Automata Minimization: a Functorial Approach[*][†]

Thomas Colcombet[1] and Daniela Petrişan[2]

1 **CNRS, IRIF, Univ. Paris-Diderot, Paris 7, France**
thomas.colcombet@irif.fr

2 **CNRS, IRIF, Univ. Paris-Diderot, Paris 7, France**
petrisan@irif.fr

Abstract

In this paper we regard languages and their acceptors – such as deterministic or weighted automata, transducers, or monoids – as functors from input categories that specify the type of the languages and of the machines to categories that specify the type of outputs.

Our results are as follows: a) We provide sufficient conditions on the output category so that minimization of the corresponding automata is guaranteed. b) We show how to lift adjunctions between the categories for output values to adjunctions between categories of automata. c) We show how this framework can be applied to several phenomena in automata theory, starting with determinization and minimization (previously studied from a coalgebraic and duality theoretic perspective). We apply in particular these techniques to Choffrut's minimization algorithm for subsequential transducers and revisit Brzozowski's minimization algorithm.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases functor automata, minimization, Choffrut's minimization algorithm, subsequential transducers, Brzozowski's minimization algorithm

Digital Object Identifier 10.4230/LIPIcs.CALCO.2017.8

1 Introduction

There is a long tradition of interpreting results of automata theory using the lens of category theory. Typical instances of this scheme interpret automata as algebras (together with a final map) as put forward in [3, 14, 1], or as coalgebras (together with an initial map), see for example [16]. This dual narrative proved very useful [7] in explaining at an abstract level Brzozowski's minimization algorithm and the duality between reachability and observability (which goes back all the way to the work of Arbib, Manes and Kalman).

In this paper, we adopt a slightly different approach, and we define directly the notion of an automaton (over finite words) as a functor from a category representing input words, to a category representing the computation and output spaces. The notions of a language and of a language accepted by an automaton are adapted along the same pattern.

We provide several developments around this idea. First, we recall (see [12]) that the existence of a minimal automaton for a language is guaranteed by the existence of an initial and a final automaton in combination with a factorization system. Additionally, we explain how, in the functor presentation that we have adopted, the existence of initial and final

[*] This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and by the DeLTA ANR project (ANR-16-CE40-0007). The authors also thank the Simons Institute for the Theory of Computing where this work has been partly developed.

[†] A knowledge enriched version of the paper is available at https://arxiv.org/abs/1711.03063.

7th Conference on Algebra and Coalgebra in Computer Science (CALCO 2017).
Editors: Filippo Bonchi and Barbara König; Article No. 8; pp. 8:1–8:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automata for a language can be phrased in terms of Kan extensions. We also show how adjunctions between categories can be lifted to the level of automata for a language in these categories (Lemma 8). This lifting accounts for several constructions in automata theory, determinization to start with. We then use this framework in the explanation of two well-known constructions in automata theory.

The most involved contribution (Theorem 12) is to rephrase the minimization result of Choffrut for subsequential transducers in this framework. We do this by instantiating the category of outputs with the Kleisli category for the monad $TX = B^* \times X + 1$, where B is the output alphabet of the transducers. In this case, despite the lack of completeness of the ambient category, one can still prove the existence of an initial and final automaton, as well as, surprisingly, of a factorization system.

The second concrete application is a proof of correctness of Brzozowski's minimization algorithm. Indeed, determinization of automata can be understood as lifting the Kleisli adjunction between the categories Rel (of sets and relations) and Set (of sets and functions); and reversing a nondeterministic automaton can be understood as lifting the self-duality of Rel. In Section 5 we show how Brzozowski's minimization algorithm can be explained by combining several liftings of adjunctions, and in particular as lifting the adjunction between Set and its opposite category Set^{op}, thus recovering results from [7].

Related work. Many of the constructions outlined here have already been explained from a category-theoretic perspective, using various techniques. For example, the relationship between minimization and duality was subject to numerous papers, see [6, 7, 8] and the references therein. The coalgebraic perspective on minimization was also emphasised in papers such as [2, 19]. Understanding determinization and codeterminization, as well as studying trace semantics via lifting of adjunctions to coalgebras was considered in [17, 18], and is related to our results from Section 5.2. Subsequential transducers were understood coalgebraically in [15].

The paper which is closest in spirit to our work is a seemingly forgotten paper [4]. However, in this work, Bainbridge models the *state space* of the machines as a functor. Left and right Kan extensions are featured in connection with the initial and final automata, but in a slightly different setting. Lemma 8, which albeit technically simple, has surprisingly many applications, builds directly on his work.

2 Languages and Automata as Functors

In this section, we introduce the notion of automata via functors, which is the common denominator of the different contributions of the paper. We introduce this definition starting from the special case of classical deterministic automata.

In the standard definition, a deterministic automaton is a tuple:

$$\langle Q, A, q_0, F, \delta\rangle$$

where Q is a set of states, A is an alphabet (not necessarily finite), $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta_a \colon Q \to Q$ is the transition map for all letters $a \in A$. The semantics of an automaton is given by defining what is a run over an input word $u \in A^*$, and whether it is accepting or not. Given a word $e = a_1 \dots a_n$, the automaton accepts the word if $\delta_{a_n} \circ \dots \circ \delta_{a_1}(q_0) \in F$, and otherwise rejects it.

If we see q_0 as a map *init* from the one element set $1 = \{0\}$ to Q, that maps 0 to q_0, and F as a map *final* from Q to the set $2 = \{0, 1\}$, where 1 means 'accept' and 0 means 'reject', then the semantics of the automaton is to associate to each word $u = a_1 \dots a_n$ the map from 1 to 2 defined as $\mathit{final} \circ \delta_{a_n} \circ \dots \circ \delta_{a_1} \circ \mathit{init}$. If this map is (constant equal to) 1, this means that the word is accepted, and otherwise it is rejected.

Pushing this idea further, we can see the semantics of the automaton as a functor from the free category generated by the graph on the right to Set, and more precisely one that sends the object in to 1 and out to 2. In the above category, the arrows from in to out are of the form $\triangleright w \triangleleft$ for w an arbitrary word in A^*.

$$\begin{tikzcd}
\mathsf{in} \arrow[r, "\triangleright"] & \mathsf{states} \arrow[loop above, "a"] \arrow[r, "\triangleleft"] & \mathsf{out}
\end{tikzcd}$$

Furthermore, since a language can be seen as a map from A^* to $1 \to 2$, we can model it as a functor from the full subcategory on objects in and out to the category Set, which maps in to 1 and out to 2.

In this section we fix an arbitrary small category $\mathcal{I}$ and a full subcategory $\mathcal{O}$. We denote by ι the inclusion functor

$$\mathcal{O} \xhookrightarrow{\iota} \mathcal{I}\,.$$

We think of $\mathcal{I}$ as a specification of the inner computations that an automaton can perform, including black box behaviour, not observable from the outside. On the other hand, the full subcategory $\mathcal{O}$ specifies the observable behaviour of the automaton, that is, the language it accepts. In this interpretation, a machine/automaton $\mathcal{A}$ is a functor from $\mathcal{I}$ to a category of outputs $\mathcal{C}$, and the "behaviour" or "language" of $\mathcal{A}$ is the functor $\mathcal{L}(\mathcal{A})$ obtained by precomposition with the inclusion $\mathcal{O} \xhookrightarrow{\iota} \mathcal{I}$. We obtain the following definition:

▶ **Definition 1** (languages and the categories of automata for them)**.**
A $\mathcal{C}$-language is a functor $\mathcal{L} \colon \mathcal{O} \to \mathcal{C}$ and a $\mathcal{C}$-automaton is a functor $\mathcal{A} \colon \mathcal{I} \to \mathcal{C}$. A $\mathcal{C}$-automaton $\mathcal{A}$ accepts a $\mathcal{C}$-language $\mathcal{L}$ when $\mathcal{A} \circ \iota = \mathcal{L}$; i.e. the diagram below commutes:

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook, "\iota"'] & \mathcal{C} \\
\mathcal{I} \arrow[ur, "\mathcal{A}"'] &
\end{tikzcd}$$

We write $\mathsf{Auto}(\mathcal{L})$ for the subcategory of the functor category $[\mathcal{I}, \mathcal{C}]$ where

1. objects are $\mathcal{C}$-automata that accept $\mathcal{L}$.
2. arrows are natural transformations $\alpha \colon \mathcal{A} \to \mathcal{B}$ so that the natural transformation obtained by composition with the inclusion functor ι is the identity natural transfomation on $\mathcal{L}$, that is, $\alpha \circ \iota = id_{\mathcal{L}}$.

2.1 Minimization of $\mathcal{C}$-automata

In this section we show that the notion of a minimal automaton is an instance of a more generic notion of minimal object that can be defined in an arbitrary category $\mathcal{K}$ whenever there exist an initial object, a final object, and a factorization system $(\mathcal{E}, \mathcal{M})$.

Let X, Y be two objects of $\mathcal{K}$. We say that:

$$X \quad (\mathcal{E},\mathcal{M})\text{-divides} \quad Y \qquad \text{if} \qquad X \text{ is an } \mathcal{E}\text{-quotient of an } \mathcal{M}\text{-subobject of } Y.$$

Let us note immediately that in general this notion of ($\mathcal{E}$,$\mathcal{M}$)-divisibility may not be transitive[1]. It is now natural to define an object M to be ($\mathcal{E}$,$\mathcal{M}$)-minimal in the category, if it ($\mathcal{E}$,$\mathcal{M}$)-divides all objects of the category. Note that there is no reason a priori that an ($\mathcal{E}$,$\mathcal{M}$)-minimal object in a category, if it exists, be unique up to isomorphism. Nevertheless, in our case, when the category has both initial and a final object, we can state the following minimization lemma:

▶ **Lemma 2.** *Let $\mathcal{K}$ be a category with initial object I and final object F and let $(\mathcal{E}, \mathcal{M})$ be a factorization system for $\mathcal{K}$. Define for every object X:*

- $\texttt{Min}$ *to be the factorization of the only arrow from I to F,*
- $\texttt{Reach}(X)$ *to be the factorization of the only arrow from I to X, and* $\texttt{Obs}(X)$ *to be the factorization of the only arrow from X to F.*

Then

- $\texttt{Min}$ *is ($\mathcal{E}$,$\mathcal{M}$)-minimal, and*
- $\texttt{Min}$ *is isomorphic to both* $\texttt{Obs}(\texttt{Reach}(X))$ *and* $\texttt{Reach}(\texttt{Obs}(X))$ *for all objects X.*

Proof. The proof essentially consists of a diagram:

$$\begin{tikzcd}
 & & X \arrow[dr, bend left] & \\
I \arrow[urr, bend left] \arrow[r, two heads] \arrow[drr, bend right, two heads] & \texttt{Reach}(X) \arrow[ur, tail] \arrow[r, two heads] & \texttt{Obs}(\texttt{Reach}(X)) \arrow[r, tail] \arrow[d, dashed, no head] & F \\
 & & \texttt{Min} \arrow[ur, tail, bend right] &
\end{tikzcd}$$

Using the definition of $\texttt{Reach}$ and $\texttt{Obs}$, and the fact that $\mathcal{E}$ is closed under composition, we obtain that $\texttt{Obs}(\texttt{Reach}(X))$ is an ($\mathcal{E}$,$\mathcal{M}$)-factorization of the only arrow from I to F. Thus, thanks to the diagonal property of a factorization system, $\texttt{Min}$ and $\texttt{Obs}(\texttt{Reach}(X))$ are isomorphic. Hence, furthermore, since $\texttt{Obs}(\texttt{Reach}(X))$ ($\mathcal{E}$,$\mathcal{M}$)-divides X by construction, the same holds for $\texttt{Min}$. In a symmetric way, $\texttt{Reach}(\texttt{Obs}(X))$ is also isomorphic to $\texttt{Min}$. ◀

An object X of $\mathcal{K}$ is called reachable when X is isomorphic to $\texttt{Reach}(X)$. We denote by $\mathsf{Reach}(\mathcal{K})$ the full subcategory of $\mathcal{K}$ consisting of reachable objects. Similarly, an object X of $\mathcal{K}$ is called observable when X is isomorphic to $\texttt{Obs}(X)$. We denote by $\mathsf{Obs}(\mathcal{K})$ the full subcategory of $\mathcal{K}$ consisting of observable objects.

We can express reachability $\texttt{Reach}$ and observability $\texttt{Obs}$ as the right, respectively the left adjoint to the inclusion of $\mathsf{Reach}(\mathcal{K})$, respectively of $\mathsf{Obs}(\mathcal{K})$ into $\mathcal{K}$. It is indeed a standard fact that factorization systems give rise to reflective subcategories, see [9]. In our case, this is the reflective subcategory $\mathsf{Obs}(\mathcal{K})$ of $\mathcal{K}$. By a dual argument, the category $\mathsf{Reach}(\mathcal{K})$ is coreflective in $\mathcal{K}$. We can summarize these facts in the next lemma.

▶ **Lemma 3.** *Let $\mathcal{K}$ be a category with initial object I and final object F and let $(\mathcal{E}, \mathcal{M})$ be a factorization system for $\mathcal{K}$. We have the adjunctions*

$$\begin{tikzcd}
\mathsf{Reach}(\mathcal{K}) \arrow[r, hook, bend left] \arrow[r, phantom, "\perp"] & \mathcal{K} \arrow[l, bend left, "\texttt{Reach}"] \arrow[r, bend left, "\texttt{Obs}"] \arrow[r, phantom, "\perp"] & \mathsf{Obs}(\mathcal{K}) \arrow[l, hook, bend left]
\end{tikzcd}.$$

[1] There are nevertheless many situations for which it is the case; In particular when the category is regular, and $\mathcal{E}$ happens to be the class of regular epis. This covers in particular the case of all algebraic categories with $\mathcal{E}$-quotients being the standard quotients of algebras, and $\mathcal{M}$-subobjects being the standard subalgebras.

In what follows we will instantiate $\mathcal{K}$ with the category $\mathsf{Auto}(\mathcal{L})$ of $\mathcal{C}$-automata accepting a language $\mathcal{L}$. Assuming the existence of an initial and final automaton for $\mathcal{L}$ – denoted by $\mathcal{A}^{init}(\mathcal{L})$, respectively $\mathcal{A}^{final}(\mathcal{L})$ – and, of a factorization system, we obtain the functorial version of the usual notions of reachable sub-automaton $\mathtt{Reach}(\mathcal{A})$ and observable quotient automaton $\mathtt{Obs}(\mathcal{A})$ of an automaton $\mathcal{A}$. The minimal automaton $\mathtt{Min}(\mathcal{L})$ for the language $\mathcal{L}$ is obtained via the factorization

$$\begin{tikzcd} \mathcal{A}^{init}(\mathcal{L}) \arrow[r, two heads] & \mathtt{Min}(\mathcal{L}) \arrow[r, tail] & \mathcal{A}^{final}(\mathcal{L}) \end{tikzcd}.$$

Lemma 2 implies that the minimal automaton divides any other automaton recognising the language, while Lemma 3 instantiates to the results of [7, Section 9.4].

2.2 Minimization of $\mathcal{C}$-automata: sufficient conditions on $\mathcal{C}$

We now can list sufficient conditions on $\mathcal{C}$ so that the category of $\mathcal{C}$-automata $\mathsf{Auto}(\mathcal{L})$ accepting a $\mathcal{C}$-language $\mathcal{L}$ satisfies the three conditions of Lemma 2.

We start with the factorization system. It is well known that given a factorization system $(\mathcal{E},\mathcal{M})$ on $\mathcal{C}$, we can extend it to a factorization system $(\mathcal{E}_{[\mathcal{I},\mathcal{C}]},\mathcal{M}_{[\mathcal{I},\mathcal{C}]})$ on the functor category $[\mathcal{I},\mathcal{C}]$ in a pointwise fashion. That is a natural transformation is in $\mathcal{E}_{[\mathcal{I},\mathcal{C}]}$ if all its components are in $\mathcal{E}$, and analogously, a natural transformation is in $\mathcal{M}_{[\mathcal{I},\mathcal{C}]}$ if all its components are in $\mathcal{M}$. In turn, the factorization system on the functor category $[\mathcal{I},\mathcal{C}]$ induces a factorization system on each subcategory $\mathsf{Auto}(\mathcal{L})$.

▶ **Lemma 4.** *If $\mathcal{C}$ has a factorization system $(\mathcal{E},\mathcal{M})$, then $\mathsf{Auto}(\mathcal{L})$ has a factorization system $(\mathcal{E}_{\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})},\mathcal{M}_{\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})})$, where $\mathcal{E}_{\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})}$ consists of all the natural transformations with components in $\mathcal{E}$ and $\mathcal{M}_{\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})}$ consists of all natural transformations with components in $\mathcal{M}$.*

The proof of Lemma 4 is the same as the classical one that shows that factorization systems can be lifted to functor categories.

▶ **Lemma 5.** *If the left Kan extension $\mathsf{Lan}_\iota\mathcal{L}$ of $\mathcal{L}$ along ι exists, then it is an initial object in $\mathsf{Auto}(\mathcal{L})$, that is, $\mathcal{A}^{init}(\mathcal{L})$ exists and is isomorphic to $\mathsf{Lan}_\iota\mathcal{L}$.*

Dually, if the right Kan extension $\mathsf{Ran}_\iota\mathcal{L}$ of $\mathcal{L}$ along ι exists, then so does the final object $\mathcal{A}^{final}(\mathcal{L})$ of $\mathsf{Auto}(\mathcal{L})$ and $\mathcal{A}^{final}(\mathcal{L})$ is isomorphic to $\mathsf{Ran}_\iota\mathcal{L}$.

Proof Sketch. Assume the left Kan extension exists. Then the canonical natural transformation $\mathcal{L}\to\mathsf{Lan}_\iota\mathcal{L}\circ\iota$ is an isomorphism since ι is full and faithful. Whenever $\mathcal{A}$ accepts $\mathcal{L}$, that is, $\mathcal{A}\circ\iota=\mathcal{L}$, we obtain the required unique morphism $\mathsf{Lan}_\iota\mathcal{L}\to\mathcal{A}$ using the universal property of the Kan extension. The argument for the right Kan extension follows by duality. ◀

▶ **Corollary 6.** *Assume $\mathcal{C}$ is complete, cocomplete and has a factorization system and let $\mathcal{L}$ be a $\mathcal{C}$-language. Then the initial $\mathcal{L}$-automaton and the final $\mathcal{L}$-automaton exist and are given by the left, respectively right Kan extensions of $\mathcal{L}$ along ι. Furthermore, the minimal $\mathcal{C}$-automaton $\mathtt{Min}(\mathcal{L})$ accepting $\mathcal{L}$ is obtained via the factorization* $\begin{tikzcd} \mathsf{Lan}_\iota\mathcal{L} \arrow[r, two heads] & \mathtt{Min}(\mathcal{L}) \arrow[r, tail] & \mathsf{Ran}_\iota\mathcal{L} \end{tikzcd}$.

▶ Remark. Depending on the category $\mathcal{I}$, we may relax the conditions in Corollary 6, see Lemma 7. Furthermore, we emphasise that these conditions are only sufficient. In Section 4 we consider the example of subsequential transducers and we instantiate $\mathcal{C}$ with a Kleisli category. Although this category does not have powers, the final automaton exists.

3 Word Automata

Hereafter, we restrict our attention to the case of word automata, for which the input category $\mathcal{I}$ is the three-object category with arrows spanned by $\triangleright$, $\triangleleft$ and a for all $a \in A$, as in the diagram below and where the composite of $\mathsf{states} \xrightarrow{w} \mathsf{states} \xrightarrow{w'} \mathsf{states}$ is given by the concatenation ww'.

$$\begin{tikzcd}
\mathsf{in} \arrow[r, "\triangleright"] & \mathsf{states} \arrow[r, "\triangleleft"] \arrow[loop above, "a"] & \mathsf{out}
\end{tikzcd}$$

Let $\mathcal{O}$ be the full subcategory of $\mathcal{I}$ on objects in and out. We consider $\mathcal{C}$-languages, which are now functors $\mathcal{L}\colon \mathcal{O} \to \mathcal{C}$. If $\mathcal{L}(\mathsf{in}) = X$ and $\mathcal{L}(\mathsf{out}) = Y$ we call $\mathcal{L}$ a $(\mathcal{C}, X, Y)$-language. Similarily, we consider $\mathcal{C}$-automata that are functors $\mathcal{A}\colon \mathcal{I} \to \mathcal{C}$. If $\mathcal{A}(\mathsf{in}) = X$ and $\mathcal{A}(\mathsf{out}) = Y$ we call $\mathcal{A}$ a $(\mathcal{C}, X, Y)$-automaton.

The next lemma refines Corollary 6. It appeared in [5] in the $(\mathsf{Set}, 1, 2)$-language case.

▶ **Lemma 7** (from [12]). *If $\mathcal{C}$ has countable products and countable coproducts, and a factorization system, then the minimal $\mathcal{C}$-automaton accepting $\mathcal{L}$ is obtained via the factorization in the next diagram.*

The initial automaton has as state space the copower $\coprod_{u\in A^*} \mathcal{L}(\mathsf{in})$. In [12] we gave a direct proof of initiality, but here we can also notice that this is exactly what the colimit computation of the left Kan extension of $\mathcal{L}$ along ι yields – using the fact that there are no morphisms from out to states in $\mathcal{I}$ and the only morphism on which you take the colimit are of the form $\triangleright w\colon \mathsf{in} \to \mathsf{states}$ for all $w \in A^*$.

$$\begin{tikzcd}
& \coprod\limits_{u\in A^*} \mathcal{L}(\mathsf{in}) \arrow[d, two heads] \arrow[dr, bend left, "L?"] & \\
\mathcal{L}(\mathsf{in}) \arrow[ur, bend left, "\varepsilon"] \arrow[r, "i"] \arrow[dr, bend right, "L"'] & \mathtt{Min}(\mathcal{L}) \arrow[r, "f"] \arrow[d, tail] & \mathcal{L}(\mathsf{out}) \\
& \prod\limits_{u\in A^*} \mathcal{L}(\mathsf{out}) \arrow[ur, bend right, "\varepsilon?"'] &
\end{tikzcd}$$

3.1 Lifting Adjunctions to Categories of Automata

In this section we will juggle with languages and automata interpreted over different categories connected via adjunctions.

Assume we have an adjunction between two categories $\mathcal{C}$ and $\mathcal{D}$

$$\begin{tikzcd}
\mathcal{C} \arrow[r, shift left=1.5, "F"] \arrow[r, phantom, "\perp"] & \mathcal{D} \arrow[l, shift left=1.5, "G"]
\end{tikzcd},$$

with $F \dashv G\colon \mathcal{D} \to \mathcal{C}$. Let $(-)^*$ and $(-)_*$ denote the induced natural isomorphisms between the homsets. In particular, given objects I in $\mathcal{C}$ and O in $\mathcal{D}$, we have bijections

$$\begin{tikzcd}
\mathcal{C}(I, GO) \arrow[r, shift left=1.5, "(-)^*"] & \mathcal{D}(FI, O) \arrow[l, shift left=1.5, "(-)_*"]
\end{tikzcd} \tag{1}$$

These bijections induce a one-to-one correspondence between $(\mathcal{C}, I, GO)$-languages and $(\mathcal{D}, FI, O)$-languages, which by an abuse of notation we denote by the same symbols:

$$\begin{tikzcd}
(\mathcal{C}, I, GO)\text{-languages} \arrow[r, shift left, "(-)^*"] & (\mathcal{D}, FI, O)\text{-languages} \arrow[l, shift left, "(-)_*"]
\end{tikzcd}$$

Indeed, given a $(\mathcal{C}, I, GO)$-language $\mathcal{L} \colon \mathcal{O} \to \mathcal{C}$ we obtain a $(\mathcal{D}, FI, O)$-language $\mathcal{L}^* \colon \mathcal{O} \to \mathcal{D}$ by setting $\mathcal{L}^*(\triangleright w \triangleleft) = (\mathcal{L}(\triangleright w \triangleleft))^* \in \mathcal{D}(FI, O)$. Conversely, given a $(\mathcal{D}, FI, O)$-language $\mathcal{L}'$ we obtain a $(\mathcal{C}, I, GO)$-language $(\mathcal{L}')_*$ by setting $(\mathcal{L}')_*(\triangleright w \triangleleft) = (\mathcal{L}'(\triangleright w \triangleleft))_*$.

▶ **Lemma 8.** *Assume $\mathcal{L}_\mathcal{C}$ and $\mathcal{L}_\mathcal{D}$ are $(\mathcal{C}, I, GO)$-, respectively $(\mathcal{D}, FI, O)$-languages so that $\mathcal{L}_\mathcal{D} = (\mathcal{L}_\mathcal{C})^*$. Then the adjunction $F \dashv G$ lifts to an adjunction $\overline{F} \dashv \overline{G} \colon \mathsf{Auto}(\mathcal{L}_\mathcal{D}) \to \mathsf{Auto}(\mathcal{L}_\mathcal{C})$. The lifted functors $\overline{F}$ and $\overline{G}$ are defined as F, resp. G on the state object, that is, the following diagram commutes*

$$\begin{tikzcd}
\mathsf{Auto}(\mathcal{L}_\mathcal{C}) \arrow[r, bend left, "\overline{F}"] \arrow[r, phantom, "\perp"] \arrow[d, "\mathsf{State}"] & \mathsf{Auto}(\mathcal{L}_\mathcal{D}) \arrow[l, bend left, "\overline{G}"] \arrow[d, "\mathsf{State}"] \\
\mathcal{C} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{D} \arrow[l, bend left, "G"]
\end{tikzcd}$$

where the functor $\mathsf{State} \colon \mathsf{Auto}(\mathcal{L}_\mathcal{C}) \to \mathcal{C}$ is the evaluation at states*, that is, it sends an automaton $\mathcal{A} \colon \mathcal{I} \to \mathcal{C}$ to $\mathcal{A}(\mathsf{states})$.*

Proof sketch. The functor $\overline{F}$ maps an automaton $\mathcal{A} \colon \mathcal{I} \to \mathcal{C}$ from $\mathsf{Auto}(\mathcal{L}_\mathcal{C})$ to the $\mathcal{D}$-automaton $\overline{F}\mathcal{A} \colon \mathcal{I} \to \mathcal{D}$ mapping $\triangleright \colon \mathsf{in} \to \mathsf{states}$ to $F\mathcal{A}(\triangleright)$, $a \colon \mathsf{states} \to \mathsf{states}$ to $F(\mathcal{A}(a))$ and $\triangleleft \colon \mathsf{states} \to \mathsf{out}$ to the adjoint transpose $(\mathcal{A}(\triangleleft))^*$ of $\mathcal{A}(\triangleleft)$. The functor $\overline{G}$ is defined similarly. ◀

4 Choffrut's minimization of subsequential transducers

In [10, 11] Choffrut establishes a minimality result for subsequential transducers, which are deterministic automata that output a word while processing their input. In this section, we show the existence of minimal subsequential transducers using our functorial framework.

We first present the model of subsequential transducers in Section 4.1, show how these can be identified with automata in the Kleisli category of a suitably chosen monad, and state the minimization result, Theorem 12. The subsequent sections provide the necessary material for proving the theorem.

4.1 Subsequential transducers and automata in a Kleisli category

Subsequential transducers are (finite state) machines that compute partial functions from input words in some alphabet A to output words in some other alphabet B. In this section, we recall the classical definition of these objects, and show how it can be phrased categorically.

▶ **Definition 9.** A subsequential transducer is a tuple

$$T = (Q, A, B, q_0, t, u_0, (- \cdot a)_{a \in A}, (- * a)_{a \in A}) \ ,$$

where

- A is the input alphabet and B the output one,
- Q is a (finite) set of states.
- q_0 is either undefined or belongs to Q and is called the initial state of the transducer.
- $t \colon Q \rightharpoonup B^*$ is a partial termination function.
- $u_0 \in B^*$ is defined if and only if q_0 is, and is the initialization value.
- $- \cdot a \colon Q \rightharpoonup Q$ is the partial transition function for the letter a, for all $a \in A$.
- $- * a \colon Q \rightharpoonup B^*$ is the partial production function for the letter a for all $a \in A$; it is required that $q * a$ be defined if and only if $(q \cdot a)$ is.

A subsequential transducer T computes a partial function $[\![T]\!] \colon A^* \rightharpoonup B^*$ defined as:

$$[\![T]\!](a_1 \dots a_n) = u_0(q_0 * a_1)(q_1 * a_2) \dots (q_{n-1} * a_n)t(q_n) \qquad \text{for all } a_1 \dots a_n \in A^*,$$

where for each $1 \leq i \leq n$ either q_i is undefined or belongs to Q and is given by $q_i = q_{i-1} \cdot a_i$. Furthermore, $[\![T]\!](a_1 \dots a_n)$ is undefined when at least one of $q_0, \dots, q_n$ or $t(q_n)$ is so.

These subsequential transducers are modeled in our framework as automata in the category of free algebras for the monad $\mathcal{T}$, that we describe now.

▶ **Definition 10.** The monad $\mathcal{T} \colon \mathsf{Set} \to \mathsf{Set}$ is defined by

$$\mathcal{T}(X) = B^* \times X + 1$$

with unit η_X and multiplication μ_X defined for all $x \in X$ and $w, u \in B^*$ as:

$$\begin{array}{rrcl} \eta_X \colon & X & \to & B^* \times X + 1 \\ & x & \mapsto & (\varepsilon, x) \end{array} \qquad \begin{array}{rrcl} \mu_X \colon & \mathcal{T}^2 X & \to & \mathcal{T} X \\ & (w, (u, x)) & \mapsto & (wu, x) \\ & (w, \bot) & \mapsto & \bot \\ & \bot & \mapsto & \bot \end{array}$$

where we denote by $\bot$ the unique element of 1 (used to model the partiality of functions).

Recall that the category of free $\mathcal{T}$-algebras is the Kleisli category for $\mathcal{T}$, $\mathsf{Kl}(\mathcal{T})$, that has as objects sets $X, Y, \dots$ and as morphisms $f \colon X \nrightarrow Y$ functions $f \colon X \to B^* \times Y + 1$ in Set, that is a partial function from X to $B^* \times Y$.

Let T be a subsequential transducer. The initial state of the transducer q_0 and the initialization value u_0 together form a morphism $i \colon 1 \nrightarrow Q$ in the category $\mathsf{Kl}(\mathcal{T})$. Similarly, the partial transition function and the partial production function for a letter a of the input alphabet A are naturally identified to Kleisli morphisms $\delta_a \colon Q \nrightarrow Q$ in $\mathsf{Kl}(\mathcal{T})$. Finally, the partial termination function together with the partial production function are nothing but a Kleisli morphism of the form $t \colon Q \nrightarrow 1$. To summarise, we obtained that a subsequential transducer T in the sense of [11] is specified by the following morphisms in $\mathsf{Kl}(\mathcal{T})$:

$$\begin{tikzcd} 1 \arrow[r, "i", "\shortmid" marking] & Q \arrow[loop above, "\delta_a", "\shortmid" marking] \arrow[r, "t", "\shortmid" marking] & 1 \end{tikzcd}$$

that is by a functor $\mathcal{A}_T \colon \mathcal{I} \to \mathsf{Kl}(\mathcal{T})$ or equivalently, a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automaton. The subsequential function realised by the transducer T is a partial function $A^* \rightharpoonup B^*$ and is fully captured by the $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-language $\mathcal{L}_T \colon \mathcal{O} \to \mathsf{Kl}(\mathcal{T})$ accepted by $\mathcal{A}_T$, which is obtained as $\mathcal{A}_T \circ \iota$. Indeed, this $\mathsf{Kl}(\mathcal{T})$-language gives for each word $w \in A^*$ a Kleisli morphism

$\mathcal{L}_T(\triangleright w \triangleleft)\colon 1 \nrightarrow 1$, or equivalently, outputs for each word in A^* either a word in B^* or the undefined element $\perp$.

Putting all this together, we can state the following lemma, which validates the categorical encoding of subsequential transducers:

▶ **Lemma 11.** *Subsequential transducers are in one to one correspondence with* $(\mathsf{Kl}(\mathcal{T}), 1, 1)$*-automata, and partial maps from* A^* *to* B^* *are in one to one correspondence with* $(\mathsf{Kl}(\mathcal{T}), 1, 1)$*-languages. Furthermore, the acceptance of languages is preserved under these bijections.*

In the rest of this section we will see how to obtain Choffrut's minimization result as an application of Lemma 2. I.e., we have to provide in the category of $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata,

1. an initial object,
2. a final object, and,
3. a factorization system.

The existence of the initial transducer is addressed in Section 4.3, the one of the final transducer is the subject of Section 4.4. In Section 4.5 we show how to construct a factorization system@factorization transducer. Together, we obtain:

▶ **Theorem 12** (Categorical version of [10, 11])**.** *For every* $(\mathsf{Kl}(\mathcal{T}), 1, 1)$*-language, there exists a minimal* $(\mathsf{Kl}(\mathcal{T}), 1, 1)$*-automaton for it.*

Let us note that only the existence of the automaton is mentioned in this statement, and the way to compute it effectively is not addressed as opposed to Choffrut's work. Nevertheless, Lemma 2 describes what are the basic functions that have to be implemented, namely `Reach` and `Obs`.

The rest of this section is devoted to establishing the three above mentioned points. Unfortunately, as it is usually the case with Kleisli categories, $\mathsf{Kl}(\mathcal{T})$ is neither complete, nor cocomplete. It does not even have binary products, let alone countable powers. Also, the existence of a factorization system does not generally hold in Kleisli categories. Hence, providing the above three pieces of information requires a bit of work.

In the next section we present an adjunction between categories of $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata and $(\mathsf{Set}, 1, B^*)$-automata which is then used in the subsequent ones for proving the existence of initial and final automata. We finish the proof with a presentation of the factorization systems.

4.2 Back and forth to automata in Set

In order to understand what are the properties of the category of $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata, an important tool will be the ability to see alternatively a subsequential transducer as an automaton in $\mathsf{Kl}(\mathcal{T})$ as described above, or as an automaton in Set, since Set is much better behaved than $\mathsf{Kl}(\mathcal{T})$. These two points of view are related through an adjunction, making use of the results of Section 3.1 and Lemma 4.

Indeed, we start from the well known adjunction between Set and $\mathsf{Kl}(\mathcal{T})$:

$$\begin{tikzcd}
\mathsf{Set} \arrow[rr, bend left, "F_{\mathcal{T}}"] \arrow[rr, phantom, "\perp"] & & \mathsf{Kl}(\mathcal{T}) \arrow[ll, bend left, "U_{\mathcal{T}}"] .
\end{tikzcd} \tag{2}$$

We recall that the free functor $F_{\mathcal{T}}$ is defined as the identity on objects, while for any function $f\colon X \to Y$ the morphism $F_{\mathcal{T}} f\colon X \nrightarrow Y$ is defined as $\eta_Y \circ f\colon X \to \mathcal{T}Y$. For the other direction, the functor $U_{\mathcal{T}}$ maps an object X in $\mathsf{Kl}(\mathcal{T})$ to $\mathcal{T}X$ and a morphism $f\colon X \nrightarrow Y$ (which is seen here as a function $f\colon X \to \mathcal{T}Y$) to $\mu_Y \circ \mathcal{T}f\colon \mathcal{T}X \to \mathcal{T}Y$.

A simple, yet important observation is that the language of interest, which is a partial function $L\colon A^* \rightharpoonup B^*$ can be modeled either as a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-language $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$, or, as a $(\mathsf{Set}, 1, B^* + 1)$-language $\mathcal{L}_{\mathsf{Set}}$. This is because for each $w \in A^*$ we can identify $L(w)$ either with an element of $\mathsf{Kl}(\mathcal{T})(1, 1)$ or, equivalently, as an element of $\mathsf{Set}(1, B^* + 1)$.

$$\begin{array}{rrcl}
\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}\colon & \mathcal{O} & \to & \mathsf{Kl}(\mathcal{T}) \\
& \mathsf{in} & \mapsto & 1 \\
& \mathsf{out} & \mapsto & 1 \\
& \triangleright w \triangleleft & \mapsto & L(w)\colon 1 \nrightarrow 1
\end{array}
\qquad\qquad
\begin{array}{rrcl}
\mathcal{L}_{\mathsf{Set}}\colon & \mathcal{O} & \to & \mathsf{Set} \\
& \mathsf{in} & \mapsto & 1 \\
& \mathsf{out} & \mapsto & B^* + 1 \\
& \triangleright w \triangleleft & \mapsto & L(w)\colon 1 \to B^* + 1
\end{array}$$

To see how this fits in the scope of Section 3.1, notice that $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$ is a $(\mathsf{Kl}(\mathcal{T}), F_{\mathcal{T}}1, 1)$-language, while $\mathcal{L}_{\mathsf{Set}}$ is a $(\mathsf{Set}, 1, U_{\mathcal{T}}1)$-language and they correspond to each other via the bijections described in (1).

Applying Lemma 8 for the Kleisli adjunction (2) we obtain an adjunction $\overline{F_{\mathcal{T}}} \dashv \overline{U_{\mathcal{T}}}$ between the categories of $\mathsf{Kl}(\mathcal{T})$-automata for $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$ and of Set-automata accepting $\mathcal{L}_{\mathsf{Set}}$, as depicted in the picture below. The functor $\overline{U_{\mathcal{T}}}$ sends a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automaton with state object Q to a $(\mathsf{Set}, 1, B^* + 1)$-automaton with state object $\mathcal{T}Q$, while $\overline{F_{\mathcal{T}}}$ sends a $(\mathsf{Set}, 1, B^* + 1)$-automaton with state object Q' to a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automaton with same state object Q'.

We will make heavy use of this correspondence in what follows.

$$\begin{tikzcd}
\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[r, bend left, "\overline{F_{\mathcal{T}}}"] \arrow[r, phantom, "\perp"] \arrow[d, "\mathsf{State}"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}) \arrow[l, bend left, "\overline{U_{\mathcal{T}}}"] \arrow[d, "\mathsf{State}"] \\
\mathsf{Set} \arrow[r, bend left, "F_{\mathcal{T}}"] \arrow[r, phantom, "\perp"] & \mathsf{Kl}(\mathcal{T})\,. \arrow[l, bend left, "U_{\mathcal{T}}"]
\end{tikzcd}$$

4.3 The initial $\mathsf{Kl}(\mathcal{T})$-automaton for the language $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$

The functor $\overline{F_{\mathcal{T}}}$ is a left adjoint and consequently preserves colimits and in particular the initial object. We thus obtain that the initial $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$-automaton is $\overline{F_{\mathcal{T}}}(\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}}))$, where $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})$ is the initial object of $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}})$. This automaton can be obtained by Lemma 7 as the functor $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})\colon \mathcal{I} \to \mathsf{Set}$ specified by $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})(\mathsf{states}) = A^*$ and for all $a \in A$

$$\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})(\triangleright)\colon 1 \to A^* \\
0 \mapsto \varepsilon
\end{array}
\qquad
\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})(\triangleleft)\colon A^* \to B^* + 1 \\
w \mapsto L(w)
\end{array}
\qquad
\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})(a)\colon A^* \to A^* \\
w \mapsto wa
\end{array}$$

Hence, by computing the image of $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Set}})$ under $\overline{F_{\mathcal{T}}}$, we obtain the following description of the initial $\mathsf{Kl}(\mathcal{T})$-automaton $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ accepting $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$: $\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\mathsf{states}) = A^*$ and for all $a \in A$

$$\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleright)\colon 1 \nrightarrow A^* \\
0 \mapsto (\varepsilon, \varepsilon)
\end{array}
\qquad
\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleleft)\colon A^* \nrightarrow 1 \\
w \mapsto L(w)
\end{array}
\qquad
\begin{array}{rl}
\mathcal{A}^{init}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(a)\colon A^* \nrightarrow A^* \\
w \mapsto (\varepsilon, wa)
\end{array}$$

4.4 The final $\mathsf{Kl}(\mathcal{T})$-automaton for the language $\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}$

The case of the final $\mathsf{Kl}(\mathcal{T})$-automaton is more complicated, since it is not constructed as easily. However, assuming the final automaton exists, it has to be sent by $\overline{U_\mathcal{T}}$ to a final Set-automaton. Moreover, by Lemma 13, in order to prove that a given $\mathsf{Kl}(\mathcal{T})$-automaton $\mathcal{A}$ is a final object of $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ it suffices to show that $\overline{U_\mathcal{T}}(\mathcal{A})$ is the final object in $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}})$. The proof of the following lemma generalises the fact that $U_\mathcal{T}$ reflects final objects and can be proved in the same spirit.

▶ **Lemma 13.** *The functor $\overline{U_\mathcal{T}}\colon \mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}) \to \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}})$ reflects final objects.*

The final object in $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}})$ is the automaton $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})$ as described using Lemma 7. The functor $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})\colon \mathcal{I} \to \mathsf{Set}$ specified by

$$\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})(\mathsf{states}) = (B^*+1)^{A^*} \qquad \begin{aligned}\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})(\triangleleft)\colon (B^*+1)^{A^*} &\to B^*+1\\ K &\mapsto K(\varepsilon)\end{aligned}$$

$$\begin{aligned}\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})(\triangleright)\colon 1 &\to (B^*+1)^{A^*}\\ 0 &\mapsto L\end{aligned} \qquad \begin{aligned}\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})(a)\colon (B^*+1)^{A^*} &\to (B^*+1)^{A^*}\\ K &\mapsto \lambda w.K(aw)\end{aligned}$$

To describe the set of states of the final automaton in $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ we need to introduce a few notations. Essentially we are looking for a set of states Q so that $B^* \times Q + 1$ is isomorphic to $(B^*+1)^{A^*}$. The intuitive idea is to decompose each function in $K \in (B^*+1)^{A^*}$ (except for the one which is nowhere defined, that is the function $\kappa_\perp = \lambda w.\perp$) into a word in B^*, the common prefix of all the B^*-words in the image of K, and an irreducible function.

For $v \in B^*$ and a function $K \neq \kappa_\perp$ in $(B^*+1)^{A^*}$, denote by $v \star K$ the function defined for all $u \in A^*$ by $(v \star K)(u) = v\,K(u)$ if $K(u) \in B^*$ and $(v \star K)(u) = \perp$ otherwise. Define also the longest common prefix of K, $\mathsf{lcp}(K) \in B^*$, as the longest word that is prefix of all $K(u) \neq \perp$ for u in A^* (this is well defined since $K \neq \kappa_\perp$). The reduction of K, $\mathsf{red}(K)$, is defined as:

$$\mathsf{red}(K)(u) = \begin{cases} v & \text{if } K(u) = \mathsf{lcp}(K)\,v,\\ \perp & \text{otherwise.}\end{cases}$$

Finally, K is called irreducible if $\mathsf{lcp}(K) = \varepsilon$ (or equivalently if $K = \mathsf{red}(K)$). We denote by $\mathsf{Irr}(A^*, B^*)$ the irreducible functions in $(B^*+1)^{A^*}$.

What we have constructed is a bijection between

$$\mathcal{T}(\mathsf{Irr}(A^*,B^*)) = B^* \times \mathsf{Irr}(A^*,B^*) + 1 \qquad \text{and} \qquad (B^*+1)^{A^*}\ ,$$

that is defined as

$$\begin{aligned}\varphi\colon B^* \times \mathsf{Irr}(A^*,B^*) + 1 &\to (B^*+1)^{A^*}\\ (u,K) &\mapsto u \star K\\ \perp &\mapsto \kappa_\perp\ ,\end{aligned} \tag{3}$$

and the converse of which maps every $K \neq \kappa_\perp$ to $(\mathsf{lcp}(K), \mathsf{red}(K))$, and $\kappa_\perp$ to $\perp$.

Given $a \in A$ and $K \in (B^*+1)^{A^*}$ we denote by $a^{-1}K$ the function in $(B^*+1)^{A^*}$ that maps $w \in A^*$ to $K(aw)$.

We can now define a functor $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})\colon \mathcal{I} \to \mathsf{Kl}(\mathcal{T})$ by setting

$$\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\mathsf{in}) = 1 \quad \mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\mathsf{states}) = \mathsf{Irr}(A^*,B^*) \quad \mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\mathsf{out}) = 1$$

and defining $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ on arrows as follows

$$\begin{array}{lll}
\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleright)\colon 1 \nrightarrow \mathsf{Irr}(A^*,B^*) & \quad 0 \mapsto (\mathsf{lcp}(L),\mathsf{red}(L)) & \\
\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleleft)\colon \mathsf{Irr}(A^*,B^*) \nrightarrow 1 & K \mapsto K(\varepsilon) & \\
\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(a)\colon \mathsf{Irr}(A^*,B^*) \nrightarrow \mathsf{Irr}(A^*,B^*) & K \mapsto \kappa_\bot & \text{if } a^{-1}K = \kappa_\bot \\
 & K \mapsto (\mathsf{lcp}(a^{-1}K),\mathsf{red}(a^{-1}K)) & \text{otherwise.}
\end{array}$$

▶ **Lemma 14.** *The* $\mathsf{Kl}(\mathcal{T})$*-automaton* $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ *is a final object in* $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$.

Proof. We show that $\overline{U_\mathcal{T}}(\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))$ is isomorphic to the final automaton $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})$. Indeed, at the level of the state objects the bijection between $\overline{U_\mathcal{T}}(\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))(\mathsf{states})$ and $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})(\mathsf{states})$ is given by the function φ defined in (3). One can check that on arrows $\overline{U_\mathcal{T}}(\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))$ is the same as $\mathcal{A}^{\mathit{final}}(\mathcal{L}_{\mathsf{Set}})$ up to the correspondence given by φ. ◀

4.5 A factorization system on $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$

The factorization system on $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ is obtained using Lemma 4 from a factorization system on $\mathsf{Kl}(\mathcal{T})$. There are several non-trivial factorization systems on $\mathsf{Kl}(\mathcal{T})$, one of which is obtained from the regular epi-mono factorization system on Set, or equivalently, from the regular epi-mono factorization system on the category of Eilenberg-Moore algebras for $\mathcal{T}$. Notice that this is a specific result for the monad $\mathcal{T}$ since in general, there is no reason that the Eilenberg-Moore algebra obtained by factorizing a morphism between free algebras be free itself. Nevertheless, in order to capture precisely the syntactic transducer defined by Choffrut [10, 11], we will provide yet another factorization system $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})},\mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$, which we define concretely as follows. Given a morphism $f\colon X \nrightarrow Y$ in $\mathsf{Kl}(\mathcal{T})$ we write $\pi_1(f)\colon X \to B^* + \{\bot\}$ and $\pi_2(f)\colon X \to Y + \{\bot\}$ for the 'projections' of f, defined by

$$\pi_1(f)(x) = \begin{cases} u & \text{if } f(x) = (u,y)\,, \\ \bot & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \pi_2(f)(x) = \begin{cases} y & \text{if } f(x) = (u,y)\,, \\ \bot & \text{otherwise.} \end{cases}$$

We say that a partial function $g\colon X \to Y + \{\bot\}$ is surjective when for every $y \in Y$ there exists $x \in X$ so that $g(x) = y$.

The class $\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ consists of all the morphisms of the form $e\colon X \nrightarrow Y$ such that $\pi_2(e)$ is surjective and the class $\mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ consists of all the morphisms of the form $m\colon X \nrightarrow Y$ such that $\pi_2(m)$ is injective and $\pi_1(m)$ is the constant function mapping every $x \in X$ to ε.

▶ **Lemma 15.** $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})},\mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ *is a factorization system on* $\mathsf{Kl}(\mathcal{T})$.

Proof. Notice that f is an isomorphism in $\mathsf{Kl}(\mathcal{T})$ if and only if $f \in \mathcal{E}_{\mathsf{Kl}(\mathcal{T})} \cap \mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$.

If $f\colon X \nrightarrow Y$ is a morphism in $\mathsf{Kl}(\mathcal{T})$ then we can define

$$Z = \{y \in Y \mid \exists x \in X.\, \exists u \in B^*.\, f(x) = (u,y)\}\,.$$

We define $e\colon X \nrightarrow Z$ by $e(x) = f(x)$ and $m\colon Z \nrightarrow Y$ by $m(y) = (\epsilon, y)$. One can easily check that $f = m \circ e$ in $\mathsf{Kl}(\mathcal{T})$.

Lastly, we can show that the "diagonal property" holds. Assume we have a commuting square in $\mathsf{Kl}(\mathcal{T})$.

$$\begin{tikzcd}
X \arrow[r, two heads, "e"] \arrow[d, "f"'] & Y \arrow[d, "g"] \arrow[dl, dashed, "d"] \\
Z \arrow[r, tail, "m"'] & W
\end{tikzcd}$$

We will prove the existence of $d\colon Y \twoheadrightarrow Z$ so that $d \circ e = f$ and $m \circ d = g$. Assume $y \in Y$. If $g(y) = \bot$ we set $d(y) = \bot$. Otherwise assume $g(y) = (v, t)$, for some $v \in B^*$ and $t \in W$. Since $e \in \mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$, there exists $u \in B^*$ and $x \in X$ so that $e(x) = (u, y)$. Assume $f(x) = (w, z)$ for some $w \in B^*$ and $z \in Z$. We set $d(y) = (v, z)$. First, we have to prove that this definition does not depend on the choice of x.

Assume that we have another $x' \in X$ so that $g(x') = (u', y)$ and assume $f(x') = (w', z')$. Using the fact that $m \in \mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$, we will show that $z = z'$, and thus $d(y)$ is well defined. Indeed, notice that

$$\begin{cases} g \circ e(x) = (uv, t) \\ g \circ e(x') = (u'v, t)\,, \end{cases} \qquad \text{or equivalently,} \qquad \begin{cases} m \circ f(x) = (uv, t) \\ m \circ f(x') = (u'v, t)\,, \end{cases}$$

Assume that $m(z) = (\varepsilon, t_1)$ and $m(z') = (\varepsilon, t_2)$. This entails

$$\begin{cases} m \circ f(x) = (uv, t) = (w, t_1) \\ m \circ f(x') = (u'v, t) = (w', t_2)\,. \end{cases}$$

We obtain that $t_1 = t_2 = t$. Since $m \in \mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ (and thus $\pi_2(m)$ is injective) we get that $z = z'$, which is what we wanted to prove. It is easy to verify that $d \circ e = f$ and $m \circ d = g$. ◀

This completes the proof of Theorem 12.

5 Brzozowski's minimization algorithm

5.1 Presentation

Brzozowski's algorithm is a minimization algorithm for automata. It takes as input a non-deterministic automaton $\mathcal{A}$, and computes the deterministic automaton:

$\texttt{determinize}(\texttt{transpose}(\texttt{determinize}(\texttt{transpose}(\mathcal{A})))),$

in which

- `determinize` is the operation from classical automata theory that takes as input a deterministic automaton, performs the powerset construction to it and at the same time restricts to the reachable states, yielding a deterministic automaton, and
- `transpose` is the operation that takes as input a non-deterministic automaton, reverses all its edges, and swaps the role of initial and final states (it accepts the mirrored language).

In this section, we will establish the correctness of Brzozowski's algorithm: this sequence of operations yields the minimal automaton for the language. For easing the presentation we shall present the algorithm in the form:

$\texttt{determinize}(\texttt{codeterminize}(\mathcal{A})),$

in which `codeterminize` is the operation that takes a non-deterministic automaton, and constructs a backward deterministic one (it is equivalent to the sequence $\texttt{transpose} \circ \texttt{determinize} \circ \texttt{transpose}$).

In the next section, we show how `determinize` and `codeterminize` can be seen as adjunctions, and we use it immediately after in a correctness proof of Brzozowski's algorithm.

5.2 Non-deterministic automata and determinization

A non-deterministic automaton is completely determined by the relations described in the following diagram, where we see the initial states as a relation from 1 to the set of states Q, the final states as a relation from Q to 1 and the transition relation by any input letter a, as a relation on Q.

$$\begin{tikzcd}
1 \arrow[r, "i", "\shortmid" marking] & Q \arrow[r, "f", "\shortmid" marking] \arrow[loop above, "\delta_a", "\shortmid" marking] & 1
\end{tikzcd}$$

We can model nondeterministic automata as functors by taking as output category Rel – the category whose objects are sets and maps are relations between them. We consider Rel-automata $\mathcal{A}\colon \mathcal{I} \to \mathsf{Rel}$ such that $\mathcal{A}(\mathsf{in}) = 1$ and $\mathcal{A}(\mathsf{out}) = 1$. In this section we show how to determinize a Rel-automaton, that is, how to turn it into a Set-automaton and how to codeterminize it, that is, how to obtain a Set^{op}-automaton, all recognizing the same language.

Given a language $L \subseteq A^*$ we can model it in several equivalent ways: as a $(\mathsf{Set}, 1, 2)$-language $\mathcal{L}_{\mathsf{Set}}$, or as a $(\mathsf{Set}^{op}, 2, 1)$-language $\mathcal{L}_{\mathsf{Set}^{op}}$, or, lastly as a $(\mathsf{Rel}, 1, 1)$-language $\mathcal{L}_{\mathsf{Rel}}$. This is because we can model the fact $w \in L$ using a morphisms in either of the three isomorphic hom-sets

$$\mathsf{Set}(1, 2) \cong \mathsf{Set}^{op}(2, 1) \cong \mathsf{Rel}(1, 1)\,.$$

Determinization and codeterminization (without assuming the restriction to reachable states as in `determinize` and `codeterminize`) of a Rel-automaton can be seen as applications of Lemma 8 and are obtained by lifting the adjunctions between Set, Rel and Set^{op}.

$$\begin{tikzcd}
\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[r, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] \arrow[d] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Rel}}) \arrow[l, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[r, bend left, "\overline{U^{op}_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] \arrow[d] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, bend left, "\overline{F^{op}_{\mathcal{P}}}"] \arrow[d] \\
\mathsf{Set} \arrow[r, bend left, "F_{\mathcal{P}}"] \arrow[r, phantom, "\perp"] & \mathsf{Rel} \arrow[l, bend left, "U_{\mathcal{P}}"] \arrow[r, bend left, "U^{op}_{\mathcal{P}}"] \arrow[r, phantom, "\perp"] & \mathsf{Set}^{op} \arrow[l, bend left, "F^{op}_{\mathcal{P}}"]
\end{tikzcd} \tag{4}$$

The adjunction between Set and Rel is the Kleisli adjunction for the powerset monad: $F_{\mathcal{P}}$ is identity on objects and maps a function $f\colon X \to Y$ to itself $f\colon X \nrightarrow Y$, but seen as a relation. The functor $U_{\mathcal{P}}$ maps X to its powerset $\mathcal{P}(X)$, and a relation $R\colon X \to Y$ to the function $U_{\mathcal{P}}(R)\colon \mathcal{P}(X) \to \mathcal{P}(Y)$ mapping $A \subseteq X$ to $\{y \in Y \mid \exists x \in A.(x, y) \in R\}$.

The adjunction between Set^{op} and Rel is the dual of the previous one, composed with the self-duality of Rel. The left adjoint $\overline{F_{\mathcal{P}}}$ transforms a deterministic automaton into a non-deterministic one, while the right adjoint $\overline{U_{\mathcal{P}}}$ is the determinization functor. On the other hand, the left adjoint functor $\overline{U^{op}_{\mathcal{P}}}$ is the codeterminization functor.

5.3 Brzozowski's minimization algorithm

The correctness of Brzozowski's algorithm can be seen in the following chain of adjunctions from Lemma 3 and (4) (that all correspond to equivalences at the level of languages):

$$\begin{tikzcd}
\mathsf{Reach}(\mathcal{L}_{\mathsf{Set}}) \arrow[r, hook, bend left, "E"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[l, bend left, "\mathtt{Reach}"] \arrow[r, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Rel}}) \arrow[l, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[r, bend left, "\overline{U^{op}_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, bend left, "\overline{F^{op}_{\mathcal{P}}}"] \arrow[r, bend left, "\mathtt{Obs}"] \arrow[r, phantom, "\perp"] & \mathsf{Obs}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, hook, bend left, "E"]
\end{tikzcd}$$

A path in this diagram corresponds to a sequence of transformations of automata. It happens that when $\mathtt{Obs}$ is taken, the resulting automaton is observable, i.e., there is an injection from it to the final object. This property is preserved under the sequence of right adjoints $\mathtt{Reach} \circ \overline{U_{\mathcal{P}}} \circ \overline{F^{op}_{\mathcal{P}}} \circ E$. Furthermore, after application of $\mathtt{Reach}$, the automaton is also reachable. This means that applying the sequence $\mathtt{Reach} \circ \overline{U_{\mathcal{P}}} \circ \overline{F^{op}_{\mathcal{P}}} \circ E \circ \mathtt{Obs} \circ \overline{U^{op}_{\mathcal{P}}}$ to a non-deterministic automaton produces a deterministic and minimal one for the same language. We check for concluding that the sequence $\mathtt{Obs} \circ \overline{U^{op}_{\mathcal{P}}}$ is what is implemented by `codeterminize`, that the composite $\overline{F^{op}_{\mathcal{P}}} \circ E$ essentially transforms a backward deterministic observable automaton into a non-deterministic one, and that finally $\mathtt{Reach} \circ \overline{U_{\mathcal{P}}}$ is what is implemented by `determinize`. Hence, this indeed is Brzozowski's algorithm.

▶ Remark. The composite of the two adjunctions in (4) is almost the adjunction of [7, Corollary 9.2] upon noticing that the category $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}})$ of Set^{op}-automata accepting a language $\mathcal{L}_{\mathsf{Set}^{op}}$ is isomorphic to the opposite of the category $\mathsf{Auto}(\mathcal{L}^{\mathsf{rev}}_{\mathsf{Set}})$ of Set-automata that accept the reversed language seen as functor $\mathcal{L}_{\mathsf{Set}^{op}}$. This observation in turn can be proved using the symmetry of the input category $\mathcal{I}$.

6 Conclusion

In this paper we propose a view of automata as functors and we showed how to recast well understood classical constructions in this setting, and in particular minimization of subsequential transducers. The applications provided here are a small sample of many possible further extensions. We argue that this perspective gives a unified view of language recognition and syntactic objects. We can change the input category $\mathcal{I}$, so that we obtain monoids instead of automata, or more generally, other algebras as recognisers for languages. Minimization works out following the same recipe and yields the syntactic monoid (algebra) of a language. We can go beyond regular languages and obtain in this fashion the "syntactic space with an internal monoid" of a possibly non-regular language [13]. Our functorial treatment of automata is more general than that presented in [5] and it would be interesting to explore equations and coequations in this setting. We hope we can extend the framework to work with tree automata in monoidal categories. We discussed mostly NFA determinization, but we can obtain a variation of the generalized powerset construction [19] in this framework.

References

1 Jiří Adámek and Věra Trnková. *Automata and Algebras in Categories*. 37. Springer Netherlands, New York, 1989. URL: http://www.springer.com/fr/book/9780792300106.

2 Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A coalgebraic perspective on minimization and determinization. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'12, pages 58–73, Berlin, Heidelberg, 2012. Springer-Verlag.

3 Michael A. Arbib and Ernest G. Manes. Adjoint machines, state-behavior machines, and duality. *Journal of Pure and Applied Algebra*, 6(3):313 – 344, 1975. doi:10.1016/0022-4049(75)90028-6.

4 E. S. Bainbridge. Adressed machines and duality. In *Category Theory Applied to Computation and Control*, volume 25 of *Lecture Notes in Computer Science*, pages 93–98. Springer, 1974.

5 Adolfo Ballester-Bolinches, Enric Cosme-Llópez, and Jan J. M. M. Rutten. The dual equivalence of equations and coequations for automata. *Inf. Comput.*, 244:49–75, 2015.

6 Nick Bezhanishvili, Clemens Kupke, and Prakash Panangaden. *Minimization via Duality*, pages 191–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-32621-9_14.

7 Filippo Bonchi, Marcello M. Bonsangue, Helle Hvid Hansen, Prakash Panangaden, Jan J. M. M. Rutten, and Alexandra Silva. Algebra-coalgebra duality in Brzozowski's minimization algorithm. *ACM Trans. Comput. Log.*, 15(1):3:1–3:29, 2014.

8 Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Brzozowski's algorithm (co)algebraically. In *Logic and Program Semantics*, volume 7230 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 2012.

9 C. Cassidy, M. Hébert, and G. M. Kelly. Reflective subcategories, localizations and factorizationa systems. *Journal of the Australian Mathematical Society. Series A. Pure Mathematics and Statistics*, 38(3):287–329, 1985. doi:10.1017/S1446788700023624.

10 Christian Choffrut. A generalization of Ginsburg and Rose's characterization of G-S-M mappings. In *ICALP*, volume 71 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 1979.

11 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131 – 143, 2003. doi:10.1016/S0304-3975(01)00219-5.

12 Thomas Colcombet and Daniela Petrişan. Automata in glued vector spaces. submitted, 2017.

13 Mai Gehrke, Daniela Petrisan, and Luca Reggio. The Schützenberger product for syntactic spaces. In *ICALP*, volume 55 of *LIPIcs*, pages 112:1–112:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

14 J. A. Goguen. Minimal realization of machines in closed categories. *Bull. Amer. Math. Soc.*, 78(5):777–783, 09 1972. URL: http://projecteuclid.org/euclid.bams/1183533991.

15 Helle Hvid Hansen. Subsequential transducers: a coalgebraic perspective. *Inf. Comput.*, 208(12):1368–1397, 2010.

16 Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:62–222, 1997.

17 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. *Journal of Computer and System Sciences*, 81(5):859 – 879, 2015. 11th International Workshop on Coalgebraic Methods in Computer Science, {CMCS} 2012 (Selected Papers). doi:10.1016/j.jcss.2014.12.005.

18 Henning Kerstan, Barbara König, and Bram Westerbaan. Lifting adjunctions to coalgebras to (re)discover automata constructions. In *CMCS*, volume 8446 of *Lecture Notes in Computer Science*, pages 168–188. Springer, 2014.

19 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013.