Automata Minimization: a Functorial Approach*

Thomas Colcombet[1] and Daniela Petrişan[1]

1 **CNRS, IRIF, Univ. Paris-Diderot, Paris 7, France**
`{thomas.colcombet,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 instantiated to unify several phenomena in automata theory, starting with determinization and minimization (which have been previously studied from a coalgebraic and duality theoretic perspective). We also show how subsequential transducers can be seen as functors valued in a Kleisli category and explain Choffrut's minimization algorithm.

1998 ACM Subject Classification F.1.1

Keywords and phrases functor automata, minimization, category

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

There is a long tradition of interpreting results of automata theory using the lens of category theory. Typical instances of these scheme interpret automata as algebras (together with a final map) as put forward in [1, 3, 13], or as coalgebras (together with an initial map), see for example [15]. This dual narrative proved very useful [6] 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 language accepted by an automaton are adapted along the same pattern.

We provide several developments around this idea. First, we recall (see [11]) 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 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 languages in these categories (Lemma 3.2). This lifting accounts for several constructions in automata

* 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 Simon's Institute for the Theory of Computing where this work has been partly developped.

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 4.4) 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 of a final automaton, as well as, surprisingly, a factorization system.

The second concrete application is a proof of correctness of Brzozowski's minimization algorithm. Indeed, determinization of automata can be understood via a lifting of the Kleisli adjunction between the categories Rel (of sets and relations) and Set (of sets and functions); and reversing nondeterministic automata can be understood via a lifting of the self-duality of Rel. Brzozowski's minimization algorithm can be understood by lifting the adjunction between Set and its opposite category Set^{op}, thus recovering results from [6].

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 for example [5–7] and the references therein. The coalgebraic perspective on minimization was also emphasised in papers such as [2, 17]. Undertstanding determinization and codeterminization by lifting adjunctions to coalgebras was considered in [16], and is related to our results from Section 5.2. Subsequential transducers were understood coalgebraically in [14].

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 3.2, 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, and this is the common denominator of the different contributions of this 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 semantic of an automaton is to define 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 semantic of the automaton is to associate to each word $u = a_1 \dots a_n$ the map from 1 to 2 defined as $\mathit{final} \circ \delta_{a_n} \circ \dots \circ \delta_{a_n} \circ \mathit{init}$. 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 semantic of the automaton as a functor from the category spanned by the graph to 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^*.

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

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

$$\mathcal{O} \overset{\iota}{\hookrightarrow} \mathcal{I}\,.$$

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 $\mathcal{O} \overset{\iota}{\hookrightarrow} \mathcal{I}$. We obtain the following definition:

▶ **Definition 2.1** (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 following diagram 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:

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

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:

[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.

▶ **Lemma 2.2.** *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:*

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

Then

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

Proof. The proof essentially consists of a diagram:

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

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

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

We can express reachability $\mathtt{Reach}$ and obserbavility $\mathtt{Obs}$ 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 [8]. In our case, this is the reflective subcatgeory $\mathsf{Obs}(\mathcal{K})$ 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 2.3.** *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*

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

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 a final automaton for $\mathcal{L}$ – denoted by $\mathcal{A}^{init}(\mathcal{L})$, respectively $\mathcal{A}^{final}(\mathcal{L})$ – and, of a factorisation 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

$$\mathcal{A}^{init}(\mathcal{L}) \twoheadrightarrow \mathtt{Min}(\mathcal{L}) \rightarrowtail \mathcal{A}^{final}(\mathcal{L}) .$$

Lemma 2.2 implies that the minimal automaton divides any other automaton recognising the langauge, while Lemma 2.3 instantiates to the results of [6, Section 9.4].

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

In this section we provide sufficient conditions on $\mathcal{C}$ so that the category $\mathsf{Auto}(\mathcal{L})$ of $\mathcal{C}$-automata accepting a $\mathcal{C}$-language $\mathcal{L}$ satisfies the three coditions of Lemma 2.2.

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

The proof of Lemma 2.4 is the same as the classical one that shows that factorization systems can be lifted to functor categories.

▶ **Lemma 2.5.** *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 2.6.** *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 $\mathtt{Min}(L)$ accepting $\mathcal{L}$ is obtained via the factorization* $\mathsf{Lan}_\iota\mathcal{L} \twoheadrightarrow \mathtt{Min}(L) \rightarrowtail \mathsf{Ran}_\iota\mathcal{L}$.

▶ Remark. Depending on the category $\mathcal{I}$, we may relax the conditions in Corollary 2.6, see Lemma 3.1. Furthermore, we emphasise that these conditions are only sufficient. In Section 4 we consider the example of sequential transducers and we instantiate $\mathcal{C}$ with a Kleisli category. Although this category does not have powers, the final automaton exists.

3 Word Automata

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

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 on the right and where the composite of $\mathsf{states} \xrightarrow{w} \mathsf{states} \xrightarrow{w'} \mathsf{states}$ is given by the concatenation ww'.

Let $\mathcal{O}$ to 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 $\mathcal{L}$ an *$(\mathcal{C}, X, Y)$-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 $\mathcal{A}$ an *$(\mathcal{C}, X, Y)$-automaton*.

We can refine the statement of Corollary 2.6 as follows.

▶ **Lemma 3.1** (from [11]). *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.*

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

The initial automaton $\mathcal{A}^{init}(\mathcal{L})$ has as state space the copower $\coprod\limits_{u\in A^*} \mathcal{L}(\mathsf{in})$. The map

$$\varepsilon = \mathcal{A}^{init}(\mathcal{L})(\triangleright) \colon \mathcal{L}(\mathsf{in}) \to \coprod_{u\in A^*} \mathcal{L}(\mathsf{in})$$

is the coproduct injection corresponding to $\varepsilon \in A^*$. The map

$$L? = \mathcal{A}^{init}(\mathcal{L})(\triangleleft) \colon \coprod_{u\in A^*} \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\triangleleft)$$

is given on the component of the coproduct corresponding to $u \in A^*$ by $\mathcal{L}(\triangleright u \triangleleft)$. Lastly, for each $a \in A$ the map $\mathcal{A}^{init}(\mathcal{L})(a)$ is given on the component of the coproduct that corresponds to $u \in A^*$ as the coproduct injection corresponding to the word ua.

In [11] 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 $\mathcal{L}$ alon 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 $w \in A^*$.

For the final automaton, the proof follows by duality.

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 $\mathcal{D}$

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

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

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

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 3.2.** *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*

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

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 $(\mathcal{A}(\triangleleft))^* \colon F\mathcal{A}(\mathsf{states}) \to O$ of $\mathcal{A}(\triangleleft) \colon \mathcal{A}(\mathsf{states} \to GO$. In a diagram

$$\begin{tikzcd}
I \arrow[r, "\mathcal{A}(\triangleright)"] & \mathcal{A}(\mathsf{states}) \arrow[loop above, "\mathcal{A}(a)"] \arrow[r, "\mathcal{A}(\triangleleft)"] & GO & \arrow[r, mapsto, "\overline{F}"] & {} & FI \arrow[r, "F\mathcal{A}(\triangleright)"] & F\mathcal{A}(\mathsf{states}) \arrow[loop above, "F\mathcal{A}(a)"] \arrow[r, "(\mathcal{A}(\triangleleft))^*"] & O
\end{tikzcd}$$

The functor $\overline{G}$ is defined similarly on an $\mathcal{D}$-automaton $\mathcal{B}$.

$$\begin{tikzcd}
FI \arrow[r, "\mathcal{B}(\triangleright)"] & \mathcal{B}(\mathsf{states}) \arrow[loop above, "\mathcal{B}(a)"] \arrow[r, "\mathcal{B}(\triangleleft)"] & O & \arrow[r, mapsto, "\overline{G}"] & {} & I \arrow[r, "(\mathcal{B}(\triangleright))_*"] & G\mathcal{B}(\mathsf{states}) \arrow[loop above, "G\mathcal{B}(a)"] \arrow[r, "G(\mathcal{B}(\triangleleft))"] & GO
\end{tikzcd}$$

We show next that we have an isomorphism

$$\mathsf{Auto}(\mathcal{L}_\mathcal{D})(\overline{F}\mathcal{A}, \mathcal{B}) \cong \mathsf{Auto}(\mathcal{L}_\mathcal{C})(\mathcal{A}, \overline{G}\mathcal{B})$$

Indeed, consider a morphism $\alpha \colon \overline{F}\mathcal{A} \to \mathcal{B}$ in $\mathsf{Auto}(\mathcal{L}_\mathcal{D})$. We define a natural transformation $\alpha_* \colon \mathcal{A} \to \overline{G}\mathcal{B}$ by setting its component at states as the adjoint transpose $(\alpha_{\mathsf{states}})_*$ of

$$\alpha_{\mathsf{states}} \colon F\mathcal{A}(\mathsf{states}) \to \mathcal{B}(\mathsf{states})\,.$$

It is now easy to verify that α_* is indeed an automata morphism in $\mathsf{Auto}(\mathcal{L}_\mathcal{C})$ and that the mapping $\alpha \mapsto \alpha_*$ gives rise to the desired isomorphism. ◀

4 Choffrut's minimization of subsequential transducers

In [9, 10] 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 that this result can be established in the functorial framework of this paper.

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 4.4. 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 4.1.** A *subsequential transducer* is a tuple

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

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 either undefined and 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.

The subsequential transducer 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 each q_i is either undefined or belongs to Q, with q_0 inherited from the definition of T, and $q_i = q_{i-1} \cdot a_i$ for all $i = 1 \dots n$.

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 4.2.** 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:

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

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, i.e., the *Kleisli category* $\mathsf{Kl}(\mathcal{T})$ for $\mathcal{T}$, has as objects sets $X, Y, \dots$ and as *morphisms* $f\colon X \twoheadrightarrow Y$ functions $f\colon X \to B^* \times X + 1$ in Set (that is partial functions 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 [10] is specified by the following morphisms in $\mathsf{Kl}(\mathcal{T})$:

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

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 4.3.** *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 2.2. 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. Putting together all these results, we obtain:

▶ **Theorem 4.4** (Categorical version [9, 10])**.** *For all $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-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 2.2 describes what are the basic functions that have to be implemented, namely `Reach` and `Obs`.

The rest of this section is devoted to establish the three above mentionned 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 $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata and $(\mathsf{Set}, 1, B^*+1)$-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 system.

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 we have seen 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 2.4.

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 $\mathsf{Set}(1, B^* + 1)$.

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

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

Applying Lemma 3.2 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 diagram on the right.

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

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

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 3.1 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 $a \in A$

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

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 $a \in A$

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

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 4.5, in order to 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 4.5.** *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.*

Proof. Recall that we have the following two adjunctions for the categories $\mathsf{Kl}(\mathcal{T})$ of Kleisli algebras, respectively $\mathsf{EM}(\mathcal{T})$ of Eilenberg-Moore algebras, and the comparison functor $K\colon \mathsf{Kl}(\mathcal{T}) \to \mathsf{EM}(\mathcal{T})$ between them.

$$\begin{tikzcd} \mathsf{Kl}(\mathcal{T}) \arrow[rr, "K"] \arrow[dr, bend left, "U_{\mathcal{T}}"] & & \mathsf{EM}(\mathcal{T}) \arrow[dl, bend left, "U^{\mathcal{T}}"] \\ & \mathsf{Set} \arrow[ul, bend left, "F_{\mathcal{T}}"] \arrow[ur, bend left, "F^{\mathcal{T}}"] & \end{tikzcd} \tag{3}$$

The partial function $L\colon A^* \rightharpoonup B^*$ from Section 4.2 can also be modelled as an $(\mathsf{EM}(\mathcal{T}), T1, T1)$-langauge $\mathcal{L}_{\mathsf{EM}(\mathcal{T})}\colon \mathcal{O} \to \mathsf{EM}(\mathcal{T})$. Applying Lemma 3.2 for the adjunction $F^{\mathcal{T}} \dashv U^{\mathcal{T}}$ we obtain an adjunction $\overline{F^{\mathcal{T}}} \dashv \overline{U^{\mathcal{T}}}$ between the categories of $\mathsf{EM}(\mathcal{T})$-automata for $\mathcal{L}_{\mathsf{EM}(\mathcal{T})}$ and of Set-automata for $\mathcal{L}_{\mathsf{Set}}$. We also have a lifting $\overline{K}\colon \mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}) \to \mathsf{Auto}(\mathcal{L}_{\mathsf{EM}(\mathcal{T})})$ of the comparison functor K, which maps a $\mathsf{Kl}(\mathcal{T})$-auotmaton $\mathcal{A}$ to the $\mathsf{EM}(\mathcal{T})$-automaton $K \circ \mathcal{A}$. We obtain the following situation, which is just a lifting of diagram (3) to the categories of automata.

$$\begin{tikzcd} \mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}) \arrow[rr, "\overline{K}"] \arrow[dr, bend left, "\overline{U_{\mathcal{T}}}"] & & \mathsf{Auto}(\mathcal{L}_{\mathsf{EM}(\mathcal{T})}) \arrow[dl, bend left, "\overline{U^{\mathcal{T}}}"] \\ & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[ul, bend left, "\overline{F_{\mathcal{T}}}"] \arrow[ur, bend left, "\overline{F^{\mathcal{T}}}"] & \end{tikzcd}$$

One can readily check that the functor $\overline{U_{\mathcal{T}}}$ is the composite $\overline{U^{\mathcal{T}}} \circ \overline{K}$. The functor $\overline{K}$ is full and faithful (a property inherited from K) and thus reflects final objects. On the other hand, the final object in $\mathsf{Auto}(\mathcal{L}_{\mathsf{EM}(\mathcal{T})})$ can be computed using Lemma 3.1, since the underlying category $\mathsf{EM}(\mathcal{T})$ has all limits. Moreover, this final automaton is the reflection of the final Set-automaton $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})$. ◀

We are now ready to describe the final $\mathsf{Kl}(\mathcal{T})$-automaton. The final object in $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}})$ is the automaton $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})$ as described using Lemma 3.1. The functor $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})\colon \mathcal{I} \to \mathsf{Set}$

specified by

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

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

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

$$\begin{aligned}\varphi\colon B^* \times \mathsf{Irr}(A^*,B^*)+1 &\to (B^*+1)^{A^*}\\ (u,K) &\mapsto u \star K\\ \perp &\mapsto \kappa_\perp\ ,\end{aligned} \tag{4}$$

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 $(B^*+1)^{A^*}$ that maps $w \in A^*$ to $K(aw)$.

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

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

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

$$\begin{aligned}
&\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleright)\colon 1 \nrightarrow \mathsf{Irr}(A^*,B^*) && 0 \mapsto (\mathsf{lcp}(L), \mathsf{red}(L))\\
&\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(\triangleleft)\colon \mathsf{Irr}(A^*,B^*) \nrightarrow 1 && K \mapsto K(\varepsilon)\\
&\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})(a)\colon \mathsf{Irr}(A^*,B^*) \nrightarrow \mathsf{Irr}(A^*,B^*) && K \mapsto (\mathsf{lcp}(a^{-1}K), \mathsf{red}(a^{-1}K)) \text{ if } a^{-1}K \neq \kappa_\perp\\
& && K \mapsto \kappa_\perp \text{ if } a^{-1}K = \kappa_\perp
\end{aligned}$$

▶ **Lemma 4.6.** *The $\mathsf{Kl}(\mathcal{T})$-automaton $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$ is a final object in $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$.*

Proof. We show that $\overline{U_{\mathcal{T}}}(\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))$ is isomorphic to the final automaton $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})$. Indeed, at the level of objects the bijection between $\overline{U_{\mathcal{T}}}(\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))(\mathsf{states})$ and $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})(\mathsf{states})$ is given by the function φ defined in (4). It is easy to check that also on arrows $\overline{U_{\mathcal{T}}}(\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})}))$ is the same as $\mathcal{A}^{final}(\mathcal{L}_{\mathsf{Set}})$ 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 2.4 from a factorization system on $\mathsf{Kl}(\mathcal{T})$. This factorization system is obtained in turn from the regular-epi 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, for $\mathsf{Kl}(\mathcal{T})$ we define $\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ as the class of morphisms e in $\mathsf{Kl}(\mathcal{T})$ so that $U_{\mathcal{T}}e$ is surjective in Set, and $\mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ as the class of morphisms m in $\mathsf{Kl}(\mathcal{T})$ so that $U_{\mathcal{T}}m$ is injective in Set, and we prove that $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ is a factorization system on $\mathsf{Auto}(\mathcal{L}_{\mathsf{Kl}(\mathcal{T})})$.

Let us give a concrete characterisation of these classes of morphisms. For a word u in B^* we will denote by $\mathsf{suff}(u)$ the set of suffixes of u, that is,

$$\mathsf{suff}(u) = \{w \in B^* \mid \exists v \in B^*. u = vw\}\,.$$

We have that $f\colon X \nrightarrow Y$ (that is, $f\colon X \to B^* \times Y + 1$) is in $\mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ if and only if

$$\begin{cases} \forall x \in X.\ f(x) \neq \bot \\ \forall y \in Y.\ \forall x, x' \in X.\ f(x) = (u, y), f(x') = (u', y) \text{ and } u \in \mathsf{suff}(u') \text{ imply } x = x' \end{cases}$$

We have that $f\colon X \nrightarrow Y$ (that is, $f\colon X \to B^* \times Y + 1$) is in $\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ if and only if

$$\forall y \in Y. \exists x \in X. f(x) = (\varepsilon, y)$$

▶ **Lemma 4.7.** *$(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ is a factorization system on $\mathsf{Kl}(\mathcal{T})$.*

Proof. The interesting part of the proof is showing that any morphism $f\colon X \nrightarrow Y$ in $\mathsf{Kl}(\mathcal{T})$ can be factorised as a composite $m \circ e$ with $e \in \mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ and $m \in \mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$.

Given $f\colon X \nrightarrow Y$ in $\mathsf{Kl}(\mathcal{T})$, recall that $U_{\mathcal{T}}f\colon \mathcal{T}X \to \mathcal{T}Y$ is defined by $U_{\mathcal{T}}f(\bot) = \bot$ and for $(w, x) \in B^* \times X$ we have

$$U_{\mathcal{T}}f(w, x) = \begin{cases} (wu, y) & \text{if } f(x) = (u, y), \\ \bot & \text{if } f(x) = \bot, \end{cases}$$

We write $\mathrm{Im}(U_{\mathcal{T}}f)$ for the image of $U_{\mathcal{T}}f$ in Set, and we prove that $\mathrm{Im}(U_{\mathcal{T}}f)$ is isomorphic in Set to $B^* \times Z + 1$, where the set Z is defined as the subset of $\mathrm{Im}(U_{\mathcal{T}}f)$ such that the word on the first component of the pair is suffix-minimal. In a formula:

$$Z = \{(w, y) \in \mathrm{Im}(U_{\mathcal{T}}f) \mid \forall u \in B^*.\ u \in \mathsf{suff}(w) \Rightarrow (u, y) \notin \mathrm{Im}(U_{\mathcal{T}}f)\}$$

The next simple observation crucially uses the fact that we work with a free monoid B^*.

Fact. If $(u, y) \in \mathrm{Im}(U_{\mathcal{T}}f)$ then there exist unique words p_u, s_u in B^* so that $(s_u, y) \in Z$ and $u = p_u s_u$.

We can now factorise $f\colon X \nrightarrow Y$ as the composite in $\mathsf{Kl}(\mathcal{T})$

$$X \xrightarrow{e} Z \xrightarrow{m} Y\,,$$

where $e\colon X \to \mathcal{T}Y$ is defined by

$$e(x) = \begin{cases} (p_u, (s_u, y)) & \text{if } f(x) = (u, y) \\ \bot & \text{if } f(x) = \bot \end{cases}$$

while $m\colon Z \to \mathcal{T}Y$ is defined by $m((w,y)) = (w,y) \in B^* \times Y$ for all $(w,y) \in Z$. One can easily check that $m \circ e = f$ in $\mathsf{Kl}(\mathcal{T})$ and that $e \in \mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ and $m \in \mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$.

The fact that the two classes of morphisms $\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ and $\mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ are orthogonal follows from the fact that $U_{\mathcal{T}}$ factors through the full and faithful embedding K of $\mathsf{Kl}(\mathcal{T})$ into the category of Eilenberg-Moore algebras for $\mathcal{T}$. ◀

This completes the proof of Theorem 4.4.

5 Brzozowski's determinization 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, applies a powerset construction and at the same time restricts to the reachable states, yielding a deterministic automaton, and
- `transpose` is the operation that takes as input an 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 `transpose` $\circ$ `determinize` $\circ$ `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

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

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 codetermize it, that is, how to obtain a Set^{op}-automaton, all recognising 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 restriction to reachable states as in `determinize` and `codeterminize`) of a Rel-automaton can be seen as applications of Lemma 3.2 and are obtained by lifting the adjunctions between Set, Rel and Set^{op}.

$$\begin{tikzcd}
\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[r, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] \arrow[d] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Rel}}) \arrow[l, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[r, bend right, "\overline{F^{op}_{\mathcal{P}}}"'] \arrow[r, phantom, "\perp"] \arrow[d] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, bend right, "\overline{U^{op}_{\mathcal{P}}}"'] \arrow[d] \\
\mathsf{Set} \arrow[r, bend left, "F_{\mathcal{P}}"] \arrow[r, phantom, "\perp"] & \mathsf{Rel} \arrow[l, bend left, "U_{\mathcal{P}}"] \arrow[r, bend right, "F^{op}_{\mathcal{P}}"'] \arrow[r, phantom, "\perp"] & \mathsf{Set}^{op} \arrow[l, bend right, "U^{op}_{\mathcal{P}}"']
\end{tikzcd} \tag{5}$$

The adjunction between Set and Rel is the Kleisli adjunction for the powerset monad: $F_{\mathcal{P}}$ is identity on objects as 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 $\{y \in Y \mid \exists x \in X.(x,y) \in R\}$.

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 2.3 and (5) (that all correspond to equivalences at the level of languages):

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

A path in this diagram corresponds to a sequence of transformations of automata. It happens that when `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 `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 (5) is almost the adjunction of [6, Corollary 9.2] upon noticing that the category $\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}})$ 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 just 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 [12]. 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 [17] in this framework.

References

1 Jiří (ing) 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. URL: `http://dx.doi.org/10.1007/978-3-642-28729-9_4`, `doi:10.1007/978-3-642-28729-9_4`.

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. URL: `http://www.sciencedirect.com/science/article/pii/0022404975900286`, `doi:http://dx.doi.org/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 Nick Bezhanishvili, Clemens Kupke, and Prakash Panangaden. *Minimization via Duality*, pages 191–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: `http://dx.doi.org/10.1007/978-3-642-32621-9_14`, `doi:10.1007/978-3-642-32621-9_14`.

6 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.

7 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.

8 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`.

9 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.

10 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131 – 143, 2003. URL: `http://www.sciencedirect.com/science/`

article/pii/S0304397501002195, doi:http://dx.doi.org/10.1016/S0304-3975(01)00219-5.

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

12 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.

13 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.

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

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

16 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.

17 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.