W. de Weijer

# A decision procedure for weighted automata equivalence

Bachelor thesis

June 21, 2019

Thesis supervisors: M. Bonsangue
M. Streng


Leiden University
Mathematical Institute
Leiden Institute of Advanced Computer Science

# Contents

# 1 Introduction

Automata provide models for many different kinds of state-based computations. One kind of automata are the weighted automata, which can be used to express a weight, cost, or probability that a certain computation step will happen.

Due to the large variety of existing automata, category theory can be used as a strong tool in their general analysis. This is done using the theory of coalgebras, developed since only 1988 [Acz88].

In this thesis we will develop all the required theory for an audience of mathematicians and computer scientists alike. This will culminate in an algorithm that decides whether two weighted automata have the same behaviour by using coalgebras and we will give a proof of its correctness.

# 2 Theory

In Section 3 we will need some ideas that may be unfamiliar to a mathematician or computer scientist. We therefore summarise the necessary bits in the following two sections, either of which can be safely skipped by a reader who is already familiar with its contents.

In Section 2.1 we discuss the idea of automata and give several examples, concluding with weighted automata. For a complete introduction to automata theory we refer to [Mar03]. For weighted automata, [DK12] gives a short introduction.

Section 2.2 introduces the category theoretical concepts of algebras and coalgebras. We show how algebras can model algebraic structure and induction, and how, dually, coalgebras can model automata. The first few chapters of [Rot15] and [Win14] provide a more complete treatment of coalgebra. For a coalgebraic treatment of weighted automata, we use [BBB$^+$12].

## 2.1 Automata

One of the earliest mathematical models concretely describing the most general idea of computation was Alan Turing's "universal computing machine", also called a Turing machine. We will not reproduce a precise definition here, but there is overwhelming empirical evidence that a Turing machine can compute exactly what a person can compute (the *Church–Turing thesis*). Perhaps ironically, an important application of the Turing machine was proving that certain problems could *not* be algorithmically solved, most notably the problem of deciding whether a given Turing machine would eventually stop or not (the *halting problem*). This meant that in order to analyse the behaviour of computing machines themselves, more restricted models had to be constructed. This has led to a very wide range of automata varying in computational strength, the kinds of analyses that can be applied to them, and context in which they are used.

A common notion among many automata is the presence of a set of states, akin to the vertices in a flowchart diagram. Here each state holds information on how to act when input is received, for example to halt the automaton, to give output, or to move to a new state.

A very well-known type of automaton is the deterministic finite automaton. Let $\Sigma$ be a finite set called the **input alphabet**. A **deterministic finite automaton** (or DFA) over $\Sigma$ is a tuple $(Q, q_0, A, \delta)$ where

- $Q$ is a finite set of **states**,
- $q_0 \in Q$ is the **initial** state,
- $A \subseteq Q$ is the set of **accepting** states,
- $\delta : Q \times \Sigma \to Q$ is the **transition** function.

For any state $q \in Q$ and letter $\sigma \in \Sigma$, we interpret $\delta(q, \sigma)$ as the state to which the DFA moves if it is in state $q$ and receives the input $\sigma$. When an accepting state $q \in A$ is reached is this manner and there is no more remaining input, the automaton will halt and output that the given input sequence was accepted.

Figure 1 shows an example of a finite automaton over the input alphabet $\Sigma = \{a, b\}$. Its initial state is denoted with a "start" arrow and its accepting states with a double circle. An example input string accepted by this automaton is *abaa*.

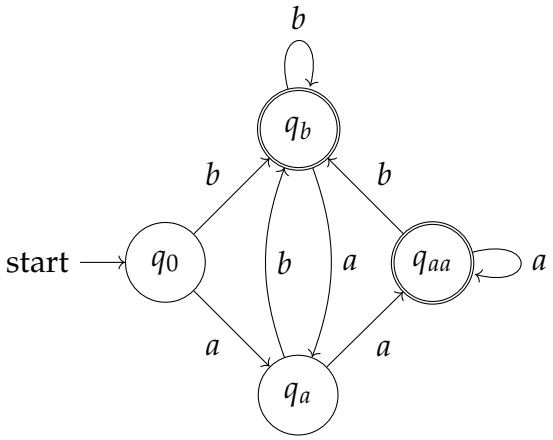



Figure 1: A finite automaton

We write $\Sigma^*$ for the **free monoid** generated by $\Sigma$ with $\varepsilon$ as its identity, i.e. the set of finite sequences (strings) of elements of $\Sigma$ where the operation is given by string concatenation and where $\varepsilon$ is the unique string of length zero.

The **extended transition function** $\delta^* : Q \times \Sigma^* \to Q$ is defined by induction as

$$\begin{aligned}(q, \varepsilon) &\mapsto q \\ (q, \sigma y) &\mapsto \delta^*(\delta(q, \sigma), y)\end{aligned}$$

where $\sigma \in \Sigma$ and $y \in \Sigma^*$. A string $x \in \Sigma^*$ is **accepted** by a DFA $M$ if

$$\delta^*(q_0, x) \in A.$$

Otherwise, the string is **rejected**. The **language** accepted by $M$ is the set

$$l(M) := \{x \in \Sigma^* \mid x \text{ is accepted by } M\} \in \mathcal{P}(\Sigma^*)$$

where $\mathcal{P}$ denotes the power set.

Two automata $M$, $N$ are called **language equivalent**, written as $M \sim_l N$, when $l(M) = l(N)$.

As promised, DFA are computationally much weaker than Turing machines. For example there is no automaton $M$ such that

$$l(M) = \{a^n b^n \mid n \geq 0\},$$

where $a, b \in \Sigma$ are different. Assume that such an $M$ exists and let $n = |Q|$ be the number of states. Then the sequence of states $q_0, \ldots, q_{2n}$ visited as the string $a^n b^n$ is read must contain some duplicate $q_i = q_j$ with $i < j \leq n$.

$$q_0 \xrightarrow{a} \cdots \xrightarrow{a} q_i \xrightarrow{a} \cdots \xrightarrow{a} q_j \xrightarrow{a} \cdots \xrightarrow{a} q_n \xrightarrow{b} \cdots \xrightarrow{b} q_{2n}$$

But this means that the subsequence $q_i, \ldots, q_j$ can be skipped, so that the word $a^i a^{n-j} b^n$ is also accepted by the automaton. But $i + n - j \neq n$ and so $a^i a^{n-j} b^n \notin L(M)$ which is a contradiction. This shows that the main limitation of DFA are their lack of a potentially infinite amount of "memory".

It turns out that the only languages that can be accepted by finite automata are precisely the **regular languages** $\mathcal{R} \subset \mathcal{P}(\Sigma^*)$, defined inductively as follows. For all regular languages $l, l_1, l_2 \in \mathcal{R}$ and every letter $\sigma \in \Sigma$ we have

- the empty language $\varnothing \in \mathcal{R}$,
- the singletons $\{\sigma\} \in \mathcal{R}$ and $\{\varepsilon\} \in \mathcal{R}$,
- the union $l_1 \cup l_2 \in \mathcal{R}$,
- the **concatenation**

$$l_1 \cdot l_2 := \{xy \mid x \in l_1,\, y \in l_2\} \in \mathcal{R},$$

- and the **Kleene star**

$$l^* := \bigcup_{n \geq 0} l^n \in \mathcal{R}$$

  where

$$l^0 = \{\varepsilon\},$$
$$l^{k+1} = l \cdot l^k.$$

### 2.1.1 Nondeterministic finite automata

The regular language accepted by the automaton in Figure 1 is $\{a, b\}^* \cdot \{aa, b\}$. It is however not so obvious how a DFA is obtained from a regular language, for example to construct the automaton from Figure 1 given the

regular language. It is often much easier and more efficient (in terms of memory) to first construct a less restricted kind of automaton and to then show these to be language equivalent to DFA.

Let $\Sigma$ again be a finite alphabet. A **nondeterministic finite automaton** (or NFA) over $\Sigma$ is a tuple $(Q, q_0, A, \delta)$ of a finite set of states $Q$, an initial state $q_0 \in Q$, a set of accepting states $A \subseteq Q$, and a transition function $\delta$. This time however the transition function is a function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

For any state $q \in Q$ and letter $\sigma \in \Sigma$, we interpret $\delta(q, \sigma)$ as the set of states to which the NFA can move if it is in state $q$ and receives the input $\sigma$. Again we define the extended transition function $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ by recursion as

$$\begin{aligned}(q, \varepsilon) &\mapsto \{q\} \\ (q, \sigma y) &\mapsto \bigcup \{\delta^*(q', y) \mid q' \in \delta(q, \sigma)\}\end{aligned}$$

where $\sigma \in \Sigma$ and $y \in \Sigma^*$. This time a string $x \in \Sigma^*$ is accepted if

$$\delta^*(q_0, x) \cap A \neq \varnothing.$$

We can construct a language equivalent DFA from a given NFA using the **powerset construction**. During the execution of a DFA we need to keep track of a single state in order to determine where to go next. In a DFA however we need to keep track of a set of states. We can thus simulate and NFA $M = (Q, q_0, A, \delta)$ using a DFA by letting every subset of $Q$ be a single state in the DFA. So we define the DFA $M'$ as

$$(\mathcal{P}(Q), \{q_0\}, A', \delta')$$

where

$$\begin{aligned}A' &= \{q' \in \mathcal{P}(Q) \mid q' \cap A \neq \varnothing\}, \\ \delta'(q', \sigma) &= \bigcup_{q \in q'} \delta(q, \sigma).\end{aligned}$$

Since the execution of $M'$ is completely the same as $M$, we have $L(M') = L(M)$.

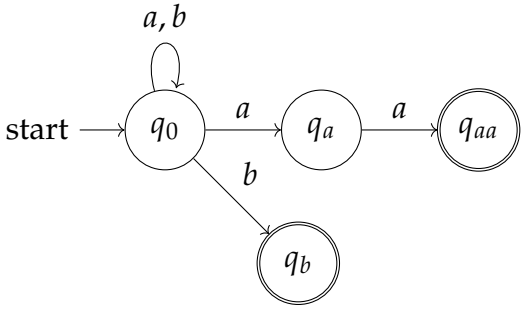



Figure 2: A nondeterministic finite automaton for the regular language $\{a, b\}^* \cdot \{aa, b\}$.

### 2.1.2 Weighted automata

Instead of using a transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ to define the structure of an NFA, we can also write this function as a subset of $Q \times \Sigma \times Q$, or rather as a function $\mathrm{wt} : Q \times \Sigma \times Q \to \mathbf{2}$, where $\mathbf{2} = \{\mathrm{true}, \mathrm{false}\}$, given by

$$\mathrm{wt}(q, \sigma, q') \coloneqq \begin{cases} \mathrm{true} & \text{if } q' \in \delta(q, \sigma), \\ \mathrm{false} & \text{otherwise.} \end{cases}$$

We also write the initial state $q_0$ and the accepting states $A$ as weight functions $\mathrm{in} : Q \to \mathbf{2}$ and $\mathrm{out} : Q \to \mathbf{2}$ given by

$$\mathrm{in}(q) \coloneqq \begin{cases} \mathrm{true} & q = q_0, \\ \mathrm{false} & \text{otherwise,} \end{cases} \qquad \mathrm{out}(q) \coloneqq \begin{cases} \mathrm{true} & q \in A, \\ \mathrm{false} & \text{otherwise.} \end{cases}$$

We can now use these weight functions to compute whether a word is accepted by the automaton using a function $\Sigma^* \to \mathbf{2}$ given by

$$\sigma_1\sigma_2 \cdots \sigma_n \mapsto \bigvee_{q_0, q_1, \ldots, q_n \in Q} \left( \mathrm{in}(q_0) \wedge \bigwedge_{1 \leq i \leq n} \mathrm{wt}(q_{i-1}, \sigma_i, q_i) \wedge \mathrm{out}(q_n) \right).$$

Doing this allows us to generalise NFA's by exchanging the structure $(\mathbf{2}, \vee, \wedge)$ for a more general algebraic structure.

Let $\Sigma$ be a finite alphabet and $R$ a set, a **weighted automaton** (or WA) over $\Sigma$ and $R$ is a tuple $(Q, \mathrm{in}, \mathrm{wt}, \mathrm{out})$ where $Q$ is a finite set of states, $\mathrm{wt} : Q \times \Sigma \times Q \to R$ is a function called the **transition weight function**, and $\mathrm{in}, \mathrm{out} : Q \to R$ are weight functions.

The least amount of structure we need on $R$ will be that of a rig: a "ring without negatives" (often also called a semiring). A **rig** $(R, +, \cdot, 0, 1)$ is a set $R$ equipped with binary operations of addition and multiplication, such that

- $(R, +, 0)$ is a commutative monoid with identity 0,
- $(R, \cdot, 1)$ is a monoid with identity 1,
- multiplication distributes over addition from the left and the right, and
- the annihilation law holds: $x \cdot 0 = 0 = 0 \cdot x$.

Now let $M$ be a weighted automaton over a rig $R$, then the **weighted language** of $M$ is the function $l_M : \Sigma^* \to R$ given by

$$l_M(\sigma_1 \cdots \sigma_n) \coloneqq \sum_{q_0, q_1, \ldots, q_n \in Q} \left( \mathrm{in}(q_0) \cdot \prod_{1 \leq i \leq n} \mathrm{wt}(q_{i-1}, \sigma_i, q_i) \cdot \mathrm{out}(q_n) \right).$$

Important examples of rigs are the natural numbers $(\mathbb{N}, +, \cdot, 0, 1)$, of course the Boolean domain $(\mathbf{2}, \vee, \wedge, \mathrm{false}, \mathrm{true})$ used in NFA's, and the tropical rig (also called the max-plus rig) $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$.

A weighted automaton over the rig $(\mathbb{N}, +, \cdot, 0, 1)$ allows us to "count" the number of paths of a certain length from the initial state to an accepted state. For example, the weighted automaton in Figure 3 with alphabet $\Sigma = \{*\}$ computes the Fibonacci sequence, i.e. $l_M(*^n) = F_n$.

Note however, that the results from Section 3 will require $R$ to be a field $\mathbb{K}$. In fact, the problem of language equivalence for weighted automata over the tropical rig with alphabet of at least two symbols is undecidable [Kro92].

## 2.2 Coalgebra

In order to study the wide range of automata in a general way, we need to define them using category theory. This leads to the concept of a coalgebra. But before this we first need to look at the historically earlier, *dual* concept of an algebra, which provides a general framework for algebraic structures. Examples of algebras are the eponymous algebras over a field or ring, but also groups and rings and fields themselves. More significant however is

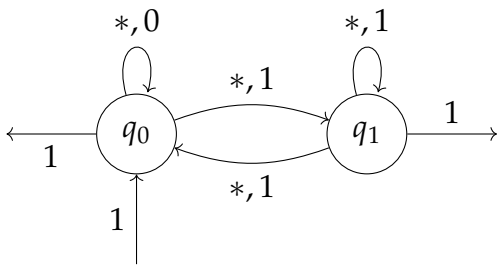



Figure 3: A weighted automaton which computes the Fibonacci sequence. The in and out functions are shown as arrows coming from and going to nowhere and $\mathrm{wt}(q, \sigma, q') = r$ is denoted as $q \xrightarrow{\sigma, r} q'$.

the use of algebras in studying the ideas of case analysis, pattern matching, recursion, and induction on objects like the natural numbers.

### 2.2.1 $F$-algebras

Given a functor $F : C \to C$ from a category $C$ to itself, an **$F$-algebra** is an object $X$ of $C$ together with an arrow $f : F(X) \to X$. An **$F$-homomorphism** of $F$-algebras $(X, f)$ to $(Y, g)$ is an arrow $h : X \to Y$ such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ \uparrow{\scriptstyle f} & & \uparrow{\scriptstyle g} \\ F(X) & \xrightarrow{F(h)} & F(Y) \end{array}$$

This gives us the **category of $F$-algebras**, where the objects are the $F$-algebras and the arrows are the $F$-homomorphisms.

Let us see how we can now define a monoid as an algebra. A monoid is a set $X$ together with a unit $e \in X$ and an operation $m : X \times X \to X$ such that $m$ is associative and $e$ is a unit for it. We first need to rewrite the unit to be a function $e : \mathbf{1} \to X$ where $\mathbf{1}$ is some singleton set $\{*\}$ so that $e(*) = e \in X$. We should also rewrite the associative and unit laws in

terms of commutative diagrams as follows.

$$\begin{array}{ccc} X \times X \times X & \xrightarrow{(m,\mathrm{id})} & X \times X \\ \downarrow{\scriptstyle (\mathrm{id},m)} & & \downarrow{\scriptstyle m} \\ X \times X & \xrightarrow{\quad m \quad} & X \end{array} \qquad \begin{array}{ccccc} 1 \times X & \xrightarrow{(e,\mathrm{id})} & X \times X & \xleftarrow{(\mathrm{id},e)} & X \times 1 \\ & \searrow^{\simeq} & \downarrow{\scriptstyle m} & \swarrow_{\simeq} & \\ & & X & & \end{array}$$

Lastly, we combine $m$ and $e$ into a single function $f : \mathbf{1} + X \times X \to X$ where $+$ is the disjoint union of sets. We can now define a monoid to be an $F$-algebra for the functor

$$\begin{aligned} F : \mathrm{Set} &\to \mathrm{Set} \\ X &\mapsto \mathbf{1} + X \times X \end{aligned}$$

such that the required laws are satisfied.

Given such a categorical definition of a monoid, we can use the power of category theory to simultaneously define topological monoids, or smooth monoids, or rings (being monoids *in* the category Ab of abelian groups). To do this we only need to generalise the singleton 1, the cartesian product $\times$, and the disjoint union $+$ to more general categories as the terminal object, the categorical product, and coproduct respectively.

To show that we can also use $F$-algebras to define the natural numbers, we will define the functor

$$\begin{aligned} N : \mathrm{Set} &\to \mathrm{Set} \\ X &\mapsto 1 + X. \end{aligned}$$

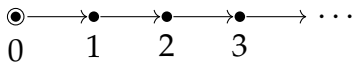
An algebra for this functor is again a pair $(X, f : \mathbf{1} + X \to X)$. Let's unpack this function $f$ as a pair of functions $(z, s)$ where $z : \mathbf{1} \to X$ (hence $z$ defines an element $z \in X$) and $s : X \to X$. An $N$-homomorphism $h : X \to Y$ must now satisfy the following commutative diagram.

$$\begin{array}{ccc} X & \xrightarrow{\quad h \quad} & Y \\ {\scriptstyle (z_X,s_X)}\uparrow & & \uparrow{\scriptstyle (z_Y,s_Y)} \\ \mathbf{1} + X & \xrightarrow{\mathbf{1}+h} & \mathbf{1} + Y \end{array}$$

We can also unpack this diagram to give us two laws that $h$ must satisfy:

- $h(z_X) = z_Y$ and
- $h(s_X(x)) = s_Y(h(x))$ for all $x \in X$.

We can think of many such algebras, however one is special: namely the algebra $(\mathbb{N}, (0, \mathrm{S}))$ where S is the *successor* function $n \mapsto n + 1$, diagrammatically:




This $N$-algebra is special since it is the initial object in the category of $N$-algebras. For an $N$-algebra $(X, (z_X, s_X))$, the initial $N$-homomorphism $i : \mathbb{N} \to X$ is recursively defined as

$$\begin{aligned} i(0) &\coloneqq z_X \\ i(S(n)) = i(n+1) &\coloneqq s_X(i(n)). \end{aligned}$$

We could in fact use the universal property of initiality to *define* recursion and even induction on the natural numbers (cf. Peano's axiom of induction).

### 2.2.2 $F$-coalgebras

A coalgebra now is the categorical *dual* of an algebra, meaning that the definition of a coalgebra is just the same as that for an algebra but with all the arrows "turned around".

Given a functor $F : C \to C$ on a category $C$, an **$F$-coalgebra** is an object $X$ of $C$ together with an arrow $f : X \to F(X)$. An **$F$-homomorphism** of $F$-coalgebras $(X, f)$ to $(Y, g)$ is an arrow $h : X \to Y$ such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle g} \\ F(X) & \xrightarrow{F(h)} & F(Y) \end{array}$$

This again gives us the **category of $F$-coalgebras**, where the objects are the $F$-coalgebras and the arrows are the $F$-homomorphisms.

Let us investigate some coalgebras and see how they can model automata. Here we let $F$ be a functor $F : \mathrm{Set} \to \mathrm{Set}$ and $\Sigma$ an input alphabet. For an

$F$-coalgebra $(X, f)$, we informally interpret the set $X$ as the set of states and $f : X \to F(X)$ as prescribing to each state its behaviour.

**Example 2.1.** Define $F(X) \coloneqq \mathbf{2} \times X^\Sigma$. An $F$-coalgebra $(X, \langle o, t \rangle)$ is a *deterministic automaton*. Here $o : X \to \mathbf{2}$ models the set of accepting states $A = \{x \mid o(x) = 1\}$, and $t : X \to X^\Sigma$ the transition function $\delta(x, \sigma) = t(x)(\sigma)$.

First note that, contrary to normal finite automata, we don't fix any initial state. Secondly, we allow the set of states $X$ to be infinite so that the category of $F$-coalgebras has a terminal object.

**Example 2.2.** Define $F(X) \coloneqq \mathbf{2} \times (\mathcal{P}_\omega(X))^\Sigma$, where $\mathcal{P}_\omega$ is the set of all finite subsets (which is a functor). An $F$-coalgebra $(X, \langle o, t \rangle)$ is a *nondeterministic automaton*. Again $o : X \to \mathbf{2}$ models the accepting states but this time the transition function $t : X \to (\mathcal{P}_\omega(X))^\Sigma$ defines the (finite) set of subsequent states.

We again allow $X$ to be infinite contrary to an NFA. However, the transition function must be *finite branching*, meaning that a single step may not put us in an infinite amount of states and hence that after a finite amount of steps, only a finite amount of states are reachable.

Using these constructions we can express the powerset construction in a very succinct way ([SBBR10]) as transforming an NFA $X \to (\mathcal{P}_\omega(X))^\Sigma$ into a DFA $\mathcal{P}_\omega(X) \to (\mathcal{P}_\omega(X))^\Sigma$ using the fact that the functor $\mathcal{P}_\omega : \mathrm{Set} \to \mathrm{Set}$ is a monad with multiplication map

$$\begin{aligned} \mu_{\mathcal{P}_\omega} : \mathcal{P}_\omega(\mathcal{P}_\omega(X)) &\to \mathcal{P}_\omega(X) \\ x &\mapsto \left\{ \bigcup x' \mid x' \in x \right\} \end{aligned}$$

and using the function

$$\begin{aligned} H : \mathcal{P}_\omega\big(X^\Sigma\big) &\to \mathcal{P}_\omega(X)^\Sigma \\ x &\mapsto \big(\sigma \mapsto \mathcal{P}_\omega(\mathrm{eval}_\sigma)(x)\big). \end{aligned}$$

Indeed the following function yields the same construction as in Section

2.1.1:

$$\begin{aligned}\mathcal{P}_\omega(X) &\xrightarrow{\mathcal{P}_\omega(t)} \mathcal{P}_\omega(\mathcal{P}_\omega(X)^\Sigma)\\ &\xrightarrow{H} \mathcal{P}_\omega(\mathcal{P}_\omega(X))^\Sigma\\ &\xrightarrow{\mu^\Sigma_{\mathcal{P}_\omega}} \mathcal{P}_\omega(X)^\Sigma.\end{aligned}$$

We will now define the coalgebra corresponding to weighted automata. For this we will need to define the free vector space over a set $X$

$$\mathbb{K}^{(X)} := \{f : X \to \mathbb{K} \mid f \text{ has finite support}\}.$$

Every element of $\mathbb{K}^{(X)}$ can be written as a $\mathbb{K}$-linear combination of elements of $X$. For sets $X$ and $Y$, a function $f : X \to Y$ defines a linear function

$$\begin{aligned}\mathbb{K}^{(f)} : \mathbb{K}^{(X)} &\to \mathbb{K}^{(Y)}\\ k_1x_1 + \cdots + k_nx_n &\mapsto k_1f(x_1) + \cdots + k_nf(x_n).\end{aligned}$$

**Definition 2.1.** The functor corresponding to *weighted automata* with input alphabet $\Sigma$ over the field $\mathbb{K}$ is

$$\mathcal{W}(X) := \mathbb{K} \times (\mathbb{K}^{(X)})^\Sigma.$$

A $\mathcal{W}$-coalgebra is a pair $(X, \langle o, t\rangle)$ where $o : X \to \mathbb{K}$ and $t : X \to (\mathbb{K}^{(X)})^\Sigma$.

Notice that, similarly to the coalgebra for NFA, the free vector space construction makes the transition function *finite branching*, even though the set $X$ can be infinite.

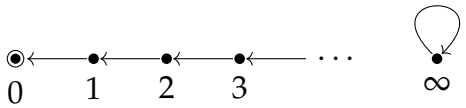
Just like the initial object for algebras, we will see that the *terminal* object plays a very important role. An $F$-coalgebra $(\Omega, \omega)$ is called **terminal** if it is terminal in the category of $F$-coalgebras, hence if for every $F$-coalgebra $(X, f)$ there exists a unique $F$-homomorphism $[\![-]\!]^F_X : X \to \Omega$.

A terminal object $(X, f)$ may not always exist, since if it does, $f$ must be an isomorphism (a result known as Lambek's theorem). Hence the covariant powerset functor $\mathcal{P} : \text{Set} \to \text{Set}$ has no terminal coalgebra because by Cantor's theorem there is no bijection between $X$ and $\mathcal{P}(X)$. This is the reason we used the finite powerset functor in example 2.2.

To see how terminal coalgebras work, let us consider again the functor $N(X) = \mathbf{1} + X$ from the last section. Its terminal coalgebra is formed by the set $\mathbb{N} \cup \{\infty\}$ together with the function

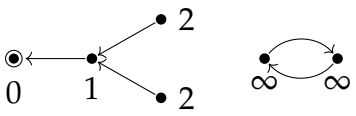
$$\begin{aligned} f : \mathbb{N} \cup \{\infty\} &\to \mathbf{1} + \mathbb{N} \cup \{\infty\} \\ 0 &\mapsto * \\ S(n) &\mapsto n \\ \infty &\mapsto \infty. \end{aligned}$$

We can see this coalgebra as a very simple automaton with 0 as the only accepting state and transitions as follows.




The terminal $N$-homomorphism from an $N$-coalgebra $(X, f)$ can informally be seen as assigning to each state $x \in X$ its behaviour $[\![x]\!]^N_X \in \mathbb{N} \cup \{\infty\}$: either accepting after $n \in \mathbb{N}$ steps, or being non-terminating ($\infty$). Any two states with the same behaviour are considered equivalent.

**Example 2.3.** Consider the following $N$-coalgebra. Every state $x$ is labelled with its image under the terminal $N$-homomorphism $[\![x]\!]^N_X$.




Two states $x_1, x_2 \in X$ in a coalgebra $(X, f)$ are called **$F$-behaviourally equivalent**, written as $x_1 \approx_F x_2$, if $[\![x_1]\!]^F_X = [\![x_2]\!]^F_X$.

In Section 2.1 we already saw another type of equivalence, namely weighted language equivalence $\sim_l$ between two automata with initial states. This equivalence can also be adapted to $\mathcal{W}$-coalgabra. It turns out however that weighted language equivalence is not the same as $\mathcal{W}$-behavioural equivalence, but $\approx_{\mathcal{W}} \subset \sim_l$ (by Theorem 1 and Proposition 2 in [BBB$^+$12]).

# 3 Deciding weighted automaton equivalence

In this Section we will reproduce the results from [BBB+12].

As noted in Section 2.2.2 we cannot use the coalgebraic notion of behavioural equivalence to decide weighted language equivalence $\approx_{\mathcal{W}}$ in general. To solve this problem we will provide a similar construction to the powerset construction, which we used to translate NFA into DFA.

The powerset construction worked by substituting the state space with the space of all possible combinations of states. Since for a weighted automaton a combination of states consists of a *linear combination* of states, the new state space will be the free vector space $\mathbb{K}^{(X)}$ generated by the original state space $X$.

**Definition 3.1.** A **linear weighted automaton** (or LWA) with input alphabet $\Sigma$ over the field $\mathbb{K}$ is a coalgebra for the functor $\mathcal{L} : \mathrm{Vect} \to \mathrm{Vect}$ with $\mathcal{L}(V) = \mathbb{K} \times V^{\Sigma}$.

As with WA, we can also assign a weighted language to each state of the LWA.

**Definition 3.2.** Let $(V, \langle o, t \rangle)$ be an $\mathcal{L}$-coalgebra. Given a state $v \in V$, define the **weighted language** $[\![v]\!]_V^{\mathcal{L}} : \Sigma^* \to \mathbb{K}$ by

$$[\![v]\!]_V^{\mathcal{L}}(w) := \begin{cases} o(v) & \text{if } w = \varepsilon, \\ [\![t(v)(\sigma)]\!]_V^{\mathcal{L}}(w') & \text{if } w = \sigma w'. \end{cases}$$

We would normally use the notation $l_V^{\mathcal{L}}$ for this, but we will see in Theorem 3.1 that $[\![-]\!]_V^{\mathcal{L}}$ is in fact the unique terminal $\mathcal{L}$-homomorphism. This also justifies the notation for the next definition.

**Definition 3.3.** For an $\mathcal{L}$-coalgebra $V$, two states $v_1, v_2 \in V$ are called **$\mathcal{L}$-behaviourally equivalent**, written as $v_1 \approx_{\mathcal{L}} v_2$, if $[\![v_1]\!]_V^{\mathcal{L}} = [\![v_2]\!]_V^{\mathcal{L}}$.

From a WA we can construct a linear weighted automaton using the *linearization* of a function $f : X \to V$ where $V$ is a vector space. Since $X$ gives a basis in $\mathbb{K}^{(X)}$, the linearization $f^\sharp : \mathbb{K}^{(X)} \to V$ is defined by $x \mapsto f(x)$. Given a weighted automaton $(X, \langle o, t \rangle)$ where $o : X \to \mathbb{K}$ and $t : X \to (\mathbb{K}^{(X)})^{\Sigma}$, define the linear weighted automaton $(\mathbb{K}^{(X)}, \langle o^\sharp, t^\sharp \rangle)$ where $o^\sharp : \mathbb{K}^{(X)} \to \mathbb{K}$ and $t^\sharp : \mathbb{K}^{(X)} \to (\mathbb{K}^{(X)})^{\Sigma}$.

Additionally, if $f : X \to Y$ is a $\mathcal{W}$-homomorphism of weighted automata, then $\mathbb{K}^{(f)} : \mathbb{K}^{(X)} \to \mathbb{K}^{(Y)}$ is an $\mathcal{L}$-homomorphism.

The following lemma shows that for a WA $X$ the weighted language $l_X$ corresponds to the weighted language $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^{(X)}}$ induced by the LWA $\mathbb{K}^{(X)}$. The proof is easy by an induction on the word length and will not be included.

**Lemma 3.1.** *Let $(X, \langle o, t\rangle)$ be a* WA *and $(\mathbb{K}^{(X)}, \langle o^\sharp, t^\sharp\rangle)$ the* LWA *constructed from it. Then for all $v = k_1x_1 + \cdots + k_nx_n$*

$$[\![v]\!]^{\mathcal{L}}_{\mathbb{K}^{(X)}} = k_1 \cdot l_X(x_1) + \cdots + k_n \cdot l_X(x_n).$$

We will now show that a terminal $\mathcal{L}$-coalgebra exists and that the behavioural equivalence $\approx_{\mathcal{L}}$ induced by it coincides precisely with weighted language equivalence $\sim_l$.

**Theorem 3.2.** *The category of $\mathcal{L}$-coalgabra has a terminal coalgebra $(\mathbb{K}^{\Sigma^*}, \langle \varepsilon, d\rangle)$ where*

$$\begin{aligned} \varepsilon : \mathbb{K}^{\Sigma^*} &\to \mathbb{K} \\ (l : \Sigma^* \to \mathbb{K}) &\mapsto l(\varepsilon) \end{aligned}$$

*is the* **empty function** *and*

$$\begin{aligned} d : \mathbb{K}^{\Sigma^*} &\to (\mathbb{K}^{\Sigma^*})^{\Sigma} \\ (l : \Sigma^* \to \mathbb{K}) &\mapsto \big(\sigma \mapsto (w \mapsto l(\sigma w))\big) \end{aligned}$$

*is the* **derivative function**. *Its terminal homomorphism is $[\![-]\!]^{\mathcal{L}}_V$.*

*Proof.* Let us first prove that $\varepsilon$ and $d$ are indeed linear functions. Let $l_1, l_2 \in \mathbb{K}^{\Sigma^*}$ be two weighted languages. For all letters $\sigma \in \Sigma$ and words $w \in \Sigma^*$ we have

$$\begin{aligned} d(l_1 + l_2)(\sigma)(w) &= (l_1 + l_2)(\sigma w) \\ &= l_1(\sigma w) + l_2(\sigma w) \\ &= d(l_1)(\sigma)(w) + d(l_2)(\sigma)(w). \end{aligned}$$

Secondly, let $k \in \mathbb{K}$ be a scalar and $l \in \mathbb{K}^{\Sigma^*}$ a weighted language. Now for all letters $\sigma \in \Sigma$ and words $w \in \Sigma^*$ we have

$$d(k \cdot l)(\sigma)(w) = (k \cdot l)(\sigma)(w) = k \cdot (l(\sigma)(w)) = k \cdot d(l)(\sigma)(w).$$

The proof for $\varepsilon$ is similar.

Now we will show that $(\mathbb{K}^{\Sigma^*}, \langle \varepsilon, d \rangle)$ is terminal. Let $(V, \langle o, t \rangle)$ be any $\mathcal{L}$-coalgebra. As noted, the weighted language function $[\![-]\!]_V^{\mathcal{L}} : V \to \mathbb{K}^{\Sigma^*}$, will be the terminal $\mathcal{L}$-homomorphism. Indeed, the required diagram

$$\begin{array}{ccc} V & \xrightarrow{[\![-]\!]_V^{\mathcal{L}}} & \mathbb{K}^{\Sigma^*} \\ {\scriptstyle \langle o,t \rangle}\downarrow & & \downarrow{\scriptstyle \langle \varepsilon,d \rangle} \\ \mathbb{K} \times V^{\Sigma} & \xrightarrow{\mathrm{id} \times \left([\![-]\!]_V^{\mathcal{L}}\right)^{\Sigma}} & \mathbb{K} \times \left(\mathbb{K}^{\Sigma^*}\right)^{\Sigma} \end{array}$$

commutes since, for every $v \in V$, $\sigma \in \Sigma$ and $w \in \Sigma^*$ we have

$$\begin{gathered} \varepsilon\big([\![v]\!]_V^{\mathcal{L}}\big) = [\![v]\!]_V^{\mathcal{L}}(\varepsilon) = o(v), \quad \text{and} \\ \Big([\![t(v)]\!]_V^{\mathcal{L}}\Big)^{\Sigma}(\sigma)(w) = [\![t(v)(\sigma)]\!]_V^{\mathcal{L}}(w) \overset{\text{def.}}{=} [\![v]\!]_V^{\mathcal{L}}(\sigma w) = d\big([\![v]\!]_V^{\mathcal{L}}\big)(\sigma)(w). \end{gathered}$$

To prove the uniqueness of the terminal morphism $[\![-]\!]_V^{\mathcal{L}}$, let $h : V \to \mathbb{K}^{\Sigma^*}$ be an $\mathcal{L}$-homomorphism. Then the following diagram must commute.

$$\begin{array}{ccc} V & \xrightarrow{\quad h \quad} & \mathbb{K}^{\Sigma^*} \\ {\scriptstyle \langle o,t \rangle}\downarrow & & \downarrow{\scriptstyle \langle \varepsilon,d \rangle} \\ \mathbb{K} \times V^{\Sigma} & \xrightarrow{\mathrm{id} \times h^{\Sigma}} & \mathbb{K} \times \left(\mathbb{K}^{\Sigma^*}\right)^{\Sigma} \end{array}$$

Hence for all $v \in V$, $\sigma \in \Sigma$ and $w \in \Sigma^*$ we have

$$\begin{gathered} o(v) = \varepsilon(h(v)) = h(v)(\varepsilon), \quad \text{and} \\ h(v)(\sigma w) = d(h(v))(\sigma)(w) = h(t(v)(\sigma))(w). \end{gathered}$$

And so $h$ must be equal to $[\![-]\!]_V^{\mathcal{L}}$ by induction on $\Sigma^*$. □

Another kind of equivalence often employed for automata is that of bisimulation. A bisimulation is a binary relation on the set of states such that related states both "simulate" each other. The largest bisimulation is called a bisimilarity, denoted as $\simeq$. For some automata it provides a stronger (i.e. more distinguishing) relation than language equivalence. However we will see that for linear weighted automata, it coincides with language equivalence.

**Definition 3.4.** Let $(V, \langle o, t \rangle)$ be an LWA. A subspace $U \subseteq V$ is a **linear weighted bisimulation** if

1. $U \subseteq \ker(o)$,
2. for all $\sigma \in \Sigma$, we have $t(U)(\sigma) \subseteq U$.

Two states $v, w \in V$ are **bisimilar** if $v - w \in U$.

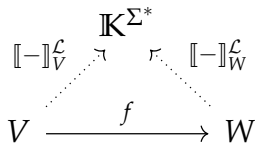
We will now show that the relation induced by the largest linear weighted bisimulation is the same as weighted language equivalence. The following lemma and theorem are proven in [BBB$^+$12].

**Lemma 3.3.** *If $f : V \to W$ is an $\mathcal{L}$-homomorphism, then $\ker(f)$ is a linear weighted bisimulation on $V$. Conversely, if $U \subseteq V$ is a linear weighted bisimulation, then there exists an LWA $W = V/U$ and an $\mathcal{L}$-homomorphism $f : V \to W$ such that $\ker(f) = U$.*

**Theorem 3.4.** *The largest linear weighted bisimulation on an LWA $V$ is precisely $\ker(\llbracket - \rrbracket_V^{\mathcal{L}})$.*

*Proof.* Since $\llbracket - \rrbracket_V^{\mathcal{L}} : V \to \mathbb{K}^{\Sigma^*}$ is an $\mathcal{L}$-homomorphism, by the first part of lemma 3.3 $\ker(\llbracket - \rrbracket_V^{\mathcal{L}})$ is a linear weighted bisimulation.

Now assume $U \subseteq V$ is a linear weighted bisimulation. By the second part of lemma 3.3 there is and LWA $W$ and an $\mathcal{L}$-homomorphism $f : V \to W$ such that $\ker(f) = U$. By the uniqueness of the terminal homomorphism, the following diagram commutes:




Hence $U = \ker(f) \subseteq \ker(\llbracket - \rrbracket_W^{\mathcal{L}} \circ f) = \ker(\llbracket - \rrbracket_V^{\mathcal{L}})$. □

We can now construct an algorithm for finding the largest bisimulation, and hence for deciding whether two states are language equivalent.

**Theorem 3.5.** *Let $(V, \langle o, t \rangle)$ be an LWA such that $\dim(V) < \infty$. Define the*

*sequence* $(U_i)_{i\geq 0}$ *of subspaces of* $V$ *by*

$$U_0 \coloneqq \ker(o),$$
$$U_{i+1} \coloneqq U_i \cap \bigcap_{\sigma\in\Sigma} \{v \in V \mid t(v)(\sigma) \in U_i\}.$$

*Then there is a* $j \leq \dim(V)$ *such that* $U_j = U_{j+1}$*. The largest linear bisimulation is* $\simeq_{\mathcal{L}} = U_j$*.*

*Proof.* For every $i$ we have $U_{i+1} \subseteq U_i$ and hence the required $j$ must exist. We now check that $U_j$ is a linear weighted bisimulation.

1. Indeed $U_j \subseteq U_0 = \ker(o)$.
2. Since $U_j = U_{j+1}$ we must have

$$U_j = U_j \cap \bigcap_{\sigma\in\Sigma} \{v \in V \mid t(v)(\sigma) \in U_j\}$$

   and hence $t(U_j)(\sigma) \subseteq U_j$ for all $\sigma \in \Sigma$.

To show that $U_j$ is the largest bisimulation, we show that every linear weighted bisimulation $U$ is also included in $U_j$. In fact $U \subseteq U_i$ for all $i$. We prove this by induction on $i$. Obviously $U \subseteq U_0 = \ker(o)$. Now assume that $U \subseteq U_i$. By definition $t(U)(\sigma) \subseteq U$ for every $\sigma \in \Sigma$, hence

$$U \subseteq \{v \in U_i \mid \forall \sigma \in \Sigma, t(v)(\sigma) \in U_i\} = U_{i+1}.$$

□

# References


[Acz88] Peter Aczel. *Non-Well-Founded Sets*. Stanford: CSLI Publications, 1988. CSLI Lecture Notes: Number 14.

[BBB+12] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, February 2012. URL: `http://dx.doi.org/10.1016/j.ic.2011.12.002`, `doi:10.1016/j.ic.2011.12.002`.

[DK12] Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Eric Pin, editor, *Automata: from Mathematics to Applications*, 2012. URL: `http://eiche.theoinf.tu-ilmenau.de/kuske/Submitted/weighted.pdf`.

[Kro92] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 101–112. Springer-Verlag, 1992. URL: `http://dl.acm.org/citation.cfm?id=646246.684713`.

[Mar03] John C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003.

[Rot15] Jurriaan Rot. *Enhanced coinduction*. PhD thesis, Leiden University, October 2015.

[SBBR10] Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing the powerset construction, coalgebraically. In *FSTTCS*, 2010.

[Win14] Joost Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. PhD thesis, Radboud University Nijmegen, July 2014.