A (co)algebraic theory of succinct automata

Gerco van Heerdt\textsuperscript{a}, Joshua Moerman\textsuperscript{a,b}, Matteo Sammartino\textsuperscript{a}, Alexandra Silva\textsuperscript{a}

\textsuperscript{a}University College London
\textsuperscript{b}Radboud University

Abstract

The classical subset construction for non-deterministic automata can be generalized to other side-effects captured by a monad. The key insight is that both the state space of the determinized automaton and its semantics—languages over an alphabet—have a common algebraic structure: they are Eilenberg-Moore algebras for the powersetgen monad. In this paper we study the reverse question to determinization. We will present a construction to associate succinct automata to languages based on different algebraic structures. For instance, for classical regular languages the construction will transform a deterministic automaton into a non-deterministic one, where the states represent the join-irreducibles of the language accepted by a (potentially) larger deterministic automaton. Other examples will yield alternating automata, automata with symmetries, CABA-structured automata, and weighted automata.

1. Introduction

Non-deterministic automata are often used to provide compact representations of regular languages. Take, for instance, the language

\[ \mathcal{L} = \{ w \in \{a, b\}^* \mid |w| > 2 \text{ and the 3}^{rd} \text{ symbol from the right is an } a \}. \]

There is a simple non-deterministic automaton accepting it (below, top automaton) and it is not very difficult to see that the smallest deterministic automaton (below, bottom automaton) will have 8 states.

\[ s_1 \xrightarrow{a,b} s_2 \xrightarrow{a,b} s_3 \xrightarrow{a} s_4 \]

\[ 1 \xrightarrow{a,b} 12 \xrightarrow{a} 123 \xrightarrow{a} 1234 \]

\[ 14 \xrightarrow{b} 13 \xrightarrow{a} 124 \xrightarrow{a} 134 \]

\[ b \]

\[ \star \]

This work was partially supported by ERC starting grant ProFoundNet (679127) and a Leverhulme Prize (PLP-2016-129).
The labels we chose for the states of the deterministic automaton are not coincidental—they represent the subsets of states of the non-deterministic automaton that would be obtained when constructing a deterministic one using the classical subset construction.

The question we want to study in this paper has as starting point precisely the observation that non-deterministic automata provide compact representations of languages and hence are more amenable to be used in algorithms and promote scalability. In fact, the origin of our study goes back to our own work on automata learning [15], where we encountered large nominal automata that, in order for the algorithm to work for more realistic examples, had to be represented non-deterministically. In other recent work [7, 3], different forms of non-determinism are used to learn compact representations of regular languages. This left us wondering whether other side-effects could be used to overcome scalability issues.

Moggi [16] introduced the idea that monads could be used as a general abstraction for side-effects. A monad is a triple \((T, \eta, \mu)\) in which \(T\) is an endofunctor over a category whose objects can be thought of as capturing pure computations. The monad is equipped with a unit \(\eta: X \rightarrow TX\), a natural transformation that enables embedding any pure computation into an effectful one, and a multiplication \(\mu: TTX \rightarrow TX\) that allows flattening nested effectful computations. Examples of monads capturing side-effects include powerset (non-determinism) and distributions (randomness).

Monads have been used extensively in programming language semantics (see e.g. [22] and references therein). More recently, they were used in categorical studies of automata theory [6]. One example of a construction in which they play a key role is a generalization of the classical subset construction to a class of automata [21, 20], which we will describe next.

The classical subset construction, connecting non-deterministic and deterministic automata, can be described concisely by the following diagram.

\[
\begin{array}{c}
X \xrightarrow{(\cdot)} \mathcal{P}(X) \xrightarrow{\cdot l} 2^{A^*} \\
\downarrow \delta \downarrow \delta^* \\
2 \times \mathcal{P}(X)^A \xrightarrow{\cdot \delta \cdot l^A} 2 \times (2^{A^*})^A
\end{array}
\]

We omit initial states and represent a non-deterministic automaton as a pair \((X, \delta)\) where \(X\) is the state space and \(\delta: X \rightarrow 2 \times \mathcal{P}(X)\) is the transition function which has in the first component the (non-)final state classifier. The language semantics of a non-deterministic automaton \((X, \delta)\) is obtained by first constructing a deterministic automaton \((\mathcal{P}(X), \delta^*)\) which has a larger state space consisting of subsets of the original state space and then computing the accepted language of the determinized automaton. The language map \(l\) associating the accepted language to a state is a universal map: for every deterministic automaton \((Q, Q \rightarrow 2 \times Q^A)\) the map \(l\) is the unique map into the automaton of languages \((2^{A^*}, 2^{A^*} \downarrow \cdot l^A \rightarrow 2 \times (2^{A^*})^A)\).

The universal property of the automaton of languages inspired the development of a categorical generalization of automata theory, including of the subset construction which we detail below. In particular, we can consider general automata as pairs \((X, X \xrightarrow{t} FX)\) where the transition dynamics \(t\) is parametric on a functor \(F\). Such pairs are usually called coalgebras for the functor \(F\) [18].
For a wide class of functors $F$, the category of coalgebras has a final object $(\Omega, \omega)$, the so-called final coalgebra, which plays the analogue role to languages.

The classical subset construction was generalized in previous work [21] by replacing deterministic automata with coalgebras for a functor $F$ and the powerset monad with a suitable monad $T$. As above, it can be summarized in a diagram:

$$
\begin{array}{c}
X \\
\delta
\end{array}
\xymatrix{
\ar[r]^n & TX \\
\ar@{->}[ur]^{\delta^*} & \Omega \\
FTX \\
\ar[r]*!/_1pc/{\delta^*} & F\Omega
}
$$

The monad $T$ will be the structure we will explore to enable succinct representations. The crucial ingredient in generalizing the subset construction was the observation that the target of the transition dynamics—$2 \times \mathcal{P}(-)$—and the set of languages—$2^{2^A}$—both have a complete join-semilattice structure. This enables one to define the determinized automaton as a unique lattice extension of the non-deterministic one, and, moreover, the language map $l$ preserves the semantics:

$l(\{s_1, s_2\}) = l(\{s_1\}) \cup l(\{s_2\})$.

This latter somewhat trivial observation was also exploited in the work of Bonchi and Pous [8] in defining an efficient algorithm for language equivalence of NFAs by using coinduction-up-to. Join-semilattices are precisely the Eilenberg-Moore algebras of the powerset monad, and one can show that if a functor has a final coalgebra in $\textbf{Set}$, this can be lifted to the category of Eilenberg-Moore algebras of a monad $T$ ($T$-algebras). This makes it possible to construct the more general diagram above, where the coalgebra structure is generalized using a functor $F$ and a monad $T$. The only assumptions for the existence of $T$-algebra maps $\delta^*$ and $l$ are the existence of a final coalgebra for $F$ in $\textbf{Set}$ and that $FTX$ can be given a $T$-algebra structure.

In this paper we ask the reverse question—given a deterministic automaton, if we assume the state space has a join-semilattice structure, can we build a corresponding succinct non-deterministic one? More generally, given an $F$-coalgebra in the category of $T$-algebras, can we build a succinct $FT$-coalgebra in the base category that represents the same behavior?

We will provide an abstract framework to understand this construction, based on previous work by Arbib and Manes [4]. Our abstract framework relies on alternative, more modern, presentation of some of their results. Due to our focus on set-based structures, we will conduct our investigation within the category $\textbf{Set}$, which enables us to provide effective procedures. This does mean that not all of the results due to Arbib and Manes will be given in their original generality. We present a comprehensive set of examples that will illustrate the versatility of the framework. We also discuss more algorithmic aspects that are essential if the present framework is to be used as an optimization, for instance as part of a learning algorithm.

After recalling basic facts about monads and structured automata in Section 2, the rest of this paper is organized as follows:

- In Section 3 we introduce a general notion of generators for a $T$-algebra, and we show that automata whose state space form a $T$-algebra—which we call $T$-automata—admit an equivalent $T$-succinct automaton, defined over generators. We also characterize minimal generators and give a condition under which they are globally minimal in size.
• In Section 4 we give an effective procedure to find a minimal set of generators for a $T$-algebra, and we present an algorithm that uses that procedure to compute the $T$-succinct version of a given $T$-automaton. The algorithm works by first minimising the $T$-automaton: the explicit algebraic structure allows states that correspond to algebraic combinations of other states to be detected, and then discarded when generators are computed.

• In Section 5 we show how the algorithm of Section 4 can be applied to “plain” finite automata—without any algebraic structure—in order to derive an equivalent $T$-succinct automaton. We conclude with a result about the compression power of our construction: it produces an automaton that is at least as small as the minimal version of the original automaton.

• Finally, in Section 6 we give several examples, and in Section 7 we discuss related and future work.

2. Preliminaries

Side-effects and different notions of non-determinism can be conveniently captured as a monad $T$ on a category $C$. A monad $T = (T, \mu, \eta)$ is a triple consisting of an endofunctor $T$ on $C$ and two natural transformations: a unit $\eta : \textbf{Id} \Rightarrow T$ and a multiplication $\mu : T^2 \Rightarrow T$. They satisfy the following laws:

$$\mu \circ \eta T = \text{id} = \mu \circ T \eta \quad \mu \circ \mu T = \mu \circ T \mu.$$

An example is the triple $(\mathcal{P}, \{-\}, \cup)$ where $\mathcal{P}$ denotes the powerset functor in $\textbf{Set}$ that assigns to each set the set of all its subsets, $\{-\}$ is the function that returns a singleton set, and $\cup$ is just union of sets.

Given a monad $T$, the category of $C^T$ of Eilenberg-Moore algebras over $T$, or simply $T$-algebras, has as objects pairs $(X, h)$ consisting of an object $X$, called carrier, and a morphism $h : TX \rightarrow X$ such that $h \circ \mu_X = h \circ T h$ and $h \circ \eta_X = \text{id}_X$. A $T$-homomorphism between two $T$-algebras $(X, h)$ and $(Y, k)$ is a morphism $f : X \rightarrow Y$ such that $f \circ h = k \circ Tf$.

We will often refer to a $T$-algebra $(X, h)$ as $X$ if $h$ is understood or if its specific definition is irrelevant. Given an object $X$, $(TX, \mu_X)$ is a $T$-algebra called the free $T$-algebra on $X$. Given an object $U$ and a $T$-algebra $(V, v)$, there is a bijective correspondence between $T$-algebra homomorphisms $TU \rightarrow V$ and morphisms $U \rightarrow V$: for a $T$-algebra homomorphism $f : TU \rightarrow V$, define $f^\dagger = f \circ \eta : U \rightarrow V$; for a morphism $g : U \rightarrow V$, define $g^\ddagger = v \circ T g : TU \rightarrow V$. Then $g^\ddagger$ is a $T$-algebra homomorphism called the free $T$-extension of $g$, and we have

$$f^{\dagger \ddagger} = f \quad g^{\dagger \ddagger} = g. \quad (1)$$

Furthermore, for all objects $S$ and morphisms $h : S \rightarrow U$,

$$g^{\ddagger} \circ Th = (g \circ h)^\ddagger. \quad (2)$$

Example 2.1. For the monad $\mathcal{P}$ the associated Eilenberg-Moore category is the category of (complete) join-semilattices. Given a set $X$, the free $\mathcal{P}$-algebra on $X$ is the join-semilattice $(\mathcal{P}X, \cup)$ of subsets of $X$ with the union operation as join.
Although some results are completely abstract, the central definition of minimal generators in Section 3 is specific to monads $T$ on the category $\mathbf{Set}$. Therefore we restrict ourselves to this setting. More precisely, we consider automata over a finite alphabet $A$ with outputs in a set $O$. In order to define automata in $\mathbf{Set}^T$ as (pointed) coalgebras for the functor $O \times (-)^A$, we need to lift this functor from $\mathbf{Set}$ to $\mathbf{Set}^T$. Such a lifting corresponds to a distributive law of $T$ over $O \times (-)^A$ [see e.g., 13]. A distributive law of the monad $T$ over a functor $F: \mathbf{Set} \to \mathbf{Set}$ is a natural transformation $\rho: TF \Rightarrow FT$ satisfying $\rho \circ \eta F = F \eta$ and $F \mu \circ \rho T \circ T \rho = \rho \circ \mu F$. In most examples we will define a $T$-algebra structure $\beta: TO \to O$ on $O$, which is well known to induce a distributive law $\rho: T(O \times (-)^A) \Rightarrow O \times T(-)^A$ given by

$$\rho_X = T(O \times X^A) \xrightarrow{(T\pi_1, T\pi_2)} TO \times T(X^A) \xrightarrow{\beta \times \rho_X} O \times T(X)^A$$

(3)

for any set $X$, where $\rho'(U)(a) = T(\lambda f: A \to X, f(a))$. In general, we assume an arbitrary distributive law $\rho: T(O \times (-)^A) \Rightarrow O \times T(-)^A$, which gives us the following notion of automaton.

**Definition 2.2** ($T$-automaton). A $T$-automaton is a triple $(X, i: 1 \to X, \delta: X \to O \times X^A)$, where $X$ is an object of $\mathbf{Set}^T$ denoting the state space of the automaton, $i$ is a function designating the initial state, and $\delta$ is a $T$-algebra map assigning an output and transitions to each state.

Notice that the initial state map $i: 1 \to X$ in the above definition is not required to be a $T$-algebra map. However, it corresponds to the $T$-algebra map $i^\delta: T1 \to X$. Thus, a $T$-automaton is an automaton in $\mathbf{Set}^T$.

The functor $F(X) = O \times X^A$ has a final coalgebra in $\mathbf{Set}^T$ [12] that can be used to define the language accepted by a $T$-automaton.

**Definition 2.3** (Language accepted). Given a $T$-automaton $(X, i: 1 \to X, \delta: X \to O \times X^A)$, the language accepted by $X$ is $l \circ i: 1 \to O^A$, where $l$ is the final coalgebra map. In the diagram below, $\omega$ is the final coalgebra.

$$\begin{array}{cccc}
1 & \xrightarrow{i} & X & \xrightarrow{\delta} & O^A \\
\downarrow & & \downarrow & & {} \\
O \times X^A & \xrightarrow{id \times \delta^A} & O \times (O^A)^A & \xrightarrow{\omega} & \omega(\varphi) = (\varphi(e), \lambda a \varphi(aw)) \\
\end{array}$$

(4)

$$l(x)(e) = \pi_1(\delta(x))$$

$$l(x)(aw) = l(\pi_2(\delta(x)))(a)(w)$$

We use $e$ to denote the empty word.

If the monad $T$ is finitary, then the category $\mathbf{Set}^T$ is locally finitely presentable, and hence it admits (strong epi, mono)-factorizations [2]. As in [4], we use these factorizations to quotient the state-space of an automaton under language equivalence. The transition structure, $\gamma$, is obtained by diagonalization via the factorization system. Diagrammatically:

$$\begin{array}{cccc}
1 & \xrightarrow{j} & X & \xleftarrow{\epsilon} & M & \xrightarrow{m} & O^A \\
\downarrow & & \downarrow & & \uparrow & & {} \\
O \times X^A & \xrightarrow{id \times \epsilon^A} & O \times M^A & \xrightarrow{id \times m^A} & O \times (O^A)^A & \xrightarrow{\omega} & \omega(\varphi) = (\varphi(e), \lambda a \varphi(aw)) \\
\end{array}$$

(4)

$$l(x)(e) = \pi_1(\delta(x))$$

$$l(x)(aw) = l(\pi_2(\delta(x)))(a)(w)$$

We use $e$ to denote the empty word.
Here the epi $e$ and mono $m$ are obtained by factorizing the final coalgebra map $l: X \to O^A$. We call the quotient automaton $(M, \bar{j}, \bar{\gamma})$ the *observable quotient* of $(X, i, \delta)$.

3. $T$-succinct automata

Given a $T$-automaton $X = (X, i, \delta)$, our aim is to obtain an equivalent automaton in $\mathbf{Set}$ with transition function $Y \to O \times T(Y)^A$, where $Y$ is smaller than $X$.$^1$ The key idea is to find *generators* for $X$. Our definition of generators is equivalent to the definition of a *scoop* due to Arbib and Manes [4, Section 7, Definition 8].

**Definition 3.1** (Generators for an algebra). We say that a set $G$ is a set of generators for a $T$-algebra $X$ whenever there exists a function $g: G \to X$ such that $g^\sharp: TG \to X$ is a split epi in $\mathbf{Set}$.

The intuition of requiring a split epi is that every element of $X$ can now be decomposed into a “combination” (defined by $T$) of elements of $G$. We show two simple results on generators, which will allow us to find initial sets of generators for a given $T$-algebra.

**Lemma 3.2.** The carrier of any $T$-algebra $X$ is a set of generators for it.

**Proof.** Let $TX \to X$ be the $T$-algebra structure on $X$. Then $\text{id}_X$ satisfies $\text{id}_X^\sharp = \chi$, and $\chi$ is a split epi because it is required to satisfy $\chi \circ \eta_X = \text{id}_X$. □

**Lemma 3.3.** Any set $X$ is a set of generators for the free $T$-algebra $TX$.

**Proof.** Follows directly from the fact that $\eta_X: X \to TX$ satisfies $\eta_X^\sharp = \text{id}_{TX}$. □

Once we have a set of generators $G$ for $X$, we can define an equivalent free representation of $X$, that is, an automaton whose state space is freely generated from $G$.

**Proposition 3.4** (Free representation of an automaton [4, Section 7, Proposition 9]). The free algebra $TG$ forms the state space of an automaton equivalent to $X$.

**Proof.** Let $g: G \to X$ witness $G$ being a set of generators for $X$ and let $s: X \to TG$ be a right inverse of $g^\sharp$. Recall that $X = (X, i, \delta)$ and define

$$
\begin{align*}
  j &= \begin{array}{c} 1 \end{array} X \xrightarrow{s} TG \\
  \gamma &= \begin{array}{c} G \end{array} X \xrightarrow{\delta} O \times X^A \xrightarrow{\text{id} \times s^A} O \times (TG)^A
\end{align*}
$$

Then $(TG, j, \gamma^\sharp)$ is an automaton. We will show that $g^\sharp: TG \to X$ is an automaton homomorphism. We have $g^\sharp \circ j = g^\sharp \circ s \circ i = i$, and, writing $F$ for the functor

$^1$Here, we are abusing notation and using $O$ and $A$ for both the objects in $\mathbf{Set}^T$ and in the base category $\mathbf{Set}$. In particular, we use $TC$ to also denote the free $T$-algebra over $C$. 

6
$O \times (-)^A$ and $\chi$ for the $T$-algebra structure on $X$,

\[
\begin{array}{c}
TG \xrightarrow{Tg} TX \xrightarrow{T\delta} TFX \xrightarrow{TFs} TFTG \xrightarrow{\rho} FT^2G \xrightarrow{F\mu} FTG \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
X \xrightarrow{\delta} FX \xrightarrow{id} FX \xrightarrow{s} FTX \xrightarrow{\rho} FTG^\delta \xrightarrow{Fg^\delta} FX
\end{array}
\]

commutes. Here $\circledast$ commutes because $\delta$ is a $T$-algebra homomorphism, $\circledcirc$ commutes by naturality of the distributive law $\rho$, and $\circledast$ commutes because $g^\delta$ is a $T$-algebra homomorphism. The triangle on the left unfolds the definition of $g^\delta$, and the remaining triangle commutes by $s$ being right inverse to $g^\delta$. Note that the composition in the top row of the diagram is $\gamma^\delta$. We conclude that $(TG, j, \gamma^\delta)$ accepts the same language as $X$.

The state space $TG$ of this free representation can be extremely large. Fortunately, the fact that $TG$ is a free algebra allows for a much more succinct version of this automaton.

**Definition 3.5 ($T$-succinct automaton).** Given an automaton of the form $(TX, i, \delta)$, where $TX$ is the free $T$-algebra on $X$, the corresponding $T$-succinct automaton is the triple $(X, i, \delta \circ \eta)$. The language accepted by the $T$-succinct automaton is the language $l \circ i$ accepted by $(TX, i, \delta)$:

\[
\begin{array}{c}
X \xrightarrow{\eta} TX \xrightarrow{i} O^A \\
\downarrow \quad \downarrow \\
O \times (TX)^A \xrightarrow{id \times t^A} O \times (O^A)^A
\end{array}
\]

The goal of our construction is to build a $T$-succinct automaton from a set of generators that is minimal in a way that we will define now. In what follows below we use the following piece of notation: if $U$ and $V$ are sets such that $U \subseteq V$, then we write $i_U^V$ for the inclusion map $U \to V$.

**Definition 3.6 (Minimal generators).** Given a $T$-algebra $X$ and a set of generators $G$ for $X$ witnessed by $g: G \to X$, we say that $r \in G$ is redundant if there exists a $U \in T(G \setminus \{r\})$ satisfying $(g \circ t_G^{G \setminus \{r\}})(U) = g(r)$; all other elements are said to be isolated [4]. We call $G$ a minimal set of generators for $X$ if $G$ contains no redundant elements.

---

2Arbib and Manes [4] define isolated elements only for the full set $X$ rather than relative to a set of generators for $X$. Our refinement plays an important role in finding a minimal set of generators.
A minimal set of generators is not necessarily minimal in size. However, under certain conditions this is the case. The following result was mentioned but not proved by Arbib and Manes [4], who showed that its conditions are satisfied for any finitely generated \( \mathcal{P} \)-algebra. We note that these conditions do not apply (in general) to any of the further examples in Section 6.

**Proposition 3.7.** If a \( T \)-algebra \( X \) is generated by the isolated elements \( I \) of the set of generators \( X \) (Lemma 3.2) with their inclusion map \( i^X_I \) and \( I \) is finite, then there is no set of generators for \( X \) smaller than \( I \), and every minimal set of generators for \( X \) has the same size as \( I \).

**Proof.** Let \( G \) be a set of generators for \( X \), and assume towards a contradiction that \( G \) is smaller than \( I \). Then there must be an \( i \in I \) such that there is no \( v \in G \) satisfying \( g(v) = i \). Let \( g' : G \rightarrow X \setminus \{i\} \) be pointwise equal to \( g \). Because \( g' \) is a split epi and thus surjective, there is a \( U \in TG \) such that \( g^U(U) = i \). Note that by (2),

\[
g^U = (\iota^X_X \circ g')^U = TG \xrightarrow{T(g')} T(X \setminus \{i\}) \xrightarrow{(\iota^X_X \setminus \{i\})^U} X.
\]

Then \((\text{id} \circ \iota^X_X \setminus \{i\})^U(T(g')(U)) = i\), contradicting the fact that \( i \) is isolated in the full set of generators \( X \). Thus, \( G \) cannot be smaller than \( I \). In fact, we see that for every \( i \in I \) there is a \( v \in G \) satisfying \( g(v) = i \). This yields a function \( h : I \rightarrow G \) such that \( g \circ h = \iota^X_i \).

Suppose \( G \) is a minimal set of generators, and take any \( v \in G \) not in the image of \( h \). We will show that \( v \) is redundant in \( G \). Since \( I \) constitutes a set of generators for \( X \), there exists a \( U \in TI \) such that \( (\iota^X_I)^U(U) = g(v) \). Then

\[
g^U(h(U)) = (g \circ h)^U(U) = (\iota^X_I)^U(U) = g(v).
\]

It follows that \( v \) is redundant in \( G \), which contradicts \( G \) being minimal. Therefore, \( h \) is surjective and \( G \) has the same size as \( I \).

\[\square\]

**4. \( T \)-minimization**

In this section we describe a construction to compute a “minimal” succinct \( T \)-automaton equivalent to a given \( T \)-automaton. This crucially relies on a procedure that finds a minimal set of generators by removing redundant elements one by one. All that needs to be done for specific monads is determining whether an element is redundant.

**Proposition 4.1** (Generator reduction). Given a \( T \)-algebra \( X \) and a set of generators \( G \) for \( X \), if \( r \in G \) is redundant, then \( G \setminus \{r\} \) is a set of generators for \( X \).

**Proof.** Let \( G' = G \setminus \{r\} \) and let \( g' : G' \rightarrow X \) be the restriction of \( g : G \rightarrow X \) to \( G' \). Since \( r \) is redundant, there is a \( U \in T(G') \) such that \( g'^U(U) = g(r) \). Define \( e : G \rightarrow T(G') \) by

\[
e(x) = \begin{cases} U & \text{if } x = r \\ \eta(x) & \text{if } x \neq r. \end{cases}
\]
We will show that \( g^\sharp \circ e = g \). Consider any \( x \in G \). If \( x = r \), then

\[
g^\sharp(e(x)) = g^\sharp(e(r)) = g^\sharp(U) = g(r) = g(x).
\]

If \( x \neq r \), then, using (1),

\[
g^\sharp(e(x)) = g^\sharp(\eta(x)) = g^\sharp = g'(x) = g(x).
\]

Let \( \chi : TX \to X \) be the algebra structure on \( X \) and take any right inverse \( s : X \to TG \) of \( g^\sharp \). Then

\[
g^\sharp \circ e^s \circ s = g^\sharp \circ \mu \circ Te \circ s \quad \text{(definition of } e^s) \\
= \chi \circ T(g^\sharp) \circ Te \circ s \quad \text{(} g^\sharp \text{ is a } T\text{-algebra homomorphism)} \\
= \chi \circ T(g^\sharp \circ e) \circ s \quad \text{(functoriality of } T) \\
= \chi \circ Tg \circ s \quad \text{(} g^\sharp \circ e = g \text{ as shown above)} \\
= g^\sharp \circ s \quad \text{(definition of } g^\sharp) \\
= \text{id}_x \quad \text{(} s \text{ is right inverse to } g^\sharp).
\]

We thus see that \( e^s \circ s \) is right inverse to \( g^\sharp \), which means that \( G' \) is a set of generators for \( X \).

If we determine that an element is isolated, there is no need to check this again later when the set of generators has been reduced. This is thanks to the following result.

**Proposition 4.2.** If \( G \xrightarrow{\sigma} X \) and \( G' \xrightarrow{\sigma'} X \) are sets of generators for a \( T \)-algebra \( X \) such that \( G' \subseteq G \) and \( g' \) is the restriction of \( g \) to the domain \( G' \), then whenever an element \( r \in G' \) is isolated in \( G \), it is also isolated in \( G' \).

**Proof.** We will show that redundant elements in \( G' \) are also redundant in \( G \). If \( r \in G' \) is isolated in \( G' \), then there exists \( U \in T(G' \setminus \{r\}) \) such that \( (g' \circ \iota_{G'}(r))^\sharp(U) = g'(r) \). Note that \( g' = g \circ \iota_{G'}(r) \). We have

\[
(g \circ \iota_{G'}(r))^\sharp(T(\iota_{G \setminus \{r\}}(U))) = (g \circ \iota_{G}^{G \setminus \{r\}}) \circ \iota_{G \setminus \{r\}}(r)^\sharp(U) \\
= (g \circ \iota_{G}^{G \setminus \{r\}}) \circ \iota_{G \setminus \{r\}}(r)^\sharp(U) \\
= (g' \circ \iota_{G \setminus \{r\}}(r))^\sharp(U) \\
= g'(r) \\
= g(r),
\]

so \( r \) is redundant in \( G \).

Finally, taking the observable quotient \( M \) of a \( T \)-automaton \( Q \) preserves generators, considering that the \( T \)-automaton homomorphism \( m : Q \to M \) is a split epi in \( \textbf{Set} \) under the axiom of choice.

**Proposition 4.3.** If \( Q \) and \( M \) are \( T \)-algebras, \( m : Q \to M \) is a \( T \)-algebra homomorphism that is a split epi in \( \textbf{Set} \), and \( G \xrightarrow{\sigma} Q \) is a set of generators for \( Q \), then \( G \xrightarrow{\sigma} Q \xrightarrow{m} M \) is a set of generators for \( M \).
Proof. Let \( a : TQ \to Q \) be the \( T \)-algebra structure on \( Q \) and \( b : TM \to M \) the one on \( M \). We have
\[
(m \circ g)^\sharp = b \circ T(m \circ g) = b \circ T(m) \circ T(g) = m \circ a \circ Tg = m \circ g^\sharp
\]
using that \( m \) is a \( T \)-algebra homomorphism. It is well known that compositions of split epis are split epis themselves, so \( G \) is a set of generators for \( M \).

Now we are ready to define the construction that builds a \( T \)-succinct automaton accepting the same language as a \( T \)-automaton.

Construction 4.4 (\( T \)-minimization). Starting from a \( T \)-automaton \((X, i, \delta)\), where \( X \) has a finite set of generators, we execute the following steps.

1. Take the observable quotient \((M, i_0, \delta_0)\) of \((X, i, \delta)\).
2. Compute a minimal set of generators \( G \) of \( M \) by starting from the full set \( M \) and applying Proposition 4.1.
3. Compute and return the corresponding \( T \)-succinct automaton as defined in Definition 3.5 via Proposition 3.4.

Generic minimization algorithms have been proposed in the literature. For example, Adámek et al. give a general procedure to compute the observable quotient [1], and König and Küpper provide a generic partition refinement algorithm for coalgebras, with a focus on instantiations to weighted automata [14]. None of these works provide any complexity analysis. Recently, Dorsch et al. [11] have presented a coalgebraic Paige–Tarjan algorithm and provided a complexity analysis for a class of functors in categories with image-factorization. These restrictions match well the ones we make, and therefore their algorithm could be applied in our first step. Given a finite set of generators \( G \), the loop in the second step involves considering each element of \( G \) and checking whether it is redundant. If so, we will remove the element from \( G \) and continue the loop. The redundancy check is the only part for which computability needs to be determined in each specific setting.

Example 4.5 (Join-semilattices). We give an example of the construction in the category \( \text{JSL} \) of complete join-semilattices. We start from a minimal \( \mathcal{P} \)-automaton (in \( \text{JSL} \)) that has 4 states and is depicted below on the left. The dashed blue lines indicate the JSL structure.

Since the automaton is minimal, it is isomorphic to its observable quotient. We start from the full set of generators \( \{\bot, x, y, z\} \). Note that \( z \) is the union of \( x \) and \( y \), so we can eliminate it. Additionally, \( \bot \) is the empty union and can be removed as well. Both \( x \) and \( y \) are isolated elements and form the unique minimal set of generators \( G = \{x, y\} \) (see the remark above Proposition 3.7).
These are exactly the join-irreducibles of $M$. They induce by Proposition 3.4 an automaton $(TG, j, \gamma)$, where $\gamma$ is the same transition structure as the above automaton, but with $\{x, y\}$ substituted for $z$; the initial state is the singleton set $\{x\}$. The $P$-succinct automaton corresponding to this minimal set of generators (Definition 3.5) is the non-deterministic automaton shown on the right.

Note that the definition of the automaton defined in Proposition 3.4 depends on the right inverse chosen for the extension of the generator map. When the original JSL automaton is reachable (every state is reached by some set of words, where a set of words reaches the join of the states reached by the words it contains), this right inverse may be chosen in such a way to recover the canonical residual finite state automaton (RFSA), as well as the simplified canonical RFSA, both due to Denis et al. [10]. Details are given in [23]. See [17] for conditions under which the canonical RFSA, referred to as the jiromaton, is a state-minimal NFA.

5. Main construction

In this section we present the main construction of the paper. Given a finite automaton $(X, i, \delta)$ in $\text{Set}$, i.e., an automaton where $X$ is finite, this construction builds an equivalent $T$-succinct automaton.

The first step is taking the reachable part $R$ of $X$ and converting this automaton into a $T$-automaton recognising the same language.

**Proposition 5.1.** Let $(TR, \hat{i}, \hat{\delta})$ be the $T$-automaton defined as follows:

\[
\begin{array}{ccc}
R & \xrightarrow{\eta^R \circ i} & TR \\
\downarrow{\eta^R} & & \downarrow{\eta^R} \\
O \times R^A & \xrightarrow{id \times \eta^A_R} & O \times T(R)^A \\
\end{array}
\]

Then $(R, i, \delta)$ and $(TR, \hat{i}, \hat{\delta})$ accept the same language.

**Proof.** The diagram above means that $\eta^R$ is a coalgebra homomorphism, and as such it preserves language. Explicitly: $x \in R$ accepts the same language as $\eta^R(x)$, which in particular holds for $i(*)$ and $\hat{i}(*).$ \hfill \square

Now we can $T$-minimize $(TR, \hat{i}, \hat{\delta})$ (Construction 4.4), which yields an equivalent $T$-automaton. Notice that, $R$ being finite, any quotient of $TR$ has a finite set of generators. This is a consequence of $R$ being a set of generators for $TR$ (Lemma 3.3) and of generators being preserved by quotients (Proposition 4.3). It follows that every step of the $T$-minimization construction terminates.

**Proposition 5.2.** The $T$-succinct automaton defined above is at least as small as the minimal deterministic automaton equivalent to $X$.

**Proof.** The situation is summed up in the following commutative diagram:

\[
\begin{array}{ccc}
G & \xrightarrow{g} & R \\
\downarrow{\eta} & & \downarrow{\eta^R} \\
TR & \xleftarrow{\epsilon} & M \\
\end{array}
\]

\[
\begin{array}{ccc}
M & \xrightarrow{m^*} & O^* \\
\end{array}
\]
Here $G$ is the final minimal set of generators for $M$ resulting from the construction. Commutativity follows from $G$ being a subset of the set of generators $R$.

The minimal deterministic automaton equivalent to $X$ is obtained from $R$ by merging language-equivalent states. Recalling (4) and the proof of Proposition 5.1, we see that $e \circ \eta_R$ is a coalgebra homomorphism. Together with commutativity of the above diagram, this means that the language accepted by $r \in G$ (seen as a state of $R$) is given by $(m \circ g)(r)$. Since $G$ is a subset of $R$, to show that $G$ is at least as small as the minimal deterministic automaton, we only have to show that different states in $G$ accept different languages. That is, we will show that $m \circ g$ is injective. We know that $m$ is injective by definition; to see that $g$ is injective, consider $r_1, r_2 \in G$ such that $g(r_1) = g(r_2)$. Then $g(r_1) = g(r_2) = g^0(\eta(r_2))$. Assuming $r_1 \neq r_2$ leads to the contradiction that $G$ is not a minimal set of generators because in this case $\eta(r_2) \in T(G \setminus \{r_1\})$.

Computing the determinization $T_R$ is an expensive operation that only terminates if $T$ preserves finite sets. One could devise an optimized version of Construction 4.4 in which the determinization is not computed completely in order to minimize it. Instead, we could choose to work with data structures as Böling et al. [7] did for non-deterministic automata, and which we generalized in recent work [23]. In these papers, partial representations of the determinized automaton are used in an iterative process to compute the generators of the state space of the minimal one.

6. Examples

6.1. Monads preserving finite sets

If $T$ preserves finite sets, then there is a naive method to find a redundant element: assuming a finite set of generators $G$ for a $T$-algebra $X$, the set $T(G \setminus \{r\})$ is also finite for any $r \in G$. Thus, we can loop over all $U \in T(G \setminus \{r\})$ and check if the generator map $g: G \to X$ satisfies $g^0(U) = g(r)$. 

6.1.1. Alternating automata.

We now use our construction to get small alternating finite automata (AFAs) over a finite alphabet $A$. AFAs generalize both non-deterministic and universal automata, where the latter are the dual of non-deterministic automata: a word is accepted when all paths reading it are accepting. In an AFA, reading a symbol leads to a DNF formula (without negation) of next states.

We use the characterization of alternating automata due to Bertrand [5]. Given a partially ordered set $(P, \leq)$, an upset is a subset $U$ of $P$ such that whenever $x \in U$ and $x \leq y$, then $y \in U$. Given $Q \subseteq P$, we write $\uparrow Q$ for the upward closure of $Q$, that is the smallest upset of $P$ containing $Q$. We consider the monad $T_{\text{Alt}}$ that maps a set $X$ to the set of all upsets of $P(X)$. Its unit is given by $\eta_X(x) = \uparrow \{\{x\}\}$ and its multiplication by

$$\mu_X(U) = \{V \subseteq X \mid \exists W \in U \forall Y \in W \exists Z \in Y \ Z \subseteq V\}.$$ 

The sets of sets in $T_{\text{Alt}}(X)$ can be seen as DNF formulae over elements of $X$: the outer powerset is interpreted disjunctively and the inner one conjunctively. Accordingly, we define an algebra structure $\beta: T_{\text{Alt}}(2) \to 2$ on the output set
Figure 1: Automata for the language \{a, ba, bb, baa\}

2 by letting \(\beta(U) = 1\) if \(\{1\} \in U\) and \(\beta(U) = 0\) otherwise. Recall from (3) in Section 2 that such an algebra structure induces a distributive law.

We now explicitly spell out the \(T\)-minization algorithm that turns a DFA \((X, i, \delta)\) into a \(T\)\(_{\text{Alt}}\)-succinct AFA.

1. Compute the reachable states \(R\) of \((X, i, \delta)\) via a standard visit of its graph.
2. Compute the corresponding freely-generated \(T\)\(_{\text{Alt}}\)-automaton \((T\)\(_{\text{Alt}}\)R, \(\hat{i}, \hat{\delta}\)) by generating all DNF formulae \(T\)\(_{\text{Alt}}\)R on \(R\).
3. Compute the observable quotient \((M, i_0, \delta_0)\) of \((T\)\(_{\text{Alt}}\)R, \(\hat{i}, \hat{\delta}\)) via a standard minimization algorithm, such as the coalgebraic Paige–Tarjan algorithm [11].
4. Compute a minimal set of generators for \(M\) as follows. Consider the generator map \(\text{id}_M : M \to M\), for which we have that \(\text{id}^\delta\) is the algebra map of \(M\). Pick \(r \in M\), and iterate over all DNF formulae \(\varphi\) over \(M\) \(\setminus \{r\}\); if there is \(\varphi\) which is mapped to \(r\) by the algebra map of \(M\) (i.e., \(\text{id}^\delta\)), \(r\) is redundant and can be removed from \(M\). Repeat until no more elements are removed from \(M\), which yields a minimal set of generators \(G\).
5. Return the \(T\)\(_{\text{Alt}}\)-succinct automaton \((G, i_0, i_0 \circ \eta)\).

Note that every step of this algorithm terminates, as \(X\) is finite and the size of \(T\)\(_{\text{Alt}}\)R is \(2^{2^n}\).

Example 6.1. Consider the regular language over \(A = \{a, b\}\) given by the finite set \(\{a, ba, bb, baa\}\). The minimal DFA accepting this language is given in Figure 1a.

According to our construction, we first construct a \(T\)\(_{\text{Alt}}\)-automaton with state space freely generated from this automaton (which is already reachable). Then we \(T\)\(_{\text{Alt}}\)-minimize it in order to obtain a small AFA. In this case, there is a unique minimal subset of 3 generators: \(G = \{q_0, q_1, q_2\}\). To see this, consider the languages \([q]\) accepted by states \(q\) of the deterministic automaton:

\[
\begin{align*}
[q_0] &= \{a, ba, bb, baa\} \\
[q_1] &= \{a, b, aa\} \\
[q_2] &= \{a, b, bb\} \\
[q_3] &= \{\varepsilon, a\}. \\
[q_4] &= \emptyset
\end{align*}
\]

These languages generate the states of the minimal \(T\)\(_{\text{Alt}}\)-automaton by interpreting joins as unions and meets as intersections. We note that \([q_4]\) is just an empty
join and $[[q_3]] = ([[q_0]] \cap [[q_1]]) \cup [[q_2]]$. These are the only redundant generators. Removing them leads to the AFA in Figure 1b. Here the black square represents a conjunction of next states.

6.1.2. Complete Atomic Boolean Algebras

We now consider the monad $C$ given by the double contravariant powerset functor, namely $CX = 2^{2^X}$. Here the outer powerset is treated disjunctively as in the case of $T_{\text{Alt}}$, and the sets provided by the inner powerset are interpreted as valuations. Thus, elements of $C(X)$ can be seen as full DNF formulae over $X$: every conjunctive clause contains for each $x \in X$ either $x$ or the negation $\overline{x}$ of $x$. The unit assigns to an element $x$ the disjunction of all full conjunctions containing $x$, and the multiplication turns formulae into full DNF formulae in the usual way. Algebras for this monad are known as complete atomic boolean algebras (CABAs).

Using the fact that $2$ is a free CABA ($2 \cong C(\emptyset)$), we obtain the following semantics for $C$-succinct automata: a set of sets of states is accepting if and only if it contains the exact set $F$ of accepting states. This is different from alternating automata, where a subset of $F$ is sufficient. Reading a symbol in a $C$-succinct automaton works as follows. Suppose we are in a set of sets of states $S \in C(Q)$, where we read a symbol $a$. The resulting set of sets contains $U \subseteq Q$ if and only if there is a set $V \in S$ such that every state in $V$ transitions into a set of sets containing $U$, and every state not in $V$ does not transition into any set of sets containing $U$.

Note that every DNF formula can be converted to a full DNF formula. This implies that $C$-succinct automata can always be as small as the smallest AFAs for a given language. With the following example we show that they can actually be strictly smaller. The $T$-minimization algorithm for AFA we have given in the previous section applies to this setting as well (including negation in DNF formulae).

Example 6.2. Consider the regular language of words over the singleton alphabet $A = \{a\}$ whose length is non-zero and even. The minimal DFA accepting this language is shown in Figure 2a. We start the algorithm with the $C$-automaton

14

---

---

14
with state space freely generated from this this DFA and merge the language-equivalent states. Initially, the set of generators is the set of states of the original DFA. By noting that the language accepted by \( q_2 \) is the negation of the one accepted by \( q_1 \), in full DNF form \( \mathcal{L}_{q_2} = (\mathcal{L}_{q_0} \cap \mathcal{L}_{q_1^c}) \cup (\mathcal{L}_{q_0^c} \cap \mathcal{L}_{q_1}) \) (where for any language \( U \) its complement is defined as \( U^c = A^* \setminus U \)), we see that \( q_2 \) is redundant. The set of generators \( \{q_0, q_1\} \) is minimal and corresponds to the \( C \)-succinct automaton in Figure 2b. We depict \( C \)-succinct automata in the same manner as AFAs, but note that their interpretation is different. Here the transition into the black square represents the transition into the conjunction of the negations of \( q_0 \) and \( q_1 \).

We now show that there is no AFA with two states accepting the same language. Suppose such an AFA exists, and let the state space be \( \mathcal{X} = \{x_0, x_1\} \). Since \( a \) and \( aaa \) are not in the language but \( aa \) is, one of these states must be accepting and the other must be rejecting.\(^4\) Without loss of generality we assume that \( x_0 \) is rejecting and \( x_1 \) is accepting. The empty word is not in the language, so our initial configuration has to be \( \uparrow\{\{x_0\}\} \). Since \( a \) is also not in the language, \( x_0 \) will have to transition to \( \uparrow\{\{x_0\}\} \) as well. However, this implies that \( aa \) is not accepted by the AFA, which contradicts the assumption that it accepts the right language.

Unfortunately, the fact that the transition behavior of a set of states depends on states not in that set generally makes it difficult to work with \( C \)-succinct automata by hand.

### 6.1.3. Symmetry

We now consider succinct automata that exploit symmetry present in their accepted language. Given a finite group \( G \), consider the monad \( G \times (-) \), where the unit pairs any element with the unit of \( G \) and the multiplication applies the multiplication of \( G \). The algebras for \( G \times (-) \) are precisely left group actions. We assume an action on the alphabet \( A \); if no such action is relevant, one may consider the trivial action \( G \times A \twoheadrightarrow A \). We also assume an action on the output set \( O \). Group actions will be denoted by a centered dot. We consider the distributive law \( \rho: G \times (O \times (-))^A \Rightarrow O \times (G \times (-))^A \) given by

\[
\rho_X(g, o, f) = (g \cdot o, \lambda a. (g, f(g^{-1} \cdot a))).
\]

We explain the resulting semantics of \((G \times (-))-\text{succinct automata} in an example.

**Example 6.3.** Consider the group \( \text{Perm}(\{a, b\}) = \{e, (ab)\} \) of permutations over elements \( a \) and \( b \). Here \( e \) is the identity and \( (ab) \) swaps \( a \) and \( b \). We consider the alphabet \( A = \{a, b\} \) with an action \( \text{Perm}(A) \times A \to A \) given by applying the permutation to the element of \( A \), and the output set \( O = A \cup \{\bot\} \) with an action given by

\[
(ab) \cdot a = b \quad (ab) \cdot b = a \quad (ab) \cdot \bot = \bot.
\]

\(^4\)If there were no rejecting states, the only way to reject a word is by ending up in the empty set of sets of states. However, this means that extensions of that word are rejected as well. Similarly, if there are no accepting states one can only accept by ending up in \( \uparrow\{\emptyset\} \), which accepts everything.
Figure 3: Automata outputting the first symbol to appear twice in a row

(a) Deterministic automaton

(q₀, ⊥) → (q₁, ⊥) → (q₃, a) → (a, b)

(b) Corresponding \((G \times (-))\)-succinct automaton

(q₀, ⊥) → (q₁, ⊥) → (a/e, b/e)

\[(ab) \cdot q₀ = q₀ \quad (ab) \cdot q₁ = q₂ \quad (ab) \cdot q₂ = q₃ \quad (ab) \cdot q₃ = q₄ \quad (ab) \cdot q₄ = q₃.\]

We note that in the set of generators given by the full state space, q₁, q₂, q₃, and q₄ are redundant. After removing q₂, only q₃ and q₄ are redundant. Subsequently removing q₄ leaves no redundant elements.

The final \((G \times (-))\)-succinct automaton is shown in Figure 3b. Its actual configurations are pairs of a group element and a state. Transition labels are of the form \(x/g\), where \(x \in A\) and \(g \in \text{Perm}(A)\). If we are in a configuration \((g, q)\) and state \(q\) has an associated output \(o \in O\), the actual output is \(g \cdot o\). On reading a symbol \(x \in A\), we find the outgoing transition of which the label starts with the symbol \(g^{-1} \cdot x\). Supposing this label contains a group element \(g'\) and leads to a state \(q'\), the resulting configuration is \((gg', q')\). For example, consider reading the word \(bb\). We start in the configuration \((e, q₀)\). Reading \(b\) here simply takes the transition corresponding to \(b\), which brings us to \((ab), q₁\). Now reading the second \(b\), we actually read \((ab)^{-1} \cdot b = (ab) \cdot b = a\). This brings us to \((ab), q₃\). The output is then given by \((ab) \cdot a = b\).

In general, sets of generators in this setting correspond to subsets in which all orbits are represented. The orbits of a set \(X\) with a left group action are the equivalence classes of the relation that identifies elements \(x, y \in X\) whenever there exists \(g \in G\) such that \(g \cdot x = y\). Minimal sets of generators contain a single representative for each orbit. The algorithm given for AFAs in section 6.1.1 can be applied to this setting as well: step 4 will remove elements until only orbit representatives are left.

6.2. Vector Spaces

We now exploit vector space structures. Given a field \(F\), consider the free vector space monad \(V\). It maps each set \(X\) to the set of functions \(X \rightarrow F\) with
finite support (finitely many elements of $X$ are mapped to a non-zero value). A function $f: X \to Y$ is mapped to the function $V(f): V(X) \to V(Y)$ given by

$$V(f)(g)(y) = \sum_{x \in X, f(x) = y} g(x).$$

The unit $\eta: X \to V(X)$ and multiplication $\mu: VV(X) \to V(X)$ of the monad are given by

$$\eta(x)(x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{if } x \neq x' \end{cases} \quad \mu(f)(x) = \sum_{g \in V(X)} f(g) \cdot g(x) \in \mathbb{F}.$$

Here 0 and 1, as well as addition and multiplication, are those of the field $\mathbb{F}$. Elements of $V(X)$ can alternatively be written as formal sums $v_1 x_1 + \cdots + v_n x_n$ with $v_i \in \mathbb{F}$ and $x_i \in X$ for all $i$. We will use this notation in the example below.

Algebras for the free vector space monad are precisely vector spaces. We use the output set $O = \mathbb{F}$, and the alphabet can be any finite set $A$. Instantiating (3), this leads to a pointwise distributive law $\rho: V(O \times (-)^A) \Rightarrow O \times V(-)^A$ given at a set $X$ by

$$\rho(f) = \left( \sum_{(o,g) \in O \times X^A} f(o,g) \cdot o, \lambda a. \lambda x. \sum_{(o,g) \in O \times X^A, g(x) = x} f(o,g) \right).$$

With these definitions, the $V$-succinct automata are weighted automata. We note that if $\mathbb{F}$ is infinite, any non-trivial $V$-automaton will also be infinite. However, we can still start from a given weighted automaton and apply a slight modification of Construction 4.4: minimize from the succinct representation, use the states of the succinct representation as initial set of generators, and finally find a minimal set of generators. Moreover, we may add a reachability analysis, which in this case cannot lead to a larger automaton. Thus, the resulting algorithm essentially comes down to the standard minimization algorithm for weighted automata [19], where the process of removing redundant generators is integrated into the minimization. If $\mathbb{F}$ is finite and we do want to start from a deterministic automaton, we can consider this automaton as a weighted one by assigning each transition a weight of 1.

**Example 6.4.** Consider for $\mathbb{F} = \mathbb{R}$ the deterministic automaton in Figure 4a. This is a minimal automaton in Set; the freely generated $V$-automaton is infinite, and so is its minimization. However, that minimization has the states of the automaton in Figure 4a as a set of generators. To gain insight into this minimization, we compute the languages accepted by those generators (apart from $q_0$):

$q_1$: $\varepsilon \mapsto 1$ \hspace{1cm} $a \mapsto 1$ \hspace{1cm} $b \mapsto 1$ \hspace{1cm} $c \mapsto 1$
$q_2$: $\varepsilon \mapsto 1$ \hspace{1cm} $a \mapsto 0$ \hspace{1cm} $b \mapsto 0$ \hspace{1cm} $c \mapsto 0$
$q_3$: $\varepsilon \mapsto 3$ \hspace{1cm} $a \mapsto 1$ \hspace{1cm} $b \mapsto 1$ \hspace{1cm} $c \mapsto 1$
$q_4$: $\varepsilon \mapsto 0$ \hspace{1cm} $a \mapsto 0$ \hspace{1cm} $b \mapsto 0$ \hspace{1cm} $c \mapsto 0$

Words not displayed are mapped to 0 by any state. The language of $q_0$ is the only one assigning non-zero values to certain words of length two, such as $aa$, 

17
and therefore $q_0$ cannot be a redundant generator. The other generators are redundant: writing $[q]$ for the language of a state $q$, $[q_4]$ is just a zero-ary sum, and we have

$$[q_1] = [q_3] - 2[q_2] \quad [q_2] = \frac{1}{2}[q_4] - \frac{1}{2}[q_1] \quad [q_3] = [q_1] + 2[q_2].$$

Once $q_4$ is removed, all other generators are still redundant. Further removing $q_3$ makes $q_1$ and $q_2$ isolated. Therefore, $V$-minimization yields the weighted automaton shown in Figure 4b. Here a transition on an input $x \in A$ with weight $w \in F$ receives the label $x/w$, or just $x$ if $w = 1$. Weights multiply along a path, and different possible paths add up to assign a value to a word. Reading $c$ from $q_0$, for example, we move to $q_1 + 2q_2$, which has an output of $1 + 2 \times 1 = 3$.

In general, the (sub)sets of generators of a vector space are its subsets that span the whole space, and such a set of generators is minimal precisely when it forms a basis. The weighted automaton resulting from our algorithm is the usual minimal weighted automaton for the language. Redundant elements can be found using standard techniques such as Gaussian elimination.

7. Conclusions

We have presented a construction to obtain succinct representations of deterministic finite automata as automata with side-effects. This construction is very general in that it is based on the abstract characterisation of side-effects as monads. Nonetheless, it can be easily implemented. An essential part of our construction is the computation of a minimal set of generators for an algebra. We have provided an algorithm for this that works for any suitable Set monad. We have applied the construction to several non-trivial examples: alternating automata, automata with symmetries, CABA-structured automata, and weighted automata.

Related work. This work revamps and extends results of Arbib and Manes [4], as discussed throughout the paper. We note that most of their results are formulated in a more general category, whereas here we work specifically in Set. The reason for this is that we focus on the procedure for finding minimal sets of generators.
by removing redundant elements, which are defined using set subtraction (Definition 3.6). This limitation is already present in the work of Arbib and Manes, who spend little time on the subject and only study the non-deterministic case in detail. Our main contribution, the general procedure for finding a minimal set of generators, is not present in their work. It generalizes several techniques to obtain compact automaton representations of languages, some of them presented in the context of learning algorithms [10, 7, 3]. Preliminary results on generalizing succinct automaton constructions within a learning algorithm can be found in [23].

In [17], Myers et al. present a coalgebraic construction of canonical non-deterministic automata. Their specific examples are the atomaton [9], obtained from the atoms of the boolean algebra generated by the residual languages (the languages accepted by the states of the minimal DFA); the canonical RFSA; the minimal xor automaton [24], actually a weighted automaton over field with two elements rather than a non-deterministic one; and what they call the distromaton, obtained from the atoms of the distributive lattice generated by the residual languages. They further provide specific algorithms for obtaining some of their example succinct automata.

The underlying idea in the work of Myers et al. for finding succinct representations of algebras is similar to ours, and the deterministic structured automata they start from are equivalent: in their paper the deterministic automata live in a locally finite variety, which translates to the category of algebras for a monad that preserves finite sets (such as those in Section 6.1). They also define the succinct automaton using a minimal set of generators for the algebra, but instead of our algorithmic approach of getting to this set by removing redundant generators, they use a dual equivalence between finite algebras and a suitable modification of the category of sets and relations between them. This seems to restrict their work to non-deterministic automata, although there may be an easy generalization: the equivalence would be with a modification of a Kleisli category. A major difference with our work is that they have no general algorithm to construct the succinct automata; as mentioned, specific ones are provided for their examples. In fact, they provide no guidelines on how to find a suitable equivalence for a given variety. On the other hand, their equivalences guarantee uniqueness up to isomorphism of the succinct automata, which is a desirable property for many applications.

The restriction in the work of Myers et al. to locally finite varieties means that our example of weighted automata over an infinite field (Section 6.2) cannot be captured in their work. Conversely, since both the atomaton and the distromaton are non-deterministic NFAs obtained from categories of algebras with more structure than JSLs, these examples are not covered by our work. Their other examples, however, the canonical RFSA and the minimal xor automaton, are obtained using instances of our method as well. The fact that the problem of finding in general a suitable equivalence is open means it is not trivial to determine whether our approach can be seen as a special case of a generalized version of theirs when we restrict to monads that preserve finite sets.

Future work. The main question that remains is under which conditions the notion of a minimal set of generators actually describes a size-minimal set of generators. Proposition 3.7 provides a partial answer to this question, but its conditions fail to apply the majority of our examples, even though in some of
these cases minimal does mean size-minimal. A related question is whether we can find heuristics to increase the state space of a $T$-automaton in such a way that the number of generators decreases. The reason the canonical RFSAs of Denis et al. [10] are not always state-minimal NFAs is because the states of these NFAs, seen as singletons in the determinized automaton, in general are not reachable. Hence, removing unreachable states from a $T$-automaton may increase the size of minimal sets of generators, which is why Construction 4.4 does not include a reachability analysis. Although finding state-minimal NFAs is PSPACE-complete, a moderate gain might still be possible.

References


