)}(x) \supseteq \eta_{(X,\leq)}(y)$. But because $\mathcal{U}p(X)$ is ordered

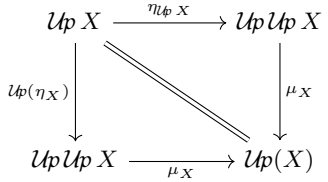
with respect to the **reversed** inclusion order, $\eta_{(X,\leq)}$ is monotone, and it is an arrow $\eta_P : P \to \mathcal{U}p(P)$. To prove the naturality, we need to show $\uparrow f_*(\uparrow\{x\}) = \uparrow\{f(x)\}$. But using lemma A.2, we have $\uparrow f_*(\uparrow\{x\}) = \uparrow f_*(\{x\})$ and since $f_*(\{x\}) = \{f(x)\}$, we have naturality of $\eta$.

Now, the multiplication. Any union of upwards closed sets is upwards closed, and if $S \subseteq S'$ then $\cup S \subseteq (\cup S) \cup (\cup(S'\backslash S)) = \cup S'$ so $\mu_P$ is an arrow $\mu_P : \mathcal{U}p(\mathcal{U}p(P)) \to \mathcal{U}p(P)$. For naturality, we need to prove $\cup(\uparrow\{\uparrow f_*(s), s \in S\}) = \uparrow f_*(\cup S)$. Now because of the reversed order on $\mathcal{U}p(P)$, we have $\cup(\uparrow\{\uparrow f_*(s), s \in S\}) = \cup(\downarrow\{\uparrow f_*(s), s \in S\})$, and using lemma A.1, we deduce

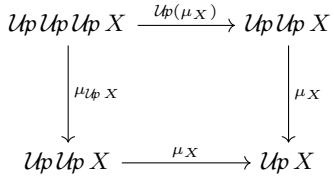
$$\begin{aligned}\cup(\uparrow\{\uparrow f_*(s), s \in S\}) &= \cup\{\uparrow f_*(s), s \in S\}\\ &= \cup\{\uparrow\{f(x), x \in s\}, s \in S\}\\ &= \{\uparrow f(x), x \in \cup S\} \text{ because } x \in s \in S \Leftrightarrow x \in \cup S\end{aligned}$$

and so $\mu$ is natural.

Next, the multiplication and unit are compatible, because $\mu_X \circ \eta_{\mathcal{U}p(X)}(S) = \cup\uparrow\{S\} = \cup\downarrow\{S\} = S$ and $\mu_X \circ \mathcal{U}p(\eta_X)(S) = \cup_{s\in S}(\uparrow\{s\})$, so $x \in \mu \circ \mathcal{U}p(\eta_X)(S)$ if and only if $\exists\, s \in S,\ x \geq s$, that is if and only if $x \in \uparrow S$. But since $S$ is an upset, $\uparrow S = S$. So we indeed have




Finally the diagram




is easy to deduce, as

$$\begin{aligned}\cup\uparrow\{\cup s, s \in S\} &= \cup\downarrow\{\cup s, s \in S\}\\ &= \cup\{\cup s, s \in S\}\\ &= \{x \in X \mid \exists\, s \in S,\ \exists\, t \in s,\ x \in t\}\\ &= \cup\cup S\end{aligned}$$

**Proposition A.5 (Downset monad)**
The triple $(\mathcal{D}n, \eta^{\mathcal{D}n}, \mu^{\mathcal{D}n})$, defined as follows:

- on objects, $\mathcal{D}n(X)$ is the set of downwards closed sets of $X$, ordered by inclusion order, that is $\mathcal{D}n(X) = (\{P \subseteq X \mid \forall\, x, y \in X,\ x \geq y \wedge x \in P \Rightarrow y \in P\}, \subseteq)$,
- on arrows, $\mathcal{D}n(f)(P) = \downarrow f_*(P)$,
- the unit is $\eta^{\mathcal{D}n}(x) = \downarrow\{x\}$,
- the multiplication is $\mu^{\mathcal{D}n}(S) = \cup S$,

is a monad.

**Proof**
The proof is extremely similar to the one for $\mathcal{U}p$, so we will not repeat it. The only thing to note is that every time we transformed a $\uparrow$ into a $\downarrow$ in the previous proof, in the case of $\mathcal{D}n$ we would get a $\downarrow$ on a set ordered by regular inclusion, so it would also translate into $\downarrow$.

## A.3 Naturality Of The Distributive Law

**Theorem A.6 (Distributive law)**
The following $\lambda : \mathcal{D}n\,\mathcal{U}p \Rightarrow \mathcal{U}p\,\mathcal{D}n$ is a distributive law between monads:

$$\begin{array}{rccl} \lambda_X : & \mathcal{D}n(\mathcal{U}p(X)) & \to & \mathcal{U}p(\mathcal{D}n(X)) \\ & S & \mapsto & \{T \in \mathcal{D}n(X) \mid \forall\, s \in S,\ s \cap T \neq \emptyset\} \end{array}$$

**Proof**
First, $\lambda$ is well defined because $\{T \in \mathcal{D}n(X) \mid \forall\, s \in S,\ s \cap T \neq \emptyset\}$ is an upset of downsets.

Next, we prove naturality. Take $X, Y$ posets and $f : X \to Y$ and arrow in **Poset**. We need to prove the following diagram:

$$\begin{array}{ccc} \mathcal{D}n\,\mathcal{U}p\,X & \xrightarrow{\lambda_X} & \mathcal{U}p\,\mathcal{D}n\,X \\ \downarrow{\scriptstyle \mathcal{D}n\,\mathcal{U}p\,f} & & \downarrow{\scriptstyle \mathcal{U}p\,\mathcal{D}n\,f} \\ \mathcal{D}n\,\mathcal{U}p\,Y & \xrightarrow{\lambda_Y} & \mathcal{U}p\,\mathcal{D}n\,Y \end{array}$$

Take $S \in \mathcal{D}n\,\mathcal{U}p\,X$ and $d \in \mathcal{D}n\,Y$. First, we rewrite the lower triangle:

$$\begin{aligned} d \in \lambda_Y \circ \mathcal{D}n\,\mathcal{U}p(f)(S) &\Leftrightarrow \forall\, u \in \mathcal{D}n\,\mathcal{U}p\,f(S),\ u \cap d = \emptyset \\ &\Leftrightarrow \forall\, u \in \downarrow\{\uparrow f_*(s), s \in S\},\ u \cap d \neq \emptyset \\ &\Leftrightarrow \forall\, u \in \restriction\{\uparrow f_*(s), s \in S\},\ u \cap d \neq \emptyset \\ &\Leftrightarrow \forall\, u \in \{\uparrow f_*(s), s \in S\},\ u \cap d \neq \emptyset \text{ by lemma A.3} \\ &\Leftrightarrow \forall\, s \in S,\ d \cap \uparrow f_*(s) \neq \emptyset \\ &\Leftrightarrow \forall\, s \in S,\ d \cap f_*(s) \neq \emptyset \text{ because } d \text{ is a downwards closed set} \\ &\Leftrightarrow \forall\, s \in S,\ \exists\, x \in s,\ f(x) \in d \end{aligned}$$

Now the upper triangle, we obtain:

$$\begin{aligned} d \in \mathcal{U}p\,\mathcal{D}n(f) \circ \lambda_X(S) &\Leftrightarrow d \in \uparrow\{\downarrow f_*(t), t \in \lambda_X(S)\} \\ &\Leftrightarrow \exists\, d' \in \mathcal{D}n\,Y,\ d \geq d' \wedge \exists\, t \in \lambda_X(S),\ d' = \downarrow f_*(t) \\ &\Leftrightarrow \exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (d \geq \downarrow f_*(t)) \\ &\Leftrightarrow \exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (d \supseteq \downarrow f_*(t)) \\ &\Leftrightarrow \exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (f_*(t) \subseteq d) \text{ because } d \text{ is a downset} \\ &\Leftrightarrow \exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (\forall\, x \in t,\ f(x) \in d) \end{aligned}$$

Now suppose $\exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (\forall\, x \in t,\ f(x) \in d)$. Then given $s \in S$, there is some $x_s \in t \cap s$, but then $f(x_s) \in d$, and so $\forall\, s \in S,\ \exists\, x \in s,\ f(x) \in d$. Conversely, suppose $\forall\, s \in S,\ \exists\, x \in s,\ f(x) \in d$. For each $s \in S$ fix some $x_s \in s$ such that $f(x_s) \in d$, and define $t = \downarrow\{x_s, s \in S\}$. Then by construction $t \in \mathcal{D}n\,X$, and moreover $\forall\, s \in S,\ t \cap s \supseteq \{x_s\} \supsetneq \emptyset$. Also, if $x \in t$, then take some $s \in S$ such that $x \leq x_s$. Then $f(x) \leq f(x_s) \in d$, and because $d$ is a downwards closed set, $f(x) \in d$. Thus, $\exists\, t \in \mathcal{D}n\,X,\ (\forall\, s \in S,\ t \cap s \neq \emptyset) \wedge (\forall\, x \in t,\ f(x) \in d)$.
In the end $d \in \lambda_Y \circ \mathcal{D}n\,\mathcal{U}p(f)(S) \Leftrightarrow d \in \mathcal{U}p\,\mathcal{D}n(f) \circ \lambda_X(S)$ and so the diagram commutes.

Next, the first triangle diagram, namely

$$\begin{array}{ccc} \mathcal{U}p\,X & \xrightarrow{\eta^{\mathcal{D}n}_{\mathcal{U}p\,X}} & \mathcal{D}n\,\mathcal{U}p\,X \\ & {\scriptstyle \mathcal{U}p(\eta^{\mathcal{D}n}_X)} \searrow & \downarrow{\scriptstyle \lambda_X} \\ & & \mathcal{U}p\,\mathcal{D}n\,X \end{array}$$

Take $S \in \mathcal{U}p\,X$, we have one one hand

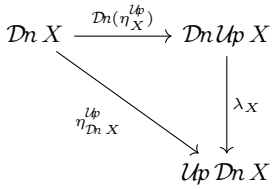
$$\begin{aligned} \mathcal{U}p(\eta^{\mathcal{D}n}_X)(S) &= \uparrow\{\downarrow s, s \in S\} \\ &= \{T \in \mathcal{D}n\,X \mid \exists\, s \in S,\ \downarrow s \subseteq T\} \\ &= \{T \in \mathcal{D}n\,X \mid \exists\, s \in S,\ s \in T\} \text{ because } T \text{ is a downwards closed set} \\ &= \{T \in \mathcal{D}n\,X \mid T \cap S \neq \emptyset\} \end{aligned}$$

On the other hand,

$$\begin{aligned}
\lambda_X \circ \eta^{\mathcal{D}n}_{\mathcal{U}p\,X}(S) &= \left\{T \in \mathcal{D}n\,X \mid \forall\, s \in \eta^{\mathcal{D}n}_{\mathcal{U}p\,X}(S),\ s \cap T \neq \emptyset\right\} \\
&= \{T \in \mathcal{D}n\,X \mid \forall\, s \in \downarrow\{S\},\ s \cap T \neq \emptyset\} \\
&= \{T \in \mathcal{D}n\,X \mid \forall\, s \in \uparrow\{S\},\ s \cap T \neq \emptyset\} \\
&= \{T \in \mathcal{D}n\,X \mid \forall\, s \in \{S\},\ s \cap T \neq \emptyset\} \text{ by lemma A.3} \\
&= \{T \in \mathcal{D}n\,X \mid S \cap T \neq \emptyset\}
\end{aligned}$$

and so the diagram commutes.

The second triangle diagram is:




Take $S \in \mathcal{D}n\,X$, we have on one hand

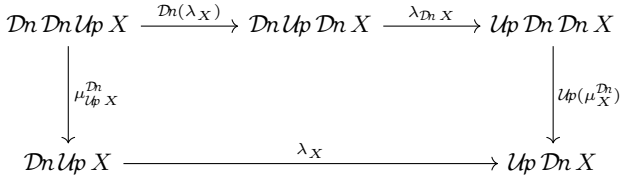
$$\eta^{\mathcal{U}p}_{\mathcal{D}n\,X}(S) = \uparrow\{S\}$$

On the other hand, we have

$$\begin{aligned}
\lambda_X \circ \mathcal{D}n(\eta^{\mathcal{U}p}_X)(S) &= \{T \in \mathcal{D}n(X) \mid \forall\, t \in \downarrow\{\uparrow\{s\}, s \in S\},\ T \cap t \neq \emptyset\} \\
&= \{T \in \mathcal{D}n(X) \mid \forall\, t \in \uparrow\{\uparrow\{s\}, s \in S\},\ T \cap t \neq \emptyset\} \\
&= \{T \in \mathcal{D}n(X) \mid \forall\, t \in \{\uparrow\{s\}, s \in S\},\ T \cap t \neq \emptyset\} \text{ by lemma A.3} \\
&= \{T \in \mathcal{D}n(X) \mid \forall\, s \in S,\ T \cap \uparrow\{s\} \neq \emptyset\} \\
&= \{T \in \mathcal{D}n(X) \mid \forall\, s \in S,\ s \in T\} \text{ because } T \text{ is a downset} \\
&= \{T \in \mathcal{D}n(X) \mid S \subseteq T\} \\
&= \uparrow\{S\}
\end{aligned}$$

And so this triangle diagram commutes as well.

Now, the first rectangle diagram, that is




Take $S \in \mathcal{D}n\,\mathcal{D}n\,\mathcal{U}p\,X$, on one hand we have

$$\begin{aligned}
\lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S) &= \{T \in \mathcal{D}n\,X \mid \forall\, s \in \cup S,\ , s \cap T \neq \emptyset\} \\
&= \{T \in \mathcal{D}n\,X \mid \forall\, s \in S,\ \forall\, t \in s,\ t \cap T \neq \emptyset\}
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
\mathcal{U}p(\mu^{\mathcal{D}n}_X) \circ \lambda_{\mathcal{D}n\,X} \circ \mathcal{D}n(\lambda_X)(S) &= \uparrow\{\cup T, T \in \lambda_{\mathcal{D}n\,X}(\mathcal{D}n(\lambda_X)(S))\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in \mathcal{D}n(\lambda_X)(S),\ s \cap T \neq \emptyset\}\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in \downarrow(\lambda_X)_*(S),\ s \cap T \neq \emptyset\}\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in \uparrow(\lambda_X)_*(S),\ s \cap T \neq \emptyset\}\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in (\lambda_X)_*(S),\ s \cap T \neq \emptyset\}\} \text{ by lemma A.3} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in S,\ \lambda_X(s) \cap T \neq \emptyset\}\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in S,\ T \cap \{u \in \mathcal{D}n\,X \mid \forall\, t \in s,\ t \cap u \neq \emptyset\} \neq \emptyset\}\} \\
&= \uparrow\{\cup T, T \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in S,\ \exists\, u \in T,\ \forall\, t \in s,\ t \cap u \neq \emptyset\}\}
\end{aligned}$$

Take $T \in \mathcal{D}n\,X$ and suppose $T \in \lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S)$, that is

$$\forall\, s \in S,\ \forall\, t \in s,\ t \cap T \neq \emptyset$$

Then we have

$$\forall\, s \in S,\ \exists\, u \in (\downarrow\{T\}),\ \forall\, t \in s,\ t \cap u \neq \emptyset$$

with $u$ taken each time to be $T$. But $\downarrow\{T\}$ is a downset, thus, $\downarrow\{T\} \in \{T \in \mathcal{D}n\,\mathcal{D}n\,X \mid \forall\, s \in S,\ \exists\, u \in T,\ \forall\, t \in s,\ t \cap u \neq \emptyset\}$. Since $T = \cup \downarrow\{T\}$, we deduce that $T \in \mathcal{U}p(\mu^{\mathcal{D}n}_X) \circ \lambda_{\mathcal{D}n\,X} \circ \mathcal{D}n(\lambda_X)(S)$.

Conversely, take $T \in \mathcal{D}n\,\mathcal{D}n\,X$ such that $\forall\, s \in S,\ \exists\, u \in T,\ \forall\, t \in s,\ t \cap u \neq \emptyset$, and take $s \in S$. By hypothesis, take $u_s \in T$ such that $\forall\, t \in s,\ t \cap u_s \neq \emptyset$. Take some $t \in s$, we have $t \cap \cup T \supseteq t \cap u_s \supsetneq \emptyset$, and so $t \cap \cup T \neq \emptyset$. This proves $\cup T \in \lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S)$. Because the set $\lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S)$ is an upwards closed set, for any $T' \geq T$ we still have $T' \in \lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S)$. Thus, we have $\mathcal{U}p(\mu^{\mathcal{D}n}_X) \circ \lambda_{\mathcal{D}n\,X} \circ \mathcal{D}n(\lambda_X)(S) \subseteq \lambda_X \circ \mu^{\mathcal{D}n}_{\mathcal{U}p\,X}(S)$.

The double inclusion proves the equality, and so the diagram commutes.

Finally, the last diagram to prove is

$$\begin{array}{ccccc}
\mathcal{D}n\,\mathcal{U}p\,\mathcal{U}p\,X & \xrightarrow{\lambda_{\mathcal{U}p\,X}} & \mathcal{U}p\,\mathcal{D}n\,\mathcal{U}p\,X & \xrightarrow{\mathcal{U}p(\lambda_X)} & \mathcal{U}p\,\mathcal{U}p\,\mathcal{D}n\,X \\
\downarrow{\scriptstyle \mathcal{D}n(\mu^{\mathcal{U}p}_X)} & & & & \downarrow{\scriptstyle \mu^{\mathcal{U}p}_{\mathcal{D}n\,X}} \\
\mathcal{D}n\,\mathcal{U}p\,X & & \xrightarrow{\qquad\qquad\lambda_X\qquad\qquad} & & \mathcal{U}p\,\mathcal{D}n\,X
\end{array}$$

Take $S \in \mathcal{D}n\,\mathcal{U}p\,\mathcal{U}p\,X$, the lower part gives

$$\begin{aligned}
\lambda_X \circ \mathcal{D}n(\mu^{\mathcal{U}p}_X)(S) &= \{t \in \mathcal{D}n\,X \mid \forall\, u \in \mathcal{D}n(\mu^{\mathcal{U}p}_X)(S),\ u \cap t \neq \emptyset\} \\
&= \{t \in \mathcal{D}n\,X \mid \forall\, u \in \downarrow\{\cup s, s \in S\},\ u \cap t \neq \emptyset\} \\
&= \{t \in \mathcal{D}n\,X \mid \forall\, u \in \uparrow\{\cup s, s \in S\},\ u \cap t \neq \emptyset\} \\
&= \{t \in \mathcal{D}n\,X \mid \forall\, u \in \{\cup s, s \in S\},\ u \cap t \neq \emptyset\} \text{ by lemma A.3} \\
&= \{t \in \mathcal{D}n\,X \mid \forall\, s \in S,\ t \cap \cup s \neq \emptyset\}
\end{aligned}$$

The upper part gives

$$\begin{aligned}
\mu^{\mathcal{U}p}_{\mathcal{D}n\,X} \circ \mathcal{U}p(\lambda_X) \circ \lambda_{\mathcal{U}p\,X}(S) &= \cup\uparrow\{\lambda_X(T), T \in \lambda_{\mathcal{U}p\,X}(S)\} \\
&= \cup\downarrow\{\lambda_X(T), T \in \{T \in \mathcal{D}n\,\mathcal{U}p\,X \mid \forall\, s \in S,\ T \cap s \neq \emptyset\} \\
&= \cup\{\lambda_X(T), T \in \{T \in \mathcal{D}n\,\mathcal{U}p\,X \mid \forall\, s \in S,\ T \cap s \neq \emptyset\} \text{ by lemma A.1} \\
&= \{t \in \mathcal{D}n\,X \mid \exists\, T \in \mathcal{D}n\,\mathcal{U}p\,X,\ (\forall\, s \in S,\ T \cap s \neq \emptyset) \wedge (\forall\, x \in T,\ x \cap t \neq \emptyset)\}
\end{aligned}$$

Now take $t \in \mathcal{D}n\,X$, and suppose $\forall\, s \in S,\ t \cap \cup s \neq \emptyset$. For each $s \in S$, fix $y_s \in s$ and $x_s \in y_s$ such that $x_s \in t$. Define $T = \uparrow\{y_s, s \in S\} = \downarrow\{y_s, s \in S\}$, by definition we have $T \in \mathcal{D}n\,\mathcal{U}p\,X$. Moreover, for any $s \in S$, we have $y_s \in s \cap T$ and so $\forall\, s \in S,\ T \cap s \neq \emptyset$. Also, for a given $y \in T$, there is some $s \in S$ such that $y \supseteq y_s$. But then $t \cap y \supseteq t \cap y_s \supsetneq \emptyset$. Therefore, $t \in \mu^{\mathcal{U}p}_{\mathcal{D}n\,X} \circ \mathcal{U}p(\lambda_X) \circ \lambda_{\mathcal{U}p\,X}(S)$.

Conversely, take $t \in \mathcal{D}n\,X$ and suppose $\exists\, T \in \mathcal{D}n\,\mathcal{D}n\,X,\ (\forall\, s \in S,\ T \cap s \neq \emptyset) \wedge (\forall\, x \in T,\ x \cap t \neq \emptyset)$. Take such a $T$, and some $s \in S$. By hypothesis, there is a $y_s \in T \cap s$, and because $y_s \in T$, there is some $x_s \in y_s \cap t$. But then $x_s \in \cup s$, and so $t \cap \cup s \neq \emptyset$. Thus, $t \in \lambda_X \circ \mathcal{D}n(\mu^{\mathcal{U}p}_X)(S)$.

By double inclusion, the diagram commutes.

## A.4 Monad Structure of $\mathcal{A}lt$

The construction we used to construct the monad $\mathcal{A}lt$ from a monad and an adjunction is not ad-hoc. Instead it is a consequence of the three following facts, which are classical results on monads and adjoints (see for instance [10, chapter 4, 6]):

**Theorem A.7 (Monad arising from an adjunction)**
Given an adjunction $\mathcal{F} \vdash \mathcal{G}$ with unit $\eta$, the functor $\mathcal{G} \circ \mathcal{F}$ can be equipped with a monad structure, whose unit is the unit of the adjunction.

The multiplication of the monad can also be described in term of the adjunction, but not in a simple way, so we leave this out.

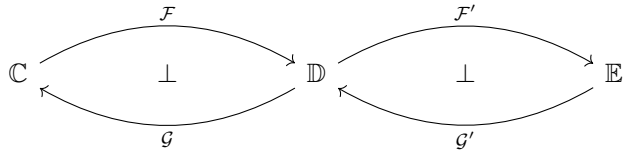
**Proposition A.8 (Adjoints arising from a monad)**
Given a monad $(\mathcal{T}, \eta, \mu)$ on a category $\mathbb{C}$, the monad arises from the following adjunction:

$$\mathbb{C} \underset{\mathcal{U}}{\overset{\mathcal{F}}{\rightleftarrows}} \mathcal{EM}(\mathcal{T}) \qquad (\perp)$$

Where $\mathcal{EM}(T)$ is the category of algebras for $\mathcal{T}$, $\mathcal{F}(X) = (\mathcal{T}X, \mu_X)$, $\mathcal{F}(f) = T(f)$, $\mathcal{U}'(X, f) = X$ and $\mathcal{U}'(\alpha) = \alpha$.

The exact details of the two adjoint functors is not very relevant, the important part is that the monad arising from the adjunction is $\mathcal{T}$.

**Proposition A.9 (Composition of adjoints)**
If $\mathcal{F} \dashv \mathcal{G}$ is an adjunction between $\mathbb{C}$ and $\mathbb{D}$, with unit $\eta$ and counit $\varepsilon$ and $\mathcal{F}' \dashv \mathcal{G}'$ is an adjunction between $\mathbb{D}$ and $\mathbb{E}$ with unit $\eta'$ and counit $\varepsilon'$, that is we are in the following situtation:



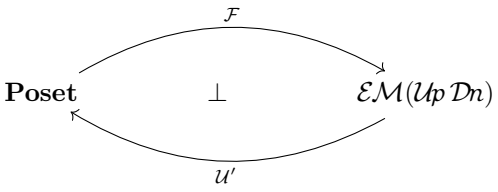

then $\mathcal{F}' \circ \mathcal{F} \dashv \mathcal{G} \circ \mathcal{G}'$ is an adjunction between $\mathbb{C}$ and $\mathbb{E}$, with unit

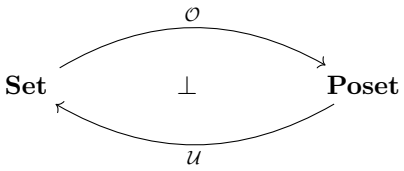
$$\eta'' : \mathrm{id}_{\mathbb{C}} \overset{\eta}{\Rightarrow} \mathcal{G} \circ \mathcal{F} \overset{\mathcal{G}\eta'\mathcal{F}}{\Rightarrow} \mathcal{G}\mathcal{G}'\mathcal{F}'\mathcal{F}$$

and counit

$$\mu'' : \mathcal{F}'\mathcal{F}\mathcal{G}\mathcal{G}' \overset{\mathcal{F}'\varepsilon\mathcal{G}'}{\Rightarrow} \mathcal{F}'\mathcal{G}' \overset{\varepsilon'}{\Rightarrow} \mathrm{id}_{\mathbb{E}}$$

Now the monadic structure for $\mathcal{A}lt$ can be constructed in three steps:

1. from $\mathcal{U}p\,\mathcal{D}n$ get an adjunction 

   using proposition A.8
2. compose this adjunction with 

   using proposition A.9
3. from this composite adjunction, get a monad on **Set** using propoition A.7

And this monad on **Set** is exactly the monad on $\mathcal{A}lt$ described at the end of section 4.2.

# B Thanks

I would like to thank Jurriaan Rot for being my guide in the world of coalgebras and pointing me again and again in the right direction, Luigi Santocanale whom I never met in person but who still gave me the right thing to look at for the distributive law, Joshua Moerman for his paper summing up all the troubles people have had on the same problem as me before, and Alexandre Goy, for keeping the mood up in the office.

*An erroneous monad structure on double covariant powerset*
*Bartek Klin. June 2016*

The following mistake in the literature has been noticed independently by Filippo Bonchi and Joost Winter. I made the mistake with Jurriaan Rot in [1], but it can be traced back to [3] and further to [2].

**Mistake by Klin and Rot, 2015 [1]**

Consider the covariant powerset monad $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$. (In [1] we consider the finite powerset instead, but the mistake is the same.) In [1], aiming to model alternating automata, we claim the existence of a monad-over-monad distributive law:

$$\lambda : \mathcal{P}\mathcal{P} \Longrightarrow \mathcal{P}\mathcal{P}$$

which could be formally defined by:

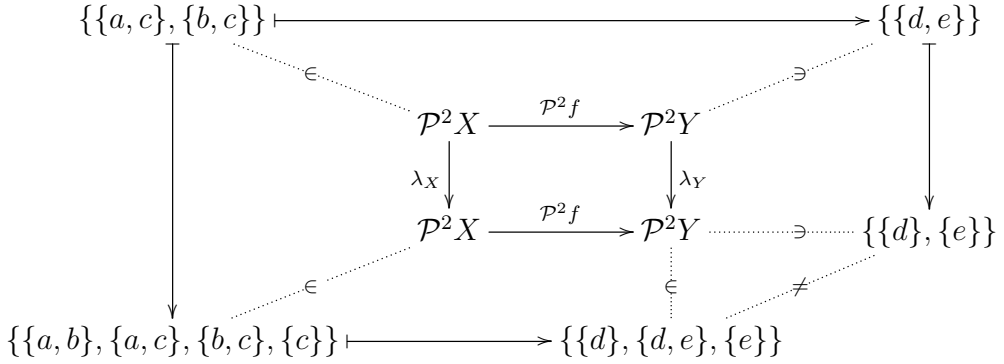
$$\lambda_X(\mathcal{A}) = \left\{ \{a_A \mid A \in \mathcal{A}\} \mid (a_A)_{A\in\mathcal{A}} \in \prod_{a\in\mathcal{A}} A \right\} \tag{1}$$

for $\mathcal{A} \subseteq \mathcal{P}X$. In words, given a family $\mathcal{A}$ of subsets of $X$, $\lambda_X$ returns the family of subsets obtained by picking a single element from every set in $\mathcal{A}$ in every possible way.

The mistake in this is that $\lambda$ is not a natural transformation. For a counterexample, shown to me by Joost Winter, consider

$$X = \{a, b, c\} \qquad Y = \{d, e\} \qquad f(a) = f(b) = d \qquad f(c) = e.$$

The naturality square for $f : X \to Y$ does not commute, as shown here:

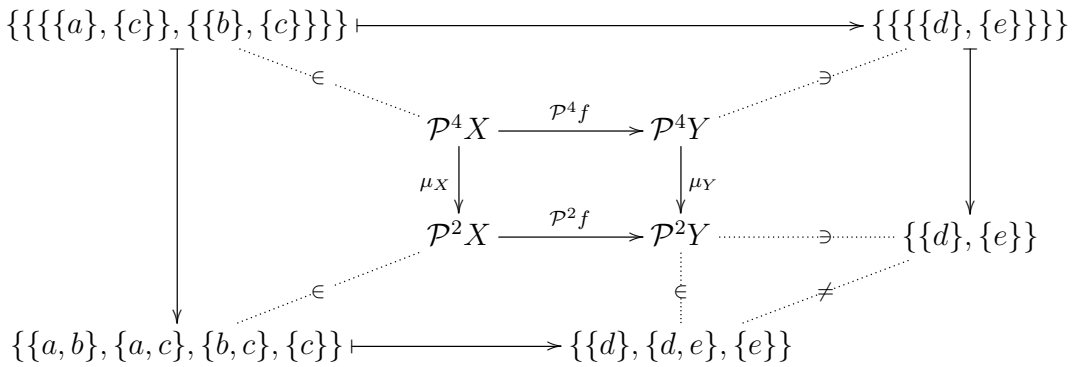


**Mistake by Manes and Mulry, 2007 [3]**

The definition in (1) is used in [3, pages 183-184] to define a monad structure on $\mathcal{PP}$ in the usual way. The (purported) monad multiplication $\mu : \mathcal{P}^4 \to \mathcal{P}^2$ that arises is explicitly defined by:

$$\mu_X(\mathbb{A}) = \left\{ \bigcup_{\mathcal{A}\in\Lambda} S_{\mathcal{A}} \mid \Lambda \in \mathbb{A},\ (S_{\mathcal{A}})_{\mathcal{A}\in\Lambda} \in \prod_{\mathcal{A}\in\Lambda} \mathcal{A} \right\}$$

for any $\mathbb{A} \in \mathcal{P}^4X$. The calculation of $\mu$ from $\lambda$ is adequate and, as expected now, $\mu$ is not a natural transformation either. The previous counterexample translates to, for the same $f : X \to Y$:




**Mistake by Manes, 2003 [2]**

In [3] naturality of $\lambda$ is actually inferred from the naturality of the unit and multiplication on the monad $\mathcal{PP}$. That naturality, as we know now, fails. In [3], the reader is referred to [2, pages 76–79] for a proof that $\mathcal{PP}$ is a monad. There, the monad is defined in terms of a Kleisli triple $(\mathcal{PP}, \eta, (-)^{\#})$ and the monad multiplication $\mu$ is derived from that in the usual sense.

Given a function $f : X \to \mathcal{PP}Y$, a function $f^{\#} : \mathcal{PP}X \to \mathcal{PP}Y$ is defined by:

$$C \in f^{\#}(\mathcal{A}) \iff \exists A \in \mathcal{A}.\exists (C_x \in f(x))_{x\in A}.\ C = \bigcup_{x\in A} C_x$$

for $\mathcal{A} \subseteq \mathcal{P}X$. Here, the second existential quantifier means that "there exists a family $(C_x)_{x\in A}$ such that every $C_x$ belongs to $f(x)$".

Viewing this $(-)^{\#}$ as a Kleisli lifting and the obvious $\eta_X(x) = \{\{x\}\}$ as the unit, the usual construction gives a monad struture on $\mathcal{PP}$ as described

in [3]. Since that structure is wrong, one expect problems with the Kleisli triple, and indeed the axiom:

$$(g^{\#}f)^{\#} = g^{\#}f^{\#} \qquad \text{for } f : X \to \mathcal{P}\mathcal{P}Y,\ g : Y \to \mathcal{P}\mathcal{P}Z$$

fails.

In [2] on pages 78–79 a proof of the axiom is attempted. The left-hand side is rewritten as:

$$\begin{aligned} C \in (g^{\#}f)^{\#}(\mathcal{A}) &\iff \exists A \in \mathcal{A}.\exists (C_x \in g^{\#}(f(x)))_{x\in A}.\ C = \bigcup_{x\in A} C_x \\ &\iff \exists A \in \mathcal{A}.\exists (B_x \in f(x))_{x\in A}.\exists (C_{x,y} \in g(y))_{x\in A, y\in B_x}.C = \bigcup_{x\in A}\bigcup_{y\in B_x} C_{x,y} \end{aligned}$$

and this transformation is correct. The right-hand side is rewritten as:

$$\begin{aligned} C \in (g^{\#}(f^{\#}(\mathcal{A})) &\iff \exists B \in f^{\#}(\mathcal{A}).\exists (C_y \in g(y))_{y\in B}.\ C = \bigcup_{y\in B} C_y \\ &\iff \exists A \in \mathcal{A}.\exists (B_x \in f(x))_{x\in A}.\exists (C_y \in g(y))_{x\in A, y\in \bigcup_{x\in A} B_x}.C = \bigcup_{y\in \bigcup_{x\in A} B_x} C_y \end{aligned}$$

And this is also correct. However, in the last equality on page 78, this is then equated to the left-hand side, and this is incorrect. Intuitively, looking at the third existential quantifiers on both sides above, the equality will not hold if the family $(B_x)_{x\ inA}$ contains some overlapping sets.

Indeed, the axiom fails for the following data:

$$\begin{gathered} X = \{1,2\} \qquad Y = \{*\} \qquad Z = \{a,b\} \\ f(1) = f(2) = \{\{*\}\} \qquad g(*) = \{\{a\},\{b\}\} \end{gathered}$$

To see this, calculate

$$C \in f^{\#}(\{\{1,2\}\}) \iff \exists A = \{1,2\}.\exists B_1, B_2 = \{*\}.C = B_1 \cup B_2$$

so $f^{\#}(\{\{1,2\}\}) = \{\{*\}\}$. Further:

$$C \in g^{\#}(\{\{*\}\}) \iff \exists A = \{*\}.\exists B_* \in \{\{a\},\{b\}\} C = B_*$$

so $g^{\#}(f^{\#}(\{\{1,2\}\})) = g^{\#}(\{\{*\}\}) = \{\{a\},\{b\}\}$. On the other hand $g^{\#}f : X \to \mathcal{P}\mathcal{P}Z$ is defined by:

$$g^{\#}f(1) = g^{\#}f(2) = g^{\#}(\{\{*\}\}) = \{\{a\},\{b\}\}$$

So calculate:

$$C \in (g^{\#} f)^{\#}(\{\{1,2\}\}) \iff \exists A = \{1,2\}.\exists B_1, B_2 \in \{\{a\},\{b\}\}.C = B_1 \cup B_2$$

therefore

$$(g^{\#} f)^{\#}(\{\{1,2\}\}) = \{\{a\},\{b\},\{a,b\}\}$$

hence $(g^{\#} f)^{\#}(\{\{1,2\}\}) \neq g^{\#}(f^{\#}(\{\{1,2\}\}))$.

There is a slight lack of consequence in these counterexamples: while the ones for $\lambda$ and $\mu$ are clearly two versions of the same counterexample, the one for $(-)^{\#}$ seems different. Of course one can try to translate counterexamples between the monad and the Kleisli triple settings, but the results both ways look more complicated then the ones I have above. Perhaps there is a good explanation of this (do such counterexamples come in pairs?), but I do not know what it could be.

# References

# References

[1] B. Klin and J. Rot. "Coalgebraic Trace Semantics via Forgetful Logics". In: *FOSSACS 2015, Proceedings*. Ed. by A. Pitts. Springer Berlin Heidelberg, 2015, pp. 151–166. DOI: 10.1007/978-3-662-46678-0_10.

[2] E. Manes. "Monads of sets". In: *Handbook of Algebra* 3 (2003), pp. 67–153. DOI: 10.1016/S1570-7954(03)80059-1.

[3] E. Manes and P. Mulry. "Monad compositions I: general constructions and recursive distributive laws". In: *Theory and Applications of Categories*. Vol. 18. 7. 2007, pp. 172–208.